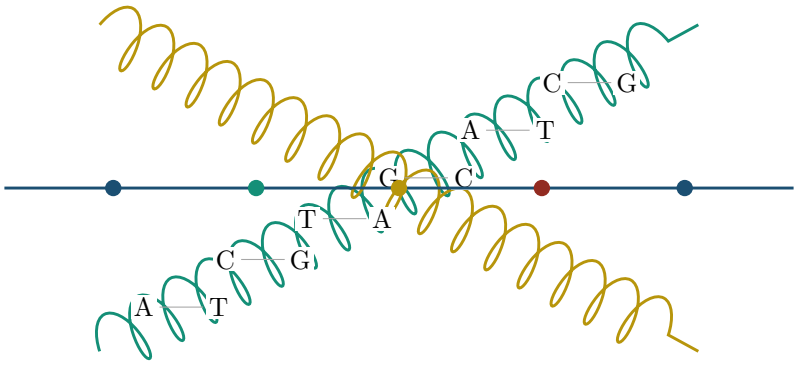
FROM DNA TO AGI

An Undergraduate Research Guide to DOGMA

Biological Computation, Non-Transformer Architecture, and Recursive Evaluation

Second Edition — Research Preview

Wenyan Qin



July 2026

Second Edition research preview, v2.1, July 2026.

This textbook was developed in conjunction with the EVO-TRAINER open-source project and the DOGMA (DNA-Organized Genomic Model Architecture) research program.

Typeset in LaTeX using Computer Modern and Latin Modern fonts.

Preface

This book begins with a research question: *Can principles from molecular information processing help us design useful learning systems that do not rely on transformer self-attention?* The answer is not known. The purpose of this book is to make the question precise enough to study.

Biological evolution encodes, regulates, expresses, and changes information. The central dogma of molecular biology—DNA → RNA → protein—describes allowed classes of information transfer [Crick, 1970]. It is a biological theory, not an AI blueprint. DOGMA asks which abstractions—persistent state, selective expression, local interaction, recurrence, and population selection—remain useful after we remove the chemistry.

DOGMA—the DNA-Organized Genomic Model Architecture—is a family of testable computational proposals. The current canonical implementation is a small byte-level sequence model built from overlapping local memory, causal convolution, state-space recurrence, bounded epigenetic memory, and sparse allosteric routing. It contains no transformer block and no self-attention. Other mechanisms in this book are historical designs or future hypotheses unless explicitly marked as implemented.

Who this book is for. This textbook is designed for upper-level undergraduate and graduate students in computer science, computational biology, bioinformatics, and artificial intelligence. We assume familiarity with introductory calculus, linear algebra, basic probability, and elementary programming. No prior knowledge of molecular biology is required—Part I builds all necessary biological foundations from scratch. Each concept is introduced step by step with worked examples, intuitive diagrams, and truth tables before proceeding to formal definitions. The book is structured to serve as the primary text for a two-semester course (undergraduate or graduate), though a focused one-semester course can be assembled from selected chapters (see the Reader's Roadmap).

How this book is organized. The book proceeds in six parts:

- **Part I: The Biological Computer** (Chapters 1–3) establishes DNA as an information system, surveys the history and state of the art of DNA computing, and reveals gene regulation as a sophisticated computational mechanism.
- **Part II: The Intelligence Stack** (Chapters 4–6) builds the AI/ML foundations needed for the rest of the book, from neural networks through transformers and large language models to biological foundation models.
- **Part III: The DOGMA Architecture** (Chapters 7–11) presents the core contribution: the six-stage DOGMA pipeline, HelixHash structural representation, Tera regime dynamics, TetraData structures, and the CodonForge DNA compiler.

- **Part IV: Building the Evolutor** (Chapters 12–14) covers evidence-gated recursive training, genomic data curricula, and the measured DOGMA research baseline.
- **Part V: Toward AGI** (Chapters 15–17) addresses scaling laws, post-transformer architectures, paths to artificial general intelligence, and the ethics and safety of genomic AI.
- **Part VI: Practicum** (Chapters 18–19) provides hands-on laboratory exercises and closes with a research-oriented epilogue.

Evidence labels. A *biological fact* is supported by primary literature. An *external result* belongs to the cited authors. An *implemented mechanism* exists in code. A *measured result* requires an identified checkpoint, data manifest, frozen evaluation version, and reproduction seeds. Everything else is a *hypothesis*. DOGMA is a research program, not AGI, and biological resemblance is not evidence of capability.

Acknowledgments. The DOGMA architecture and the EVO-TRAINER system emerged from a deep dialogue between computational biology, machine learning, and systems engineering. The author thanks the open-source community, the researchers whose work this book builds upon, and the students who will carry these ideas forward.

Wenyan Qin
July 2026

Reader's Roadmap

Ch.	Title	Key Topics
Part I: The Biological Computer		
1	The Code of Life	DNA structure, Central Dogma, information density
2	DNA Computing	Adleman, strand displacement, CRNs, molecular logic
3	Gene Regulation as Computation	Regulatory circuits, epigenetics, biological logic gates
Part II: The Intelligence Stack		
4	Machine Learning Foundations	Neural networks, optimization, representation learning
5	Transformers and Large Language Models	Attention, GPT, scaling laws, emergent abilities
6	Biological Foundation Models	AlphaFold, ESM-2, Evo, Enformer, genomic LMs
Part III: The DOGMA Architecture		
7	DOGMA: The Six-Stage Pipeline	Genomic bases, regulation, synthesis, transcription, expression, realization
8	HelixHash	Double-helix representation, motifs, bond matrices, novelty
9	Tera: The Hidden Regime State	Multi-objective pressure, mutation families, meta-policy
10	TetraData Structure & TetraMemory	3D computation, tetrahedral meshes, structured memory
11	CodonForge: The DNA Compiler	Codon opcodes, gene programs, SDANS compilation
Part IV: Building the Evolutor		
12	The Evolutionary Training Loop	Prompt genomes, mutation, selection, distillation
13	Foundation Models on Genomic Data	FASTA corpora, curriculum learning, helix-aware training
14	The Measured DOGMA Baseline	Checkpoints, perplexity, generation failures, next experiments
Part V: Toward AGI		
15	Scaling and Post-Transformer Architectures	Scaling laws, SSMs, hybrid architectures, DOGMA scaling
16	What Is AGI? Foundations and Paths	Definitions, biological arguments, DOGMA v3 roadmap
17	Ethics, Safety, and Responsibility	Dual-use, alignment, verifiability, governance
Part VI: Practicum		

Continued on next page

Ch.	Title	Key Topics
18	Hands-On: Building Your First DOGMA System	6 labs, capstone project
19	The Future We Are Building	Vision, open problems, research community

Suggested one-semester course: Chapters 1, 2, 4, 5, 7, 8, 12, 14, 16, 18.
Full two-semester sequence: All chapters in order, with Appendix A as prerequisite reading.

DOGMA in One Hour

The Four Layers

DOGMA becomes easier when four different subjects are kept separate.

1. **Molecular biology** studies how cells store and express information. DNA is transcribed into RNA; RNA can be translated into protein. Regulation determines what is expressed, where, and when.

2. **Molecular computing** constructs physical computations from molecules. Adleman encoded a graph problem in DNA [Adleman, 1994]; strand-displacement systems later implemented logic and neural-like circuits [Seelig et al., 2006, Qian and Winfree, 2011].

3. **Genomic machine learning** learns statistical structure from biological sequences. Transformers and state-space models are useful baselines here, but they are software models of sequence data, not molecular computers.

4. **DOGMA** is a software research architecture inspired by genomic organization. Its current canonical line deliberately avoids transformer self-attention. It asks whether local interaction, recurrence, selective expression, persistent bounded state, and evolutionary search can produce useful learning behavior.

What DOGMA Is Not

DOGMA is not DNA stored in a computer, not a wet-lab molecular computer, not a renamed transformer, and not AGI. It is an experimental architecture and evaluation program. "DNA-inspired" describes a source of hypotheses; it does not prove that the hypotheses work.

One Worked Example

Suppose the byte sequence `ACGTACGT` enters a small DOGMA model. Overlapping four-symbol windows are

$$(A, C, G, T),\ (C, G, T, A),\ (G, T, A, C),\ (T, A, C, G),\ (A, C, G, T).$$

Each window produces a local representation. A causal convolution mixes nearby windows; a recurrent state carries information forward; a gate chooses how strongly each state is expressed. The model predicts the next byte. This is a neural computation on digital symbols. No molecule is synthesized.

If the correct next byte is C and the model assigns it probability 0.5, the loss is

$$-\log(0.5) \approx 0.693.$$

Across N held-out bytes, average cross-entropy $\bar{\mathcal{L}}$ gives perplexity $\mathrm{PPL} = \exp(\bar{\mathcal{L}})$. Lower perplexity means better teacher-forced prediction on that split. It does *not* guarantee readable free generation, factual answers, reasoning, or general intelligence.

The Recursive Research Loop

A DOGMA candidate is never allowed to rewrite production weights directly. The controlled loop is:

$$\text{parent} \rightarrow \text{one bounded mutation} \rightarrow \text{train} \rightarrow \text{frozen probes} \rightarrow \text{archive or promote}.$$

The candidate must retain real-data anchors, avoid evaluation overlap, run with multiple fixed seeds, and pass separate sequence, explanation, and self-critique probes. A quality-diversity archive retains useful stepping stones; only a reproducible global improvement replaces the live checkpoint.

A Twelve-Week Route

Week	Reading	Question to answer
1	Chapters 1 and 2	How do biological and molecular computation differ?
2	Chapter 3	What does regulation compute?
3	Chapter 4; Appendix A	What do loss, gradient, and perplexity measure?
4	Chapter 5	What problem does self-attention solve?
5	Chapter 6	How are genomic foundation models evaluated?
6	Chapter 7	Which DOGMA mechanisms are canonical, historical, or hypothetical?
7	Chapters 8 and 10	How can local structure become a computational prior?
8	Chapters 9 and 11	How are regimes, types, and programs represented?
9	Chapter 12	Why separate exploration, archives, and promotion?
10	Chapters 13 and 14	Why can lower perplexity coexist with worse generation?
11	Chapters 16 and 17	What evidence would count as progress toward AGI?
12	Chapter 18	Reproduce one baseline and write a failure analysis.

Evidence Checklist

Before accepting any DOGMA claim, ask:

1. Is this a biological fact, an external result, an implementation statement, a measured DOGMA result, or a hypothesis?
2. Are the data source, split, checkpoint, evaluation version, and random seeds recorded?
3. Is the comparison matched for parameters, training bytes, and compute?

4. Does the evaluation test free generation and downstream behavior, not only teacher-forced likelihood?

5. Could contamination, cherry-picking, or synthetic-data feedback explain the result?

Starter Exercises

S.1. Explain in two sentences why the central dogma is a biological theory, not proof that DOGMA will work.

S.2. Compute perplexity for average losses 0.5, 1.0, and 2.0.

S.3. Design one probe for canonical DNA generation and one probe for a readable biological explanation. State exactly what counts as a pass.

S.4. Give an example of a candidate that belongs in the archive but should not replace the live model.

Contents

Preface i

Reader's Roadmap iii

DOGMA in One Hour v

I The Biological Computer 1

1 The Code of Life — DNA as an Information System 2
1.1 The Molecular Basis of Biological Information . . . 2
1.1.1 Step 1: The Atomic Ingredients . . . 2
1.1.2 Step 2: The Nucleoside — Sugar Plus Base . . . 3
1.1.3 Step 3: The Nucleotide — Adding the Phosphate . . . 4
1.1.4 Step 4: The Single Strand — A Polynucleotide Chain . . . 4
1.1.5 Step 5: The Double Helix — Watson–Crick Base Pairing . . . 5
1.1.6 DNA Structure: Summary of the Build-up . . . 7
1.2 DNA as Information Storage . . . 7
1.2.1 Information Density: Nature's Storage Medium . . . 7
1.2.2 Binary Encoding of DNA . . . 7
1.2.3 The Four-Letter Alphabet: Why Four? . . . 8
1.2.4 DNA Data Storage . . . 9
1.3 The Central Dogma of Molecular Biology . . . 9
1.3.1 Transcription: DNA to RNA, Step by Step . . . 10
1.3.2 The Genetic Code: A 64-to-21 Mapping . . . 11
1.3.3 Translation: The Ribosomal Decoder . . . 12
1.3.4 Post-Translational Modification: Runtime Polymorphism . . . 13
1.4 DNA as Information: A Quantitative Analysis . . . 13
1.4.1 Shannon Entropy of Genomic Sequences . . . 13
1.4.2 Redundancy in the Genetic Code . . . 14
1.4.3 Comparing DNA, Natural Language, and Code . . . 15
1.5 DNA Replication: Copying the Code . . . 15
1.5.1 Overview: Semi-Conservative Replication . . . 15
1.5.2 Step-by-Step Mechanism . . . 15
1.5.3 Error Correction: Proofreading and Repair . . . 16
1.6 DNA as a Storage and Computing Medium . . . 17
1.6.1 DNA Strand Displacement as a Computational Primitive . . . 17

1.6.2 Chemical Reaction Networks and Turing Completeness . . . 17
1.7 From Molecules to Architecture: The Abstraction Bridge . . . 18
1.7.1 The Genome as Source Code . . . 18
1.7.2 Why Previous Biological Metaphors in CS Were Shallow . . . 18
1.8 Worked Examples . . . 19
1.9 Exercises . . . 19
1.10 Key Takeaways . . . 21

2 A History of DNA Computing — From Adleman to Molecular Programming 22
2.1 Adleman's Breakthrough (1994) . . . 22
2.1.1 The Hamiltonian Path Problem . . . 22
2.1.2 Encoding Graphs in DNA . . . 23
2.1.3 Why It Worked—And Why It Didn't Scale . . . 24
2.2 DNA Logic Gates and Boolean Computation . . . 25
2.2.1 Binary Encoding in DNA . . . 25
2.2.2 The AND Gate . . . 25
2.2.3 The OR Gate . . . 27
2.2.4 The NOT Gate (Inverter) . . . 28
2.2.5 NAND and Universal Computation . . . 28
2.2.6 XOR Gate and Parity Detection . . . 29
2.3 DNA Arithmetic: Computing with Molecules . . . 29
2.3.1 The Half Adder . . . 29
2.3.2 The Full Adder . . . 30
2.3.3 DNA Subtraction via Two's Complement . . . 31
2.3.4 Comparison and Decision Making . . . 31
2.4 Strand Displacement Cascades . . . 32
2.4.1 The Three-Strand Mechanism . . . 32
2.4.2 Kinetic Control via Toehold Length . . . 33
2.4.3 Seesaw Gates: Scalable Digital Logic . . . 33
2.5 Restriction Enzyme Computing and Splicing Systems . . . 34
2.5.1 Restriction Enzymes as Programmable Cutters . . . 34
2.5.2 Splicing: DNA Recombination as Computation . . . 35
2.5.3 How Splicing Achieves Turing Completeness . . . 36
2.5.4 Rothemund's DNA Turing Machine (1996) . . . 38
2.5.5 Surface-Based DNA Computing . . . 39
2.6 Whiplash PCR: Hairpin-Based State Machines . . . 39
2.6.1 The Hairpin State Machine . . . 39
2.6.2 Computation Cycle . . . 40
2.7 DNA Nanotechnology and Structural Computing . . . 40
2.7.1 DNA Origami . . . 41
2.7.2 Tile Self-Assembly and Turing Completeness . . . 41
2.8 Chemical Reaction Networks: Turing-Complete Chemistry . . . 41
2.8.1 Formal Definition . . . 41
2.8.2 Computing with CRNs: Step-by-Step Examples . . . 42
2.8.3 Turing Completeness of CRNs . . . 43
2.8.4 Deterministic vs. Stochastic Semantics . . . 43
2.9 DNA Neural Networks . . . 43
2.9.1 Architecture . . . 43

2.9.2 How the Winner-Take-All Works . . . 44
2.9.3 Significance for DOGMA . . . 44
2.10 Molecular Programming Languages . . . 45
2.10.1 Visual DSD . . . 45
2.10.2 NUPACK . . . 45
2.10.3 The Compiler Analogy . . . 45
2.11 From Molecules to Silicon: The DOGMA Bridge . . . 45
2.12 The State of the Art and Open Challenges . . . 46
2.13 Exercises . . . 47
2.14 Key Takeaways . . . 48

3 Gene Regulation as Computation **50**
3.1 The Regulatory Grammar of DNA . . . 50
3.1.1 Promoters: The Entry Point . . . 50
3.1.2 Enhancers: Remote Amplifiers . . . 51
3.1.3 Silencers and Insulators . . . 51
3.2 Transcription Factor Binding: Step by Step . . . 52
3.2.1 How Transcription Factors Bind DNA . . . 52
3.2.2 Activators vs. Repressors . . . 52
3.2.3 Cooperative Binding and the Hill Function . . . 53
3.3 Transcription Factors as Logic Gates . . . 54
3.3.1 Combinatorial Regulation . . . 54
3.3.2 The Lac Operon: A Biological AND Gate . . . 54
3.4 Gene Regulatory Networks . . . 56
3.4.1 Network Motifs . . . 56
3.4.2 The Repressilator: A Biological Ring Oscillator . . . 56
3.4.3 The Toggle Switch: A Biological Flip-Flop . . . 57
3.4.4 GRNs as Programs . . . 58
3.5 Boolean Networks in Biology . . . 59
3.5.1 Kauffman's Random Boolean Networks . . . 59
3.5.2 Attractors as Cell Types . . . 60
3.6 Signal Transduction as Computation . . . 60
3.6.1 Kinase Cascades as Amplifiers . . . 60
3.6.2 The MAPK Pathway: A Signal Processing Pipeline . . . 61
3.6.3 Feedback Loops in Signaling . . . 61
3.7 Epigenetics: Computation That Modifies the Reader . . . 62
3.7.1 DNA Methylation . . . 62
3.7.2 Histone Modification . . . 63
3.7.3 The Histone Code . . . 63
3.7.4 Epigenetic Memory: State Without Sequence Change . . . 63
3.8 Alternative Splicing: One Genome, Many Programs . . . 64
3.8.1 The Mechanism . . . 64
3.8.2 Computational Significance . . . 64
3.9 The ENCODE Project: Quantifying the Regulatory Genome . . . 64
3.10 Synthetic Biology: Engineering Biological Circuits . . . 65
3.11 From Biological Regulation to DOGMA . . . 65
3.12 Exercises . . . 66
3.13 Key Takeaways . . . 68

II The Intelligence Stack **69**

4 Machine Learning Foundations for Biological AI **70**
4.1 The Learning Problem . . . 70
4.1.1 Supervised Learning as Function Approximation . . . 70
4.1.2 Loss Functions . . . 71
4.1.3 The Bias-Variance Tradeoff . . . 72
4.1.4 Unsupervised and Self-Supervised Learning . . . 73
4.2 Neural Networks from First Principles . . . 74
4.2.1 The Perceptron: A Single Neuron . . . 74
4.2.2 Activation Functions . . . 75
4.2.3 Multi-Layer Perceptrons (MLPs) . . . 76
4.3 Training Neural Networks . . . 77
4.3.1 Backpropagation: Computing Gradients Efficiently . . . 77
4.3.2 Gradient Descent Variants . . . 78
4.3.3 The Loss Landscape . . . 79
4.3.4 Regularization: Preventing Overfitting . . . 79
4.4 Convolutional Neural Networks . . . 81
4.4.1 The Convolution Operation . . . 81
4.4.2 Key CNN Concepts . . . 81
4.5 Representation Learning . . . 82
4.5.1 Embeddings: Mapping Symbols to Geometry . . . 82
4.5.2 The Geometry of Meaning . . . 83
4.5.3 Positional Encoding . . . 83
4.6 Sequence Models: From RNNs to State Spaces . . . 84
4.6.1 Recurrent Neural Networks . . . 84
4.6.2 Long Short-Term Memory (LSTM) . . . 84
4.6.3 State Space Models . . . 85
4.7 Evolutionary Algorithms . . . 86
4.7.1 The Genetic Algorithm: Step by Step . . . 86
4.7.2 Quality-Diversity Algorithms . . . 88
4.8 Multi-Objective Optimization . . . 89
4.8.1 Pareto Optimality . . . 89
4.8.2 Scalarization Methods . . . 90
4.9 Bringing It Together: The DOGMA Learning Stack . . . 90
4.10 Exercises . . . 91
4.11 Key Takeaways . . . 93

5 Transformers and Large Language Models **95**
5.1 The Attention Mechanism . . . 95
5.1.1 From Alignment to Attention . . . 95
5.1.2 Intuition: Queries, Keys, and Values . . . 96
5.1.3 Scaled Dot-Product Attention . . . 96
5.1.4 Why the Scaling Factor Matters . . . 97
5.1.5 Worked Example: Self-Attention with 3 Tokens . . . 97
5.1.6 Visualizing Self-Attention . . . 98
5.2 Multi-Head Attention . . . 99
5.2.1 Why Multiple Heads? . . . 99

5.2.2 Head Splitting and Concatenation . . . 99
5.2.3 What Different Heads Learn . . . 100
5.2.4 Multi-Head Attention Diagram . . . 100
5.3 The Transformer Architecture . . . 101
5.3.1 Encoder and Decoder Stacks . . . 101
5.3.2 Residual Connections and Layer Normalization . . . 101
5.3.3 The Complete Transformer Block . . . 102
5.4 Positional Encoding . . . 102
5.4.1 The Permutation Equivariance Problem . . . 102
5.4.2 Sinusoidal Positional Encodings . . . 102
5.4.3 Learned and Relative Position Encodings . . . 103
5.5 GPT and Autoregressive Language Models . . . 103
5.5.1 Decoder-Only Architecture . . . 103
5.5.2 The Next-Token Prediction Objective . . . 104
5.5.3 Causal Masking . . . 104
5.6 Training Large Language Models . . . 105
5.6.1 Training Data Scale . . . 105
5.6.2 Compute Requirements . . . 105
5.7 Scaling Laws for Language Models . . . 106
5.7.1 Power-Law Relationships . . . 106
5.7.2 Compute-Optimal Training (Chinchilla) . . . 106
5.8 Emergent Abilities of Large Language Models . . . 107
5.8.1 In-Context Learning . . . 107
5.8.2 Few-Shot, One-Shot, and Zero-Shot Learning . . . 108
5.8.3 Chain-of-Thought Reasoning . . . 108
5.8.4 Instruction Following . . . 109
5.9 Instruction Tuning and RLHF . . . 109
5.9.1 Supervised Instruction Tuning . . . 109
5.9.2 Reinforcement Learning from Human Feedback . . . 109
5.10 Key Models: A Comparative View . . . 110
5.10.1 GPT Family (OpenAI) . . . 110
5.10.2 BERT and Encoder-Only Models . . . 110
5.10.3 LLaMA and Open-Source Models . . . 111
5.10.4 Comparison Table . . . 111
5.11 Limitations of the Transformer Paradigm . . . 111
5.11.1 Quadratic Attention Cost . . . 111
5.11.2 No Persistent State . . . 112
5.11.3 No Native Typed Regions . . . 112
5.11.4 Output-First Bias . . . 112
5.11.5 No Evolutionary Self-Modification . . . 112
5.11.6 Summary: What Transformers Cannot Do That Biology Can . . . 113
5.12 Exercises . . . 113
5.13 Key Takeaways . . . 116

6 Biological Foundation Models **118**
6.1 What Is a Foundation Model? . . . 118
6.1.1 Definition and Core Idea . . . 118
6.1.2 An Analogy: The Medical Student . . . 119
6.1.3 Why Biology Is Special . . . 119
6.1.4 The Biological Modalities . . . 120
6.2 AlphaFold: Solving Protein Structure Prediction . . . 120
6.2.1 The Protein Folding Problem . . . 120
6.2.2 AlphaFold2: The Pipeline Step by Step . . . 121
6.2.3 The AlphaFold2 Loss Function . . . 123
6.2.4 AlphaFold2 Architecture Diagram . . . 124
6.2.5 Why AlphaFold2 Uses "Only" 93M Parameters . . . 124
6.3 ESM-2: The Language of Proteins . . . 125
6.3.1 Proteins as Language . . . 125
6.3.2 Training Objective: Masked Language Modeling . . . 125
6.3.3 Architecture and Scale . . . 126
6.3.4 Emergent Properties: Structure from Sequence . . . 126
6.3.5 ESM-2 Architecture Overview . . . 127
6.3.6 What ESM-2 Encodes . . . 127
6.4 DNABERT: Tokenizing and Learning from DNA . . . 129
6.4.1 The Challenge of DNA Tokenization . . . 129
6.4.2 Pre-Training DNABERT . . . 130
6.4.3 What DNABERT Can Predict . . . 130
6.5 Nucleotide Transformer: Scaling Across Species . . . 130
6.5.1 Multi-Species Pre-Training . . . 131
6.5.2 Cross-Species Transfer Learning . . . 131
6.5.3 Downstream Task Performance . . . 131
6.6 Evo: Genome-Scale DNA Language Modeling . . . 132
6.6.1 The Context Length Challenge . . . 132
6.6.2 State Space Models for DNA . . . 132
6.6.3 Single-Nucleotide Resolution . . . 133
6.6.4 Genome-Scale Generation . . . 133
6.7 Enformer: From DNA Sequence to Gene Expression . . . 134
6.7.1 The Gene Expression Prediction Task . . . 134
6.7.2 Architecture: Convolution Then Attention . . . 134
6.7.3 Long-Range Regulatory Interactions . . . 135
6.7.4 Limitations . . . 135
6.8 Generative Protein Design: RFdiffusion . . . 136
6.8.1 From Prediction to Design . . . 136
6.8.2 RFdiffusion . . . 136
6.8.3 The Inverse Folding Pipeline . . . 136
6.9 Comparison of Biological Foundation Models . . . 137
6.10 What Biology Teaches AI . . . 138
6.10.1 Symmetry as Architectural Prior . . . 138
6.10.2 Multi-Scale Structure . . . 138
6.10.3 Evolutionary Conservation as a Learning Signal . . . 138
6.10.4 The Generative Turn . . . 139
6.11 How DOGMA Differs: An Explicit Comparison . . . 139

6.11.1 DOGMA vs. AlphaFold2 . . . 139
6.11.2 DOGMA vs. ESM-2 . . . 139
6.11.3 DOGMA vs. DNABERT / Nucleotide Transformer . . . 139
6.11.4 DOGMA vs. Evo . . . 140
6.11.5 DOGMA vs. Enformer . . . 140
6.12 The Missing Piece: Why Current Models Are Domain-Specific . . . 140
6.12.1 The Fragmentation Problem . . . 140
6.12.2 The Central Dogma as Unifying Principle . . . 141
6.12.3 Why Unification Is Hard . . . 142
6.13 The Convergence with DOGMA . . . 142
6.13.1 The Genome as Structured Program . . . 142
6.13.2 Regulation as Conditional Computation . . . 142
6.13.3 Evolution as Optimization . . . 143
6.14 Exercises . . . 143
6.15 Key Takeaways . . . 145

III The DOGMA Architecture 147

7 DOGMA — The Six-Stage Pipeline 148
7.1 The DOGMA Thesis . . . 149
7.1.1 Why Current LLMs Are Structurally Limited . . . 149
7.2 The Genomic Base: DOGMA's Atomic Unit . . . 150
7.2.1 Anatomy of a Genomic Base . . . 151
7.2.2 Genomic Sequences and Regions . . . 152
7.2.3 Genomic Embedding: From Raw Bytes to the 4-Tuple Base Representation . 153
7.3 The Six-Stage Pipeline: Overview . . . 157
7.3.1 Stage 1: TetraMemory . . . 158
7.3.2 Stage 2: DNA Computing Block . . . 160
7.3.3 Stage 3: Regulation Gate . . . 163
7.3.4 Stage 4: Codon Translation . . . 164
7.3.5 Stage 5: Strand Attention . . . 166
7.3.6 Stage 6: Epigenetic + Allosteric Mechanisms . . . 167
7.4 The Expression Folder . . . 168
7.5 Full Architecture Diagram . . . 169
7.6 The DogmaBlock: Complete Forward Pass . . . 171
7.7 Architecture Comparison: DOGMA vs. Alternatives . . . 172
7.7.1 Complexity Summary . . . 172
7.8 Worked Example: Tracing Data Through All Six Stages . . . 172
7.9 The Evolutor State . . . 174
7.10 Why DOGMA Is Not Just a Modified Transformer . . . 175
7.11 Exercises . . . 175
7.12 Key Takeaways . . . 177

8 HelixHash — Structural Representation of Computational Genomes 179
8.1 The Double-Helix as a Data Structure . . . 179
8.1.1 Step-by-Step Construction of the Double Helix . . . 180
8.1.2 Watson–Crick Complementarity as a Hash Function . . . 181

8.1.3 Strand Alignment Score . . . 182
8.2 The Bond Matrix . . . 182
8.2.1 The 4×4 Hydrogen Bond Matrix . . . 182
8.2.2 Bond Energies as Attention Weights . . . 183
8.2.3 SantaLucia Nearest-Neighbor Parameters . . . 183
8.2.4 Worked Example: Computing ΔG° for ACGTACGT . . . 184
8.2.5 The Full Bond Matrix . . . 185
8.2.6 Bond Strength . . . 186
8.3 Motif Detection and Analysis . . . 187
8.3.1 Defining Genomic Motifs . . . 187
8.3.2 How HelixHash Detects Motifs . . . 187
8.3.3 Motif Frequency Spectrum . . . 187
8.3.4 Hash Function Design for DNA-Like Sequences . . . 188
8.4 HelixHash as Locality-Preserving Encoding . . . 188
8.4.1 Why Helical Representation Preserves Locality . . . 189
8.4.2 Comparison with Standard Positional Encodings . . . 189
8.4.3 Locality-Sensitive Hashing on Helical Structure . . . 189
8.5 Strand Sketches . . . 190
8.5.1 MinHash Lineage and Locality-Sensitive Hashing . . . 190
8.6 Novelty Detection . . . 191
8.6.1 Archive-Based Novelty . . . 191
8.6.2 Detecting Novel vs. Seen Patterns . . . 191
8.6.3 Cluster-Aware Selection . . . 192
8.6.4 Temporal Novelty Decay . . . 192
8.6.5 The Selection Objective . . . 192
8.7 HelixHash as Embedding . . . 193
8.7.1 Motif Embedding Layer . . . 193
8.7.2 Dual-Stream Architecture . . . 193
8.8 The HelixHash Algorithm . . . 194
8.9 Theoretical Properties . . . 194
8.10 Connection to the DOGMA Pipeline . . . 195
8.11 Exercises . . . 195
8.12 Key Takeaways . . . 197

9 Tera — The Hidden Regime State **199**
9.1 What Is a Regime State? . . . 199
9.1.1 Everyday Analogies . . . 199
9.1.2 Why Does DOGMA Need a Regime State? . . . 200
9.2 The Problem of Multi-Objective Evolution . . . 200
9.2.1 Why Single-Objective Optimization Fails . . . 200
9.2.2 The Three Core Pressures: Quality, Diversity, and Coherence . . . 201
9.2.3 The Pareto Landscape . . . 201
9.3 The Tera Regime State . . . 202
9.3.1 The Six Pressure Dimensions . . . 202
9.3.2 Visualizing the Pressure Space . . . 203
9.3.3 How Pressures Interact and Balance . . . 203
9.3.4 Exponential Memory Decay . . . 204
9.3.5 Regime Detection from Evaluation History . . . 205

9.4 Mutation Families . . . 205
9.4.1 What Are Mutation Families? . . . 206
9.4.2 The Six Mutation Families . . . 206
9.4.3 Adaptive Selection of Mutation Strategies . . . 207
9.4.4 Within-Family Operator Selection . . . 208
9.4.5 The Pressure Thermostat . . . 208
9.5 The Tera State Machine . . . 209
9.5.1 Formal Definition . . . 209
9.5.2 Transition Rules . . . 209
9.5.3 State Diagram . . . 210
9.5.4 Output Function: Mutation Distribution by State . . . 210
9.6 Meta-Policy: Learning Which Mutations Work Best . . . 211
9.6.1 The Problem of Starting from Scratch . . . 211
9.6.2 Cross-Run Persistent Learning . . . 211
9.6.3 Shared Regime Priors . . . 212
9.6.4 Comparison to Meta-Learning . . . 212
9.7 Connection to Biological Evolution . . . 213
9.7.1 Organisms Adapt to Environmental Pressures . . . 213
9.7.2 Multi-Scale Adaptation . . . 213
9.8 Pressure Landscape Visualization . . . 214
9.9 Worked Example: Five Evolutionary Steps . . . 215
9.9.1 Step 1: Discovery of a Grounding Problem . . . 215
9.9.2 Step 2: Grounding Problem Persists . . . 215
9.9.3 Step 3: Grounding Improves, Schema Problem Emerges . . . 215
9.9.4 Step 4: Regime Transition to Schema . . . 216
9.9.5 Step 5: Schema Takes Over . . . 216
9.10 Extended Mutation Family Catalog . . . 216
9.11 The Tera Double-Helix Complex . . . 217
9.11.1 Forward and Reverse Strand Dynamics . . . 219
9.12 Toward Post-Transformer Architectures . . . 219
9.13 Exercises . . . 220
9.14 Key Takeaways . . . 222

10 TetraData Structure and TetraMemory 224
10.1 Building a Tetrahedron from Scratch . . . 224
10.1.1 Step 0: A Single Vertex (0D) . . . 224
10.1.2 Step 1: Add a Second Vertex (1D — an Edge) . . . 225
10.1.3 Step 2: Add a Third Vertex (2D — a Triangle) . . . 225
10.1.4 Step 3: Add a Fourth Vertex (3D — the Tetrahedron!) . . . 226
10.1.5 The Construction at a Glance . . . 227
10.2 Why Tetrahedra? — A Deeper Look . . . 227
10.2.1 Tetrahedra vs. Triangles: A Detailed Comparison . . . 228
10.2.2 The Biological Justification . . . 228
10.3 Linking Tetrahedra — The LinkedTetraChain . . . 229
10.3.1 Adding Vertex 4: The Second Tetrahedron . . . 229
10.3.2 Adding Vertex 5: The Third Tetrahedron . . . 229
10.3.3 The General Pattern: LinkedTetraChain . . . 230
10.3.4 Growth Table . . . 231

10.4 DNA Mapping — The Key Insight . . . 231
10.4.1 Four Vertices, Four Bases . . . 231
10.4.2 Worked Example: Mapping 5′-ACGTACGT-3′ . . . 231
10.4.3 Why Not Triangles? An Information Density Argument . . . 232
10.4.4 The Six Edges: Pairwise Base Interactions . . . 233
10.5 The TetraData Structure (TDS) — Formal Definition . . . 233
10.5.1 Chirality . . . 234
10.5.2 Face-Sharing Adjacency . . . 234
10.6 36 Degrees of Freedom . . . 234
10.6.1 Detailed DOF Breakdown . . . 235
10.6.2 Comparison: Triangle vs. Tetrahedron DOF . . . 235
10.6.3 TetraTokenization . . . 235
10.7 TetraMemory: Structured Memory for DOGMA . . . 236
10.7.1 The Quadratic Attention Problem . . . 236
10.7.2 Face Propagation: Information Flow Through Shared Faces . . . 236
10.7.3 Volume as Attention Weight . . . 237
10.7.4 Edge Weights as Pairwise Compatibility . . . 237
10.7.5 Long-Range Information Flow . . . 237
10.7.6 Complexity Comparison . . . 238
10.8 The DoubleHelixMesh . . . 238
10.8.1 Sense and Antisense Strands . . . 238
10.8.2 Watson–Crick Cross-Links . . . 239
10.8.3 Advantages of the Double Helix Structure . . . 239
10.9 Delaunay Tetrahedralization . . . 240
10.10Multimodal Applications of TDS . . . 240
10.11Worked Examples . . . 241
10.12Exercises . . . 242
10.13Key Takeaways . . . 243

11 CodonForge — The DNA Compiler and SDANS **244**
11.1 Biological Foundations: Codon Translation in Nature . . . 245
11.1.1 The Central Dogma of Molecular Biology . . . 245
11.1.2 Step-by-Step: From mRNA to Amino Acid . . . 245
11.1.3 The Complete Biological Codon Table . . . 246
11.2 DNA as a Programming Language . . . 247
11.2.1 Opcode Families . . . 247
11.3 Gene Specifications . . . 250
11.3.1 Gene Structure in Detail . . . 251
11.3.2 Forward and Reverse Strands . . . 251
11.4 The Compilation Pipeline . . . 252
11.4.1 Stage 1: Parse . . . 253
11.4.2 Stage 2: Compile . . . 253
11.4.3 Stage 3: Optimize . . . 254
11.5 Worked Example: Compiling “What is DNA?” . . . 254
11.5.1 Stage 1: Parse — Building the Genomic IR . . . 254
11.5.2 Stage 2: Compile — Generating the Codon Program . . . 255
11.5.3 Stage 3: Optimize — Refining the Program . . . 256
11.6 Execution Semantics . . . 257

11.6.1 Example Execution Trace . . . 257
11.7 The Word2FASTA Bridge . . . 258
11.7.1 Why Encode Text as DNA? . . . 258
11.7.2 The Encoding Algorithm . . . 258
11.7.3 Complete Encoding Example: "Hi" . . . 259
11.7.4 Decoding: FASTA Back to Text . . . 260
11.8 SDANS: Strand Displacement Attention Neural System . . . 260
11.8.1 HelixSim and SDANS: Clarifying the Relationship . . . 260
11.8.2 Overview . . . 261
11.8.3 Motivation: Why Do We Need SDANS? . . . 261
11.8.4 How SDANS Maps Opcodes to Neural Operations . . . 261
11.8.5 Strand Displacement as Attention . . . 263
11.9 SDANS Compilation: Step-by-Step . . . 264
11.9.1 Step 1: Gene-to-Layer Mapping . . . 264
11.9.2 Step 2: Opcode-to-Operation Translation . . . 264
11.9.3 Step 3: Parameter Initialization from Codon Hints . . . 265
11.10 CodonForge vs. LLVM: A Compiler Comparison . . . 265
11.11 Connection to the DOGMA Architecture . . . 267
11.11.1 CodonForge in the DOGMA Pipeline . . . 267
11.11.2 Proposed CodonForge-to-Neural Compilation . . . 267
11.12 Advanced Topics . . . 268
11.12.1 Codon Optimization Strategies . . . 268
11.12.2 Error Detection and Correction . . . 268
11.12.3 Multi-Gene Regulation Patterns . . . 269
11.13 Exercises . . . 269
11.14 Key Takeaways . . . 271

IV Building the Evolutor **272**

12 Recursive, Evidence-Gated Training **273**
12.1 Three Loops, Three Responsibilities . . . 273
12.2 The Candidate Lifecycle . . . 274
12.3 Data Provenance and Model Collapse . . . 275
12.4 A Multi-Probe Evaluator . . . 275
12.5 Why Keep a Quality-Diversity Archive? . . . 276
12.6 Controlled Mutation . . . 276
12.7 Promotion Contract . . . 276
12.8 Connection to Self-Improving Systems . . . 277
12.9 Exercises . . . 277
12.10 Key Takeaways . . . 278

13 Foundation Models on Genomic Data **279**
13.1 What Is Genomic Data? . . . 279
13.1.1 From Molecules to Sequences . . . 279
13.1.2 The FASTA Format: Step by Step . . . 280
13.1.3 Beyond DNA: RNA and Protein Sequences . . . 281
13.1.4 Ambiguity Codes and Real-World Messiness . . . 281

13.2 Tokenizing Genomic Sequences . . . 281
13.2.1 Why Subword Tokenization Fails for Genomic Data . . . 281
13.2.2 Three Approaches to Genomic Tokenization . . . 282
13.2.3 Comparison Table: Tokenization Methods . . . 283
13.3 DOGMA's Unified Tokenization: CodonForge . . . 284
13.3.1 The CodonForge Vocabulary . . . 284
13.3.2 Adaptive Tokenization Logic . . . 285
13.3.3 Why Unified Tokenization Matters . . . 286
13.4 Byte-Level Tokenization as Universal Fallback . . . 286
13.5 The Bigram Baseline . . . 286
13.5.1 Character-Level Statistics of Genomic Sequences . . . 287
13.5.2 What Bigrams Reveal About Biology . . . 287
13.6 Pre-Training Objectives for Genomic Foundation Models . . . 288
13.6.1 Masked Language Modeling on DNA . . . 288
13.6.2 Next-Token Prediction on Codons . . . 289
13.6.3 Combining MLM and NTP . . . 289
13.7 Curriculum Learning for Genomic Data . . . 289
13.7.1 What Is Curriculum Learning? . . . 290
13.7.2 Why Curriculum Learning Helps . . . 290
13.7.3 DOGMA's Four-Stage Curriculum . . . 290
13.7.4 Stage Transitions and Mixing . . . 291
13.8 Training Data Pipeline: GenBank to Batching . . . 291
13.8.1 Data Sources . . . 291
13.8.2 Preprocessing Steps . . . 292
13.8.3 Structured Rendering . . . 292
13.8.4 Curriculum Assignment and Batching . . . 292
13.9 TinyGPT: Minimal Transformer for Genomic Sequences . . . 293
13.9.1 Decoder-Only Architecture Adapted for DNA . . . 293
13.9.2 Byte-Level Embedding with Positional Encoding . . . 294
13.9.3 Training on Mixed Text/Genome Corpora . . . 294
13.10 Helix-Aware Training . . . 295
13.10.1 Why the Double Strand Matters for Training . . . 295
13.10.2 Dual-Stream Architecture . . . 295
13.10.3 Reverse Complement Consistency Loss . . . 296
13.10.4 Motif-Level Feature Channels . . . 296
13.10.5 Performance Comparison: Standard vs. Helix-Aware . . . 296
13.11 Multi-Scale TetraMemory During Pretraining . . . 297
13.11.1 What Is TetraMemory? . . . 297
13.11.2 How TetraMemory Integrates with Training . . . 297
13.11.3 Multi-Scale Attention Patterns . . . 298
13.12 Cross-Modal Training . . . 298
13.12.1 The Four Modalities . . . 299
13.12.2 Cross-Modal Training Objectives . . . 299
13.13 Evaluation of Genomic Foundation Models . . . 299
13.13.1 Intrinsic Metrics . . . 300
13.13.2 Downstream Benchmarks . . . 300
13.13.3 Required Comparison with Existing Models . . . 300
13.14 Proposed Large-Scale DOGMA Configuration . . . 302

13.14.1 From Foundation Model to DOGMA Pipeline . . . 302
13.14.2 Training Configuration Alignment . . . 302
13.14.3 Transfer Learning Strategy . . . 302
13.15 DOGMA for Multimodal Intelligence . . . 303
13.15.1 TetraMesh: Converting 3D Structures to Byte-Level Tokens . . . 303
13.15.2 Image to DNA Encoding: A Detailed Pipeline . . . 304
13.15.3 Video to DNA Encoding . . . 308
13.15.4 Audio to DNA Encoding . . . 309
13.15.5 Cross-Modal Generation . . . 310
13.15.6 The Multimodal Corpus Builder . . . 312
13.15.7 Multi-Express: One Genome, Many Realizations . . . 312
13.15.8 Visual Overviews . . . 313
13.15.9 Use Cases for DOGMA Multimodal Intelligence . . . 314
13.16 DOGMA for Conversational AI . . . 317
13.16.1 The ChatSession Architecture . . . 317
13.16.2 Central Dogma Context Enrichment . . . 317
13.16.3 Realization Focus for Conversational Output . . . 318
13.16.4 Measured ATCG-to-Language Bridge . . . 319
13.16.5 Streaming Generation . . . 320
13.17 Exercises . . . 321
13.18 Key Takeaways . . . 323

14 The Measured DOGMA Baseline **325**
14.1 Implemented Architecture . . . 325
14.2 Data Snapshot . . . 326
14.3 What the Cycles Measured . . . 327
14.4 A Separate Hybrid Result . . . 327
14.5 Why Perplexity and Generation Disagreed . . . 328
14.6 The v2 Correction . . . 328
14.7 Evaluation Program . . . 329
14.7.1 Tier 1: Software Correctness . . . 329
14.7.2 Tier 2: Controlled Sequence Tasks . . . 329
14.7.3 Tier 3: Genomic Benchmarks . . . 329
14.7.4 Tier 4: Recursive-Learning Evidence . . . 329
14.8 Near-Term Experiments . . . 330
14.9 Exercises . . . 330
14.10 Key Takeaways . . . 331

V Toward AGI **332**

15 Scaling Laws and Post-Transformer Architectures **333**
15.1 Neural Scaling Laws . . . 333
15.1.1 The Kaplan Scaling Laws . . . 333
15.1.2 The Chinchilla Correction . . . 334
15.1.3 Visualizing Scaling Laws . . . 335
15.1.4 Emergent Abilities . . . 335
15.2 The Limits of Dense Scaling . . . 336

15.2.1 Quadratic Attention Cost . . . 336
15.2.2 Energy and Environmental Cost . . . 336
15.2.3 Data Exhaustion . . . 337
15.2.4 Capability Plateaus . . . 337
15.3 State Space Models: From S4 to Mamba . . . 337
15.3.1 The Continuous-Time State Space . . . 337
15.3.2 S4: Structured State Spaces for Sequences . . . 338
15.3.3 Mamba: Selective State Spaces . . . 339
15.3.4 The Hardware-Aware Scan Algorithm . . . 340
15.4 RWKV and Linear Attention . . . 341
15.4.1 The Quadratic Bottleneck in Attention . . . 341
15.4.2 Linear Attention via Kernel Decomposition . . . 341
15.4.3 RWKV: Receptance Weighted Key Value . . . 341
15.5 Mixture of Experts (MoE) . . . 342
15.5.1 The MoE Architecture . . . 342
15.5.2 Expert Parallelism and Load Balancing . . . 343
15.5.3 Expert Specialization . . . 343
15.6 How DOGMA Scales . . . 343
15.6.1 The Four Scaling Axes of DOGMA . . . 344
15.6.2 Sample Efficiency Through Biological Priors . . . 344
15.6.3 Why Four-Path Diversity Helps Scaling . . . 345
15.6.4 Parameter Allocation: DOGMA vs. Transformer . . . 345
15.6.5 Per-Block FLOP Analysis . . . 346
15.6.6 Complexity Analysis . . . 347
15.7 Hybrid Architectures . . . 348
15.7.1 Attention + SSM Hybrids . . . 348
15.7.2 Attention + MoE Hybrids . . . 348
15.7.3 Where DOGMA Fits in the Hybrid Landscape . . . 348
15.8 Comprehensive Architecture Comparison . . . 349
15.9 Worked Example: Comparing Architectures on a Long Sequence . . . 349
15.10Exercises . . . 350
15.11Key Takeaways . . . 351

16 Paths Toward AGI and Post-Transformer Intelligence 353
16.1 What Is Artificial General Intelligence? . . . 353
16.1.1 Definitions and the Capability Spectrum . . . 353
16.1.2 Narrow AI vs. General AI: A Comparison . . . 354
16.1.3 Why Current LLMs Are Broad but Not General . . . 355
16.2 The Scaling Hypothesis . . . 356
16.2.1 What the Scaling Hypothesis Claims . . . 356
16.2.2 Chinchilla and Compute-Optimal Training . . . 356
16.2.3 Arguments For the Scaling Hypothesis . . . 357
16.2.4 Arguments Against the Scaling Hypothesis . . . 357
16.3 Emergent Abilities in Large Models . . . 357
16.3.1 What Is Emergence? . . . 357
16.3.2 Step-by-Step Examples of Emergence . . . 358
16.3.3 Are Emergent Abilities Real? . . . 359
16.4 Why Transformers May Not Be Enough . . . 359

16.4.1 The Quadratic Attention Bottleneck . . . 359
16.4.2 Reasoning Limitations . . . 360
16.4.3 The World Model Problem . . . 360
16.5 Post-Transformer Architectures . . . 360
16.5.1 State Space Models: Mamba and Beyond . . . 361
16.5.2 Mixture of Experts . . . 361
16.5.3 Liquid Neural Networks . . . 362
16.5.4 Neuromorphic Computing . . . 362
16.6 DOGMA as a Post-Transformer Candidate . . . 362
16.6.1 Addressing the Compute Wall: Regulated Propagation . . . 362
16.6.2 Addressing the Reasoning Wall: Dynamic Regulation Depth . . . 363
16.6.3 Addressing the Grounding Wall: Structured State as World Model . . . 363
16.6.4 Compositional Verification Through Tetrahedral Structure . . . 363
16.7 The Biological Precedent for AGI . . . 364
16.7.1 Evolution Produced General Intelligence . . . 364
16.7.2 Three Timescales of Biological Intelligence . . . 364
16.7.3 Genomic Organization vs. Flat Sequence Prediction . . . 365
16.7.4 Multi-Scale Organization Mirrors Hierarchical Reasoning . . . 365
16.8 The Alignment Problem . . . 365
16.8.1 Instrumental Convergence . . . 366
16.8.2 Goodhart's Law and Specification Gaming . . . 366
16.8.3 Scalable Oversight . . . 367
16.9 Architecture Comparison . . . 367
16.10 The DOGMA Roadmap Toward AGI . . . 367
16.10.1 Phase 1: Epigenetic Memory Layer . . . 367
16.10.2 Phase 2: Ribosomal Translation . . . 370
16.10.3 Phase 3: Allosteric Regulation . . . 370
16.10.4 Phase 4: Chromosome-Scale Hierarchy and Horizontal Gene Transfer . . . 370
16.11 Open Problems and Challenges . . . 371
16.12 A Vision for the Future . . . 371
16.13 Exercises . . . 372
16.14 Key Takeaways . . . 374

17 Ethics, Safety, and the Responsibility of Intelligence 376
17.1 Why AI Safety Matters Now . . . 376
17.1.1 The Capability–Alignment Gap . . . 377
17.1.2 Concrete Examples of AI Failures . . . 377
17.2 The Alignment Problem in Depth . . . 378
17.2.1 Outer Alignment . . . 378
17.2.2 Inner Alignment and Mesa-Optimizers . . . 379
17.2.3 Alignment in Evolutionary Systems . . . 379
17.2.4 Multi-Objective Optimization as Implicit Alignment . . . 380
17.3 RLHF and Constitutional AI . . . 380
17.3.1 RLHF: Step by Step . . . 380
17.3.2 Constitutional AI . . . 381
17.3.3 Limitations of Current Alignment Techniques . . . 381
17.4 Interpretability and Mechanistic Transparency . . . 382
17.4.1 Feature Visualization . . . 382

17.4.2 Circuit Analysis . . . 382
17.4.3 DOGMA's Transparency Advantages . . . 383
17.5 DOGMA's Safety Advantages: Biological Inspiration . . . 385
17.5.1 The Immune System: Detecting and Neutralizing Threats . . . 385
17.5.2 Homeostasis: Maintaining Stability Under Perturbation . . . 385
17.5.3 Regulatory Networks: Constitutive and Inducible Safety . . . 385
17.6 Dual-Use Concerns: DNA Computing and Biosecurity . . . 386
17.6.1 Where Computational Biology Meets Synthetic Biology . . . 387
17.6.2 Risk Taxonomy for Genomic AI . . . 388
17.6.3 Responsible Disclosure . . . 388
17.7 Fairness, Bias, and Representation . . . 389
17.7.1 Sources of Bias . . . 389
17.7.2 Mitigation Strategies . . . 389
17.8 Environmental Impact . . . 390
17.8.1 Compute Costs and Carbon Footprint . . . 390
17.8.2 Efficiency as an Ethical Imperative . . . 391
17.9 Governance and Regulation . . . 391
17.9.1 AI Governance Frameworks . . . 391
17.9.2 Biosafety Governance . . . 392
17.9.3 Toward Integrated AI-Biosafety Governance . . . 392
17.10 The Bio-Digital Ethics Boundary . . . 393
17.10.1 Intellectual Property and Biological Inspiration . . . 393
17.10.2 The Status of Computational Organisms . . . 393
17.10.3 Informed Consent in Bio-Digital Research . . . 393
17.11 Responsible Development Principles . . . 394
17.11.1 Incremental Capability with Safety Checkpoints . . . 394
17.11.2 Red-Teaming Genomic AI Systems . . . 394
17.11.3 Community Oversight and Peer Review . . . 395
17.11.4 Ethical Review for Evolutionary Experiments . . . 395
17.12 A Worked Example: Safety-Augmented Evolutionary Training . . . 395
17.13 Exercises . . . 396
17.14 Key Takeaways . . . 399

VI Practicum **401**

18 Hands-On Labs — Building DOGMA from Scratch **402**
18.1 Environment Setup . . . 402
18.2 Lab 1: Implementing Watson-Crick Base Pairing in Code . . . 403
18.2.1 Learning Objectives . . . 403
18.2.2 Background . . . 403
18.2.3 Step-by-Step Instructions . . . 403
18.2.4 Expected Output . . . 405
18.2.5 Discussion Questions . . . 406
18.3 Lab 2: Building a Toehold-Mediated Strand Displacement Simulator . . . 406
18.3.1 Learning Objectives . . . 406
18.3.2 Background . . . 406
18.3.3 Step-by-Step Instructions . . . 407

18.3.4 Expected Output . . . 409
18.3.5 Discussion Questions . . . 409
18.4 Lab 3: Implementing a DNA Logic Gate (AND) as a CRN . . . 409
18.4.1 Learning Objectives . . . 409
18.4.2 Background . . . 410
18.4.3 Step-by-Step Instructions . . . 410
18.4.4 Expected Output . . . 412
18.4.5 Discussion Questions . . . 412
18.5 Lab 4: Building TetraMemory from Scratch . . . 412
18.5.1 Learning Objectives . . . 412
18.5.2 Background . . . 413
18.5.3 Step-by-Step Instructions . . . 413
18.5.4 Expected Output . . . 416
18.5.5 Discussion Questions . . . 417
18.6 Lab 5: Implementing the Thermodynamic Attention Mechanism . . . 417
18.6.1 Learning Objectives . . . 417
18.6.2 Background . . . 418
18.6.3 Step-by-Step Instructions . . . 418
18.6.4 Expected Output . . . 422
18.6.5 Discussion Questions . . . 422
18.7 Lab 6: Training a Minimal DOGMA Model . . . 423
18.7.1 Learning Objectives . . . 423
18.7.2 Background . . . 423
18.7.3 Step-by-Step Instructions . . . 423
18.7.4 Expected Output . . . 427
18.7.5 Discussion Questions . . . 428
18.8 Lab 7: Comparing DOGMA vs. Transformer on a Genomic Sequence Task . . . 428
18.8.1 Learning Objectives . . . 428
18.8.2 Background . . . 428
18.8.3 Step-by-Step Instructions . . . 429
18.8.4 Expected Output . . . 432
18.8.5 Discussion Questions . . . 433
18.9 Lab 8: Visualizing DOGMA's Internal Representations . . . 433
18.9.1 Learning Objectives . . . 433
18.9.2 Background . . . 433
18.9.3 Step-by-Step Instructions . . . 434
18.9.4 Expected Output . . . 440
18.9.5 Discussion Questions . . . 441
18.10 Putting It All Together: The Lab Progression . . . 441
18.11 Key Takeaways . . . 443

19 Epilogue: The Future of Intelligence **444**
19.1 The Journey from DNA to DOGMA . . . 444
19.1.1 A Molecule That Changed Everything . . . 444
19.1.2 From Biological Principles to Computational Architecture . . . 445
19.2 What DOGMA Teaches Us About Intelligence . . . 446
19.2.1 Intelligence Is Substrate-Independent . . . 446
19.2.2 Evolution Found Solutions We Are Only Now Discovering . . . 446

19.2.3 Regulation Is as Important as Computation . . . 447
19.3 The Central Dogma Revisited: From Biology to DOGMA . . . 448
19.3.1 Crick's Original Formulation . . . 448
19.3.2 The DOGMA Pipeline: A Computational Analog . . . 448
19.3.3 What the Mapping Reveals . . . 450
19.4 The Convergence of Biology and Computation . . . 450
19.4.1 Wet Computing: DNA as a Computational Substrate . . . 450
19.4.2 DNA Data Storage . . . 451
19.4.3 Synthetic Biology and Programmable Life . . . 451
19.4.4 In Vivo Computing: The Cell as a Computer . . . 452
19.4.5 The Horizon of Hybrid Systems . . . 452
19.5 Open Research Frontiers . . . 453
19.5.1 Open Problem 1: Scaling Evolutionary Synthesis . . . 453
19.5.2 Open Problem 2: Formal Theory of Regulatory Attention . . . 453
19.5.3 Open Problem 3: Cross-Modal Genomic Representations . . . 454
19.5.4 Open Problem 4: Interpretability Through Biological Analogy . . . 454
19.5.5 Open Problem 5: Energy-Efficient Inference via Biological Sparsity . . . 454
19.5.6 Open Problem 6: Developmental Programs for AI . . . 455
19.5.7 Open Problem 7: Bridging Genomic Foundation Models and DOGMA . . . 455
19.6 The Next Decade of Post-Transformer AI . . . 455
19.6.1 Beyond Bigger Transformers . . . 455
19.6.2 Predictions and Possibilities . . . 456
19.6.3 What We Are Not Predicting . . . 456
19.6.4 The Role of Hardware Co-Evolution . . . 457
19.7 What Makes Intelligence "General"? . . . 457
19.7.1 Beyond Benchmark Scores . . . 457
19.7.2 Generality as Architectural Property . . . 458
19.7.3 Intelligence as Understanding, Not Just Performance . . . 458
19.8 Honest Assessment: Where We Stand . . . 459
19.8.1 What Has Been Demonstrated . . . 459
19.8.2 What Remains . . . 459
19.8.3 The Community We Need . . . 460
19.9 A Letter to the Next Generation . . . 460
19.9.1 The Importance of Interdisciplinary Thinking . . . 460
19.9.2 What You Can Do Now . . . 461
19.9.3 The Moment You Are Entering . . . 461
19.10Key Takeaways: The Entire Book in Review . . . 462
19.11Final Reflection . . . 463

A Mathematical Foundations **464**
A.1 Functions, Vectors, and Tensors . . . 464
A.2 Probability and Information . . . 464
A.3 Gradients and Optimization . . . 465
A.4 Convolution and Recurrence . . . 465
A.5 Gates and Bounded Memory . . . 465
A.6 Multi-Objective Evaluation . . . 466
A.7 Uncertainty and Reproduction . . . 466
A.8 Complexity . . . 466

A.9 Review Problems 466

B Canonical DOGMA Formal Specification **467**
B.1 Model Contract 467
B.2 Prohibited Canonical Components 467
B.3 Checkpoint Contract 468
B.4 Corpus Contract 468
B.5 Recursive-Cycle Contract 468
B.6 Evaluation Contract 469
B.7 Archive and Promotion 469
B.8 Claim Contract 469

C Glossary **470**

Part I

The Biological Computer

Chapter 1

The Code of Life — DNA as an Information System

The secret of life is that it is written in a language we can read but did not invent.

— After Francis Crick

In April 1953, James Watson and Francis Crick published a nine-hundred-word paper that changed every science it touched [Watson and Crick, 1953]. The double-helical structure of deoxyribonucleic acid (DNA) revealed that hereditary information is encoded in a molecular sequence—a string over a four-letter alphabet. This discovery did more than solve a problem in biochemistry. It established that life is, at its deepest level, an *information system*: one that stores, copies, reads, edits, and evolves structured programs written in chemistry.

For the computer scientist reading this book, the implications are immediate. If life encodes information in sequences, then the mathematical frameworks of formal languages, information theory, and computation apply directly. If cells regulate gene expression through combinatorial logic, then biological circuits are a form of programming. And if evolution optimizes genomic programs through mutation and selection, then Darwin's algorithm is the oldest and most successful optimizer on Earth.

This chapter establishes the biological foundations on which the entire DOGMA architecture rests. We begin from the very atoms that compose DNA, build up through nucleotides, single strands, and the double helix. We then walk through the Central Dogma of molecular biology step by step, present the full codon table, examine the information-theoretic properties of genomic sequences, and finish with the mechanics of DNA replication. Every concept is developed with explicit diagrams, worked examples, and exercises designed for an undergraduate audience.

1.1 The Molecular Basis of Biological Information

1.1.1 Step 1: The Atomic Ingredients

All of molecular biology is built from a surprisingly small set of atoms. DNA uses only five elements:

Element	Symbol	Role in DNA
Carbon	C	Backbone of sugar and bases
Hydrogen	H	Bonds, structural filler
Oxygen	O	Sugar ring, phosphate group
Nitrogen	N	Part of every nitrogenous base
Phosphorus	P	Phosphodiester backbone

These five elements combine to form three molecular building blocks, which we now assemble step by step.

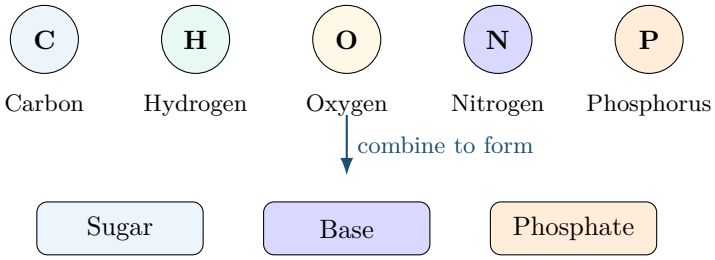


Figure 1.1: The five elements of DNA combine to form three molecular building blocks: a sugar, a nitrogenous base, and a phosphate group.

1.1.2 Step 2: The Nucleoside — Sugar Plus Base

A **nucleoside** is formed when a nitrogenous base attaches to a five-carbon sugar called *deoxyribose* (in DNA) or *ribose* (in RNA). The base bonds to the $1'$ carbon of the sugar via a glycosidic bond.

There are four nitrogenous bases in DNA, divided into two chemical families:

- **Purines** (two-ring structures): **Adenine (A)** and **Guanine (G)**.
- **Pyrimidines** (single-ring structures): **Cytosine (C)** and **Thymine (T)**.

Remark 1.1. A useful mnemonic: the pyrimidines (**C**ytosine and **T**hymine) have names that include the letter "y", just like p**y**rimidine. The purines (Adenine, Guanine) do not.

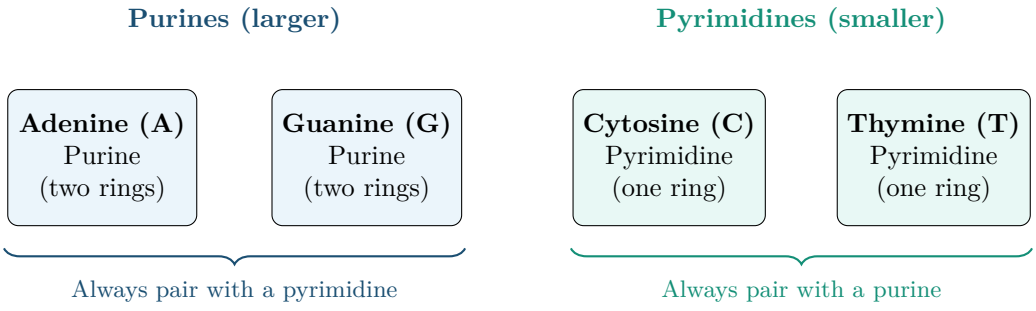


Figure 1.2: The four DNA bases grouped by chemical family. A purine always pairs with a pyrimidine, keeping the helix diameter constant.

The key structural insight is that a purine–pyrimidine pair always has the same total width (roughly 1.08 nm), which keeps the double helix at a uniform diameter. This is why A pairs with T and G pairs with C—not the other way around.

1.1.3 Step 3: The Nucleotide — Adding the Phosphate

A **nucleotide** is formed when a phosphate group attaches to the 5′ carbon of a nucleoside's sugar. The nucleotide is the fundamental monomer of DNA—the single "bead" from which the long strand is built. Its three components are:

1. A **phosphate group** (PO_4^{3-}) — carries negative charge, forms the backbone.
2. A **deoxyribose sugar** — the five-carbon ring that connects base to backbone.
3. A **nitrogenous base** (A, T, C, or G) — carries the genetic information.

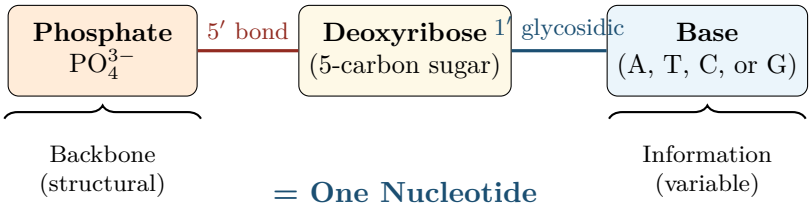


Figure 1.3: A nucleotide consists of three parts. The phosphate and sugar form the structural backbone; the base carries the genetic information.

Example 1.1 (Nucleotide naming)**.** The nucleotide containing adenine is called *deoxyadenosine 5′-monophosphate* (dAMP). Similarly: dCMP (cytosine), dGMP (guanine), dTMP (thymine). The "d" prefix stands for "deoxy," distinguishing DNA nucleotides from RNA nucleotides (which have AMP, CMP, GMP, UMP — note that RNA uses uracil instead of thymine).

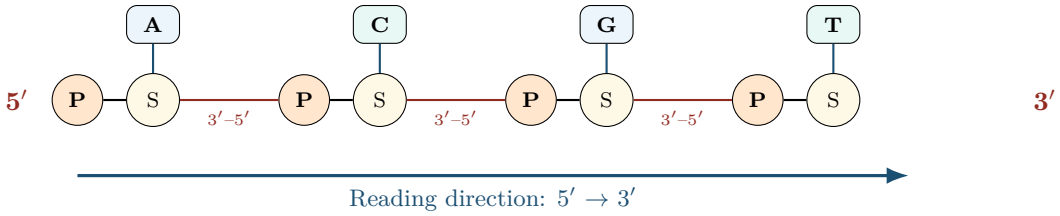
1.1.4 Step 4: The Single Strand — A Polynucleotide Chain

Nucleotides link together via **phosphodiester bonds**: the phosphate group of one nucleotide bonds to the 3′ carbon of the next nucleotide's sugar. This creates a long chain with a repeating sugar-phosphate backbone and a sequence of bases projecting from it.

The chain has *directionality*:

- The **5′ end** has a free phosphate group.
- The **3′ end** has a free hydroxyl group (−OH).

By convention, DNA sequences are always written from 5′ to 3′, just as English text is written left to right.



Sequence: 5′–ACGT–3′. Each P–S unit forms the backbone; the bases (A, C, G, T) project upward and carry the genetic information. The phosphodiester bonds (red) link successive nucleotides.

Figure 1.4: A single DNA strand (polynucleotide chain) showing the sugar-phosphate backbone and projecting bases. The strand has a defined 5′-to-3′ directionality.

1.1.5 Step 5: The Double Helix — Watson–Crick Base Pairing

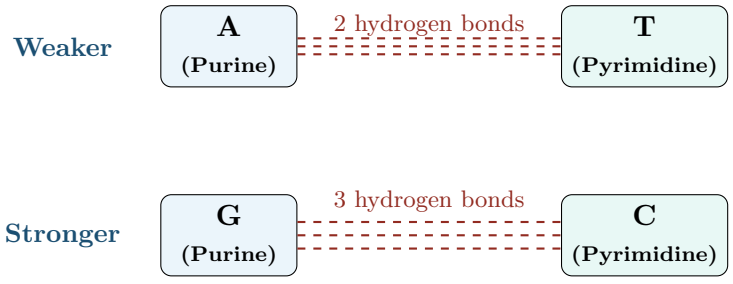
The discovery of Watson and Crick was that DNA exists as two antiparallel strands wound around each other in a right-handed helix, held together by specific hydrogen bonds between complementary bases [Watson and Crick, 1953]:

- **Adenine (A) pairs with Thymine (T)** via **2 hydrogen bonds**.
- **Guanine (G) pairs with Cytosine (C)** via **3 hydrogen bonds**.

This is called **Watson–Crick base pairing**, and it is the single most important fact in molecular biology.

The DNA Alphabet

The *DNA alphabet* is the finite set $\Sigma_{\text{DNA}} = \{A, C, G, T\}$. A *genomic sequence* of length n is a word $s = s_1 s_2 \cdots s_n \in \Sigma_{\text{DNA}}^*$, where Σ_{DNA}^* denotes the free monoid over Σ_{DNA} under concatenation.



Why this matters: G-C base pairs are thermodynamically more stable than A-T pairs because they form three hydrogen bonds instead of two. DNA regions rich in G-C content have higher melting temperatures. This is directly exploited in DNA computing when designing toeholds and displacement gates (Chapter 2).

Figure 1.5: Watson–Crick base pairing. Adenine pairs with thymine (2 H-bonds); guanine pairs with cytosine (3 H-bonds). Dashed lines represent hydrogen bonds.

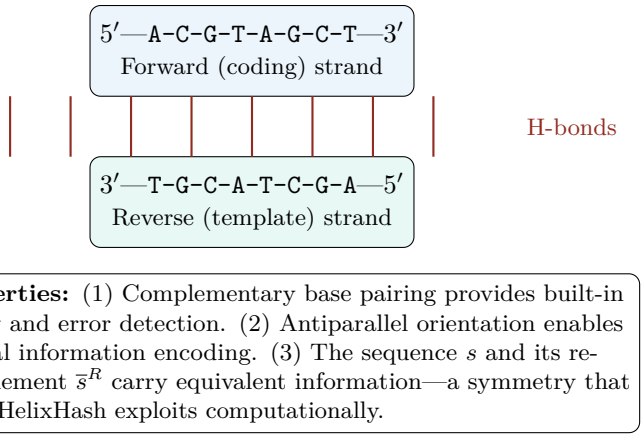


Figure 1.6: Schematic of the DNA double helix as a paired data structure. The two strands carry complementary information in antiparallel orientation.

From a computer science perspective, the double helix is a remarkable data structure. It provides:

1. **Redundancy through complementarity.** Each strand encodes the same information in complementary form. If one strand reads `ACGT`, the opposite strand necessarily reads `TGCA` (in the antiparallel direction). This is nature's error-detection mechanism—and the basis for DNA replication.

2. **Bidirectional readability.** The two strands run in opposite directions (5′ to 3′ and 3′ to 5′). Biological machinery reads the template strand in the 3′-to-5′ direction to synthesize RNA in the 5′-to-3′ direction. This antiparallel structure will later inspire HelixHash's bidirectional strand representation (Chapter 8).

3. **Structural access interfaces.** The double helix has a major groove and a minor groove. Proteins "read" DNA sequence without unwinding the helix by probing the pattern of hydrogen bond donors and acceptors exposed in these grooves [Alberts et al., 2022]. This is analogous to providing different read interfaces to the same underlying data.

Example 1.2 (Finding the complement)**.** Given the sequence 5′-`ATGCCA`-3′, find the complementary strand.

Solution. Apply the base-pairing rules (A↔T, C↔G) and reverse the direction:

1. Write the complement of each base: T–A–C–G–G–T.

2. This gives the complementary strand in the 3′-to-5′ direction.

3. Written in standard 5′-to-3′ notation: 5′-`TGGCAT`-3′.

The reverse complement of `ATGCCA` is `TGGCAT`.

1.1.6 DNA Structure: Summary of the Build-up

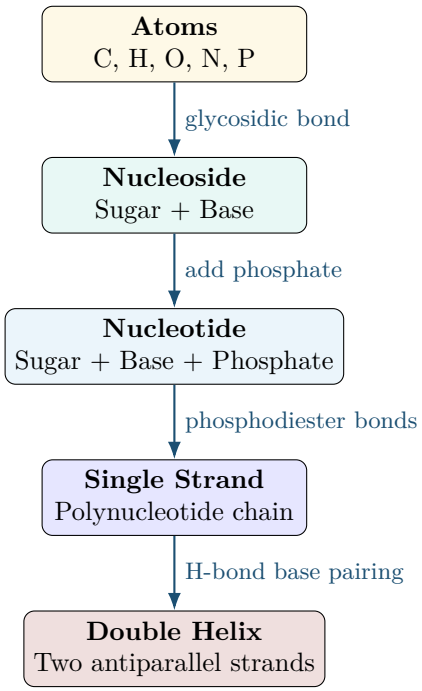


Figure 1.7: Building DNA step by step: from atoms to the double helix in five levels of organization.

1.2 DNA as Information Storage

1.2.1 Information Density: Nature's Storage Medium

The information density of DNA is extraordinary. Each nucleotide encodes $\log_2 4 = 2$ bits of information. A single gram of DNA can theoretically store approximately 2.15×10^{11} gigabytes—roughly 215 petabytes [Church et al., 2012]. For comparison:

Storage Medium	Density	Longevity
Hard disk drive	$\sim$1 TB/cm^3	$\sim$5 years
Flash memory (SSD)	$\sim$10 TB/cm^3	$\sim$10 years
Blu-ray disc	$\sim$0.03 TB/cm^3	$\sim$50 years
DNA (theoretical)	$\sim 10^9$ TB/cm^3	>10,000 years

The human genome contains approximately 3.2×10^9 base pairs, encoding roughly 6.4 gigabits ($\approx$ 800 megabytes) of raw sequence information. However, the *effective* information content is much lower due to extensive repetitive sequences, non-coding regions, and redundancy [Voss, 1992].

1.2.2 Binary Encoding of DNA

Since there are exactly four bases, each base maps naturally to a two-bit binary code. A common encoding is:

Base	Binary Code	Complement	Complement Code
A	00	T	11
C	01	G	10
G	10	C	01
T	11	A	00

Remark 1.2. Notice a useful property of this encoding: the complement of a base is obtained by flipping both bits (bitwise NOT). $\overline{00} = 11$ (A↔T), $\overline{01} = 10$ (C↔G). This makes complementation a single bitwise operation on hardware.

Example 1.3 (DNA to binary conversion)**.** Convert the DNA sequence `ATGC` to binary.

Solution.

A	T	G	C
00	11	10	01

So `ATGC` = 00 11 10 01 = `0x39` in hexadecimal (binary 00111001).

The complementary strand `GCAT` encodes as 10 01 00 11 = `0x93`. Note that the complement's binary representation is the bitwise NOT of the original: $\overline{\texttt{0x39}} = \texttt{0xC6...}$ but we must also reverse the order of the two-bit groups to get the antiparallel complement: $\overline{\texttt{0x39}}^R = \texttt{0x93}$.

Compression and Biological Information

The raw Shannon entropy of the human genome is approximately 1.8 bits per base pair—less than the theoretical maximum of 2 bits—reflecting statistical biases in base composition and local correlations such as dinucleotide frequencies [Mantegna et al., 1994]. This sub-maximal entropy is not a design flaw; it reflects the fact that genomes are not random strings but structured programs with regulatory grammar, repeated motifs, and evolutionary constraints.

1.2.3 The Four-Letter Alphabet: Why Four?

A natural question from an information-theoretic perspective: why does life use a four-letter alphabet rather than two (binary) or twenty (amino acids)?

The answer involves a tradeoff between *information density per symbol*, *chemical distinguishability*, and *error tolerance*. Two bases would require 50% longer sequences to encode the same information. More than four bases would increase error rates during replication because the polymerase must distinguish between more chemically similar substrates. The choice of four represents a near-optimal balance [Alberts et al., 2022].

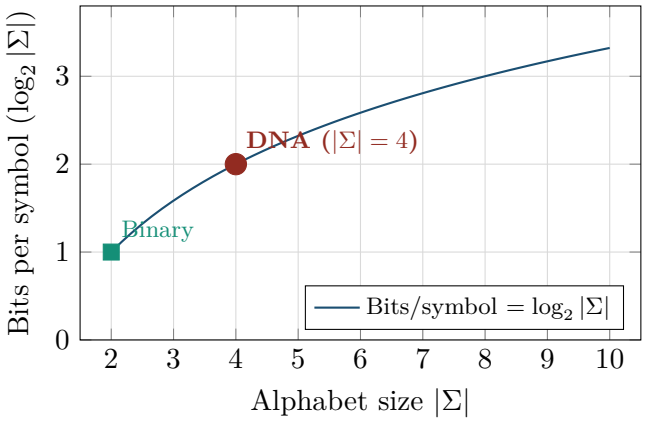


Figure 1.8: Information density as a function of alphabet size. DNA's four-letter alphabet provides 2 bits per symbol—a good balance between density and biochemical fidelity.

This same principle reappears in DOGMA, where we assign four computational roles to genomic bases: **A**ctivation, **T**ermination, **C**omposition, and **G**eneration. The four-type system is not merely aesthetic—it provides a compact basis for typed computational roles while maintaining formal compatibility with DNA's algebra.

1.2.4 DNA Data Storage

The extraordinary information density of DNA has inspired a growing field of DNA-based data storage. The pioneering work of Church et al. [2012] demonstrated the storage of an entire book (5.27 megabits) in synthetic DNA. Erlich and Zielinski [2017] improved the efficiency with their DNA Fountain architecture, achieving 1.57 bits per nucleotide—near the theoretical maximum of 1.98 bits (accounting for biochemical constraints). Organick et al. [2018] demonstrated random access to specific data items within a large DNA archive.

DNA Storage vs. Silicon Storage

DNA data storage is impractical for everyday computing: write speeds are slow (chemical synthesis), read speeds require sequencing, and costs remain high. But DNA's advantages—extreme density, durability (readable after thousands of years), and zero energy consumption during storage—make it compelling for archival applications. More importantly for this book, DNA storage research demonstrates that biological sequences *can* serve as computational substrates, lending credibility to DOGMA's use of genomic representations for AI.

1.3 The Central Dogma of Molecular Biology

In 1958, Francis Crick articulated the Central Dogma of molecular biology: information flows from DNA to RNA to protein [Crick, 1958]. He later refined this in 1970, clarifying that the Central Dogma is really a statement about *information transfer*: once sequential information has been translated into protein, it cannot flow back to nucleic acid [Crick, 1970].

The Central Dogma (Crick, 1970)

The Central Dogma states that the transfer of sequential information from nucleic acid to protein is *irreversible*. The permitted information transfers are:

- DNA → DNA (replication)
- DNA → RNA (transcription)
- RNA → Protein (translation)
- RNA → DNA (reverse transcription, discovered later)
- RNA → RNA (RNA replication, in some viruses)

The forbidden transfer is: Protein → DNA or Protein → RNA.

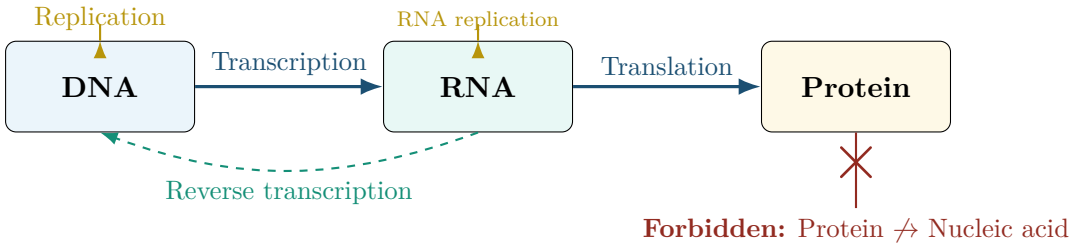


Figure 1.9: The Central Dogma of molecular biology. Solid blue arrows show the main information flow; dashed teal shows reverse transcription; gold loops show self-replication. Information never flows from protein back to nucleic acid.

1.3.1 Transcription: DNA to RNA, Step by Step

Transcription is the process by which a segment of DNA is copied into messenger RNA (mRNA) by the enzyme **RNA polymerase**. Here is how it works, step by step:

1. **Initiation.** RNA polymerase binds to a *promoter* region upstream of the gene. Transcription factors help position the enzyme and open ("melt") the double helix locally.
2. **Elongation.** RNA polymerase moves along the *template strand* (3′ to 5′ direction), reading each base and adding the complementary RNA nucleotide to the growing mRNA chain (5′ to 3′). The base-pairing rules are the same as DNA except that **uracil (U)** replaces thymine (T):

DNA template base	RNA base added
A	U
T	A
C	G
G	C

3. **Termination.** RNA polymerase reaches a *terminator* sequence and releases the completed mRNA transcript.

Example 1.4 (Transcription)**.** Given the DNA template strand 3′-`TACGGCAAT`-5′, what is the mRNA?

Solution. RNA polymerase reads 3′ to 5′ and synthesizes mRNA 5′ to 3′:

Template (3′→5′):	T	A	C	G	G	C	A	A	T
mRNA (5′→3′):	A	U	G	C	C	G	U	U	A

The mRNA sequence is 5′-`AUGCCGUUA`-3′.

Note: the mRNA has the same sequence as the *coding strand* (the non-template strand) of the DNA, but with U replacing T.

Transcription as Conditional Compilation

From a computational perspective, transcription is not mere copying. It is *conditional compilation*: the genome is the source code, transcription factors are the preprocessor directives, and the mRNA is the compiled intermediate representation—selected, spliced, and customized for the current cellular context.

Crucially, transcription is *regulated*: not all genes are transcribed in all cells at all times. The decision of which genes to transcribe, and at what rate, is controlled by an elaborate system of regulatory elements including promoters, enhancers, silencers, and transcription factors (Chapter 3).

1.3.2 The Genetic Code: A 64-to-21 Mapping

The genetic code maps 64 possible three-nucleotide sequences (*codons*) to 20 amino acids plus a stop signal. This mapping is:

- **Degenerate:** Multiple codons map to the same amino acid. For example, leucine is encoded by six different codons (UUA, UUG, CUU, CUC, CUA, CUG). This redundancy provides error tolerance—many point mutations at the third codon position are silent.
- **Nearly universal:** With minor exceptions, the same code is used by all known life on Earth, from bacteria to humans.
- **Optimized:** The code is structured so that chemically similar amino acids tend to have similar codons, minimizing the phenotypic impact of translation errors [Alberts et al., 2022].

The Standard Codon Table

The following table shows all 64 codons organized by first, second, and third position. Start codons are shown in **bold** and stop codons in red.

1st	Second Position U	C	A	G	3rd
U	Phe (F)	Ser (S)	Tyr (Y)	Cys (C)	U
	Phe (F)	Ser (S)	Tyr (Y)	Cys (C)	C
	Leu (L)	Ser (S)	**Stop**	**Stop**	A
	Leu (L)	Ser (S)	**Stop**	Trp (W)	G
C	Leu (L)	Pro (P)	His (H)	Arg (R)	U
	Leu (L)	Pro (P)	His (H)	Arg (R)	C
	Leu (L)	Pro (P)	Gln (Q)	Arg (R)	A
	Leu (L)	Pro (P)	Gln (Q)	Arg (R)	G
A	Ile (I)	Thr (T)	Asn (N)	Ser (S)	U
	Ile (I)	Thr (T)	Asn (N)	Ser (S)	C
	Ile (I)	Thr (T)	Lys (K)	Arg (R)	A
	Met (M)	Thr (T)	Lys (K)	Arg (R)	G
G	Val (V)	Ala (A)	Asp (D)	Gly (G)	U
	Val (V)	Ala (A)	Asp (D)	Gly (G)	C
	Val (V)	Ala (A)	Glu (E)	Gly (G)	A
	Val (V)	Ala (A)	Glu (E)	Gly (G)	G

- **Start codon:** `AUG` codes for methionine (Met) and signals the ribosome to begin translation. Every protein begins with methionine (though it may be removed later).
- **Stop codons:** `UAA` ("ochre"), `UAG` ("amber"), and `UGA` ("opal") signal the ribosome to terminate translation.

Codons in DOGMA

DOGMA's CodonForge compiler (Chapter 11) uses a direct analogy: 64 three-letter codons are mapped to computational opcodes. Just as the biological genetic code maps triplets to amino acids, CodonForge maps triplets to operations such as `LOAD_SEQUENCE`, `MATCH_KEY`, `EMIT_JSON`, and `REVERSE_COMPLEMENT`. The degeneracy of the biological code inspires opcode aliasing, where multiple codon patterns can invoke the same operation with different optimization hints.

1.3.3 Translation: The Ribosomal Decoder

Translation occurs on ribosomes, molecular machines that read mRNA codons and assemble the corresponding amino acid chain. Transfer RNA (tRNA) molecules serve as adapters, each carrying a specific amino acid and bearing an anticodon that base-pairs with the mRNA codon.

Translation Step by Step

1. **Initiation.** The ribosome assembles on the mRNA at the start codon `AUG`. An initiator tRNA carrying methionine binds to the P-site.
2. **Elongation.** Repeated cycle:
   (i) A tRNA with the matching anticodon enters the **A-site** (aminoacyl site).
   (ii) A peptide bond forms between the amino acid in the A-site and the growing chain in the **P-site** (peptidyl site).

(iii) The ribosome translocates one codon forward. The empty tRNA exits through the **E-site** (exit site).

3. **Termination.** When a stop codon (`UAA`, `UAG`, or `UGA`) enters the A-site, a release factor binds instead of a tRNA, and the completed polypeptide is released.

Example 1.5 (Translating a short mRNA)**.** Translate the mRNA sequence 5′-`AUGCCGUUAUAA`-3′.

Solution. Divide into codons (groups of three, starting from AUG):

`AUG`	`CCG`	`UUA`	`UAA`
Met (start)	Pro	Leu	Stop

The resulting peptide is: **Met–Pro–Leu** (a tripeptide). Translation stops at UAA.

The ribosome is a *programmable decoder*: it takes a linear input (mRNA), processes it sequentially (codon by codon), and produces a structured output (a folded protein). The analogy to a CPU executing microcode is surprisingly precise:

CPU Component	**Ribosomal Analogue**	**Function**
Instruction register	A-site (aminoacyl)	Holds current codon/instruction
Accumulator	P-site (peptidyl)	Holds growing polypeptide/result
Output buffer	E-site (exit)	Releases used tRNA/completed result
Microcode	Genetic code table	Maps instructions to operations
Program counter	mRNA 5′→3′ scan	Sequential instruction pointer

1.3.4 Post-Translational Modification: Runtime Polymorphism

After translation, proteins are frequently modified by chemical additions (phosphorylation, glycosylation, ubiquitination) that alter their function, localization, or lifetime. This is biology's version of *runtime polymorphism*: the same gene product can exhibit different behaviors depending on post-translational context.

In DOGMA terms, this corresponds to the separation between the Express stage (which produces a structured phenotype) and the Realize stage (which produces modality-specific output). The same phenotype can be realized differently depending on the output context—just as the same protein can function differently depending on its post-translational state.

1.4 DNA as Information: A Quantitative Analysis

1.4.1 Shannon Entropy of Genomic Sequences

Claude Shannon's information theory [Shannon, 1948] provides the natural framework for quantifying the information content of DNA sequences. For a sequence drawn from alphabet Σ with symbol probabilities $\{p_i\}$, the entropy per symbol is:

$$H = -\sum_{i \in \Sigma} p_i \log_2 p_i \tag{1.1}$$

For a uniform distribution over $\{A, C, G, T\}$, the maximum entropy is $H_{\max} = \log_2 4 = 2$ bits per nucleotide. Real genomes deviate from this maximum in characteristic ways:

- **GC content bias:** Many organisms show non-uniform base composition. The human genome has ≈41% GC content, reducing single-symbol entropy to ≈1.99 bits.
- **Dinucleotide correlations:** Adjacent nucleotides are not independent. The CpG dinucleotide is significantly underrepresented in vertebrate genomes (due to methylation-induced deamination), introducing local correlations that further reduce conditional entropy [Voss, 1992].
- **Long-range correlations:** DNA sequences exhibit long-range power-law correlations that extend over thousands of base pairs, particularly in non-coding regions [Mantegna et al., 1994]. This structure is absent from protein-coding regions, suggesting that non-coding DNA has a statistical signature more similar to natural language than to random sequences.

Example 1.6 (Computing Shannon entropy)**.** Suppose a short DNA sequence has the following base frequencies: $p_A = 0.30$, $p_T = 0.30$, $p_C = 0.20$, $p_G = 0.20$.

Solution.

$$\begin{aligned} H &= -\big(0.30\log_2 0.30 + 0.30\log_2 0.30 + 0.20\log_2 0.20 + 0.20\log_2 0.20\big) \\ &= -\big(2\times 0.30\times(-1.737) + 2\times 0.20\times(-2.322)\big) \\ &= -\big(-1.042 - 0.929\big) \\ &= 1.971 \text{ bits per base.} \end{aligned}$$

This is slightly below the maximum of 2.0 bits, reflecting the non-uniform distribution.

Conditional Entropy of DNA

The k-th order conditional entropy of a genomic sequence is:

$$H_k = -\sum_{w\in\Sigma^k} p(w)\sum_{s\in\Sigma} p(s|w)\log_2 p(s|w), \tag{1.2}$$

where $p(s|w)$ is the probability of observing symbol s given the preceding k-mer w. As k increases, H_k generally decreases, reflecting increasing predictability.

1.4.2 Redundancy in the Genetic Code

The genetic code maps 64 codons to only 21 outputs (20 amino acids + stop). This means the code has substantial *redundancy* or *degeneracy*. We can quantify this information-theoretically.

The maximum information a codon could carry is $\log_2 64 = 6$ bits. The information needed to specify one of 21 outputs is $\log_2 21 \approx 4.39$ bits. The difference, $6 - 4.39 = 1.61$ bits per codon, represents the redundancy.

This redundancy is not wasted—it serves as *error correction*. Many single-nucleotide mutations (especially at the third "wobble" position of a codon) are *synonymous*: they change the codon but not the amino acid. This provides a buffer against the deleterious effects of point mutations.

Example 1.7 (Wobble position degeneracy)**.** Consider the amino acid alanine (Ala). It is encoded by four codons: GCU, GCC, GCA, GCG. Notice that the first two positions (GC) are fixed, while the third position can be any of the four bases. A mutation at the third position of any alanine codon produces another alanine codon—a *silent mutation*.

This is analogous to an error-correcting code in communication theory: the "message" (amino acid) is protected by "redundant bits" (the wobble position).

1.4.3 Comparing DNA, Natural Language, and Code

A revealing exercise is to compare the information-theoretic properties of genomic sequences with those of natural language and programming languages:

Property	DNA	English	Python Code
Alphabet size	4	∼27	∼95
H_0 (bits/symbol)	2.0	∼4.76	∼6.57
H_1 (empirical)	∼1.95	∼4.03	∼5.8
Long-range correlations	Yes (power-law)	Yes (power-law)	Yes (block structure)
Redundancy	∼3% (single-symbol)	∼50%+	High (syntactic)

The key observation is that DNA and natural language share a statistical deep structure—both exhibit long-range correlations, hierarchical organization, and sub-maximal entropy. This is not coincidence. Both are *evolved communication systems*: DNA communicates developmental instructions across generations; language communicates meaning across minds. This parallel is one of the motivations for applying language model architectures to genomic data (Chapter 6).

1.5 DNA Replication: Copying the Code

Before a cell divides, it must duplicate its entire genome so that each daughter cell receives a complete copy. This process—**DNA replication**—is one of the most remarkable molecular operations in biology.

1.5.1 Overview: Semi-Conservative Replication

DNA replication is *semi-conservative*: each new double helix consists of one original ("parent") strand and one newly synthesized ("daughter") strand. This was demonstrated by the Meselson–Stahl experiment (1958).

1.5.2 Step-by-Step Mechanism

1. **Helicase unwinds the double helix.** The enzyme helicase breaks the hydrogen bonds between base pairs, creating a Y-shaped *replication fork* with two single-stranded template regions.

2. **Single-strand binding proteins (SSBPs) stabilize.** SSBPs coat the exposed single strands to prevent them from re-annealing or forming secondary structures.

3. **Primase synthesizes RNA primers.** DNA polymerase cannot start a new chain from scratch—it can only *extend* an existing strand. Primase lays down short RNA primers (about 10 nucleotides) to provide a starting point.

4. **DNA Polymerase III extends the primers.** The main replication enzyme reads the template strand (3′ to 5′) and synthesizes the new strand (5′ to 3′) by adding complementary

nucleotides. It also *proofreads* each addition, removing mismatched bases (error rate: ∼1 in 10^7).

5. **Leading vs. lagging strand.** Because DNA polymerase can only synthesize 5′ to 3′:
   - The **leading strand** is synthesized continuously toward the fork.
   - The **lagging strand** is synthesized in short fragments (*Okazaki fragments*, ∼1000–2000 nt in prokaryotes) away from the fork.
6. **DNA Polymerase I replaces RNA primers with DNA.**
7. **DNA Ligase seals the gaps** between Okazaki fragments, creating a continuous strand.

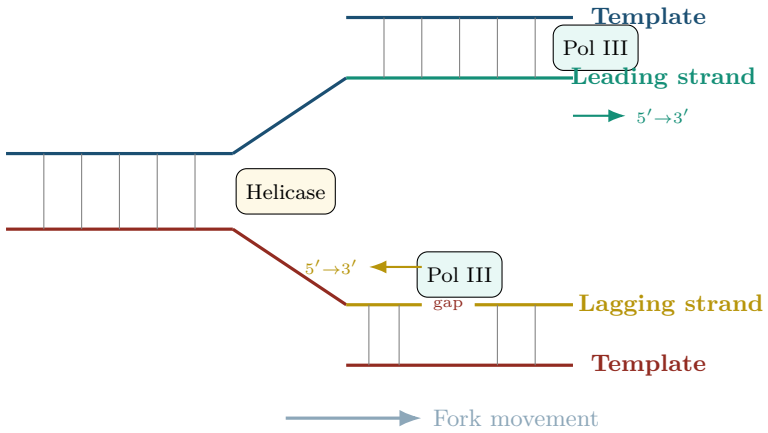


Figure 1.10: The DNA replication fork. Helicase unwinds the double helix. The leading strand (teal) is synthesized continuously; the lagging strand (gold) is synthesized as Okazaki fragments.

1.5.3 Error Correction: Proofreading and Repair

DNA replication achieves extraordinary fidelity through multiple layers of error correction:

Mechanism	Error Rate After	Improvement
Base selection (Watson–Crick)	1 in 10^4–10^5	—
Polymerase proofreading (3′→5′ exonuclease)	1 in 10^7	∼100×
Post-replication mismatch repair	1 in 10^9–10^{10}	∼100×

The net result is approximately one error per 10^9–10^{10} base pairs copied. For the human genome (3.2×10^9 bp), this means roughly 0.3–3 mutations per cell division.

Replication as a Noisy Channel

From Shannon's perspective, DNA replication is a noisy communication channel with a remarkably low bit-error rate ($\sim 10^{-10}$). The biological proofreading and repair systems function as error-correcting codes. This is vastly better than any human-engineered digital communication system—modern fiber-optic links typically achieve bit-error rates of 10^{-12} to 10^{-15}, but they use far more energy per bit corrected.

1.6 DNA as a Storage and Computing Medium

1.6.1 DNA Strand Displacement as a Computational Primitive

Beyond storage, DNA can *compute*. The key primitive is *toehold-mediated strand displacement* [Zhang and Seelig, 2011]: a short single-stranded region (the toehold) allows an incoming strand to initiate branch migration, ultimately displacing the incumbent strand from a double-stranded complex.

This simple reaction can implement:

- **Signal transduction:** An input strand triggers displacement, releasing an output strand.
- **Logic gates:** AND, OR, and NOT gates can be built from displacement cascades [Seelig et al., 2006].
- **Amplification:** Catalytic displacement circuits can amplify signals exponentially.
- **Neural networks:** Cherry and Qian [2018] demonstrated a DNA-based winner-take-all neural network capable of classifying handwritten digits.

From Wet-Lab to Silicon

DOGMA does not propose running AI on actual DNA molecules. It asks whether selected computational abstractions from DNA computing can be useful on silicon. Strand displacement, toehold binding, and thermodynamic affinity appear in historical DOGMA design studies, but the current measured baseline in Chapter 14 uses the canonical non-attention contract. No efficiency advantage from the molecular analogy has yet been established.

1.6.2 Chemical Reaction Networks and Turing Completeness

Soloveichik et al. [2010] proved that chemical reaction networks (CRNs)—abstract models of molecular interactions—are *Turing complete*. Any computation that can be performed by a Turing machine can, in principle, be performed by a suitable set of chemical reactions. This result establishes a deep theoretical connection between chemistry and computation.

Definition 1.1 (Chemical Reaction Network)**.** A *chemical reaction network* (CRN) is a tuple $(\mathcal{S}, \mathcal{R})$ where $\mathcal{S}$ is a finite set of species and $\mathcal{R}$ is a finite set of reactions. Each reaction $r \in \mathcal{R}$ has the form:

$$\alpha_1 S_1 + \alpha_2 S_2 + \cdots \xrightarrow{k_r} \beta_1 S_1 + \beta_2 S_2 + \cdots \tag{1.3}$$

where $\alpha_i, \beta_i \in \mathbb{N}$ are stoichiometric coefficients and $k_r > 0$ is a rate constant. Under mass-action kinetics, the dynamics are governed by:

$$\frac{d[S_i]}{dt} = \sum_{r \in \mathcal{R}} (\beta_{i,r} - \alpha_{i,r}) \cdot k_r \prod_j [S_j]^{\alpha_{j,r}} \tag{1.4}$$

The Turing completeness of CRNs means that molecular computation is not a curiosity—it is a fundamentally powerful computational paradigm. DOGMA's architecture draws on this power by implementing CRN-inspired computation in its neural substrate (see Chapter 14, Section on Chemical Reaction Network modules).

1.7 From Molecules to Architecture: The Abstraction Bridge

This section crystallizes the conceptual bridge between biological information processing and DOGMA's computational architecture.

1.7.1 The Genome as Source Code

Consider the following analogy, made precise throughout this book:

Biology	**Software Engineering**	**DOGMA**
Genome	Source code repository	Persistent genomic state Γ_t
Gene regulation	Preprocessor / build system	Regulation engine A_t
Transcription	Compilation to IR	Transcription T_t
mRNA	Intermediate representation	Transcript
Translation	Code generation	Expression Φ_t
Protein function	Runtime behavior	Realization y_t
Mutation	Source code edit	Synthesis / mutation
Natural selection	Test suite + deployment	Fitness evaluation
Epigenetics	Configuration / environment vars	Tera regime state

This analogy is not perfect—no analogy is. But it captures a deep structural parallel: *in both biology and DOGMA, the system's persistent state (genome/source code) is not the same as its transient output (protein/text). The path from state to output passes through multiple regulated stages of selection, transformation, and contextualization.*

1.7.2 Why Previous Biological Metaphors in CS Were Shallow

Computer science has a long history of borrowing biological terminology: "genetic algorithms," "neural networks," "evolutionary programming," "artificial immune systems." But most of these borrowings are metaphorical rather than architectural. Genetic algorithms use the *vocabulary* of genetics (crossover, mutation, fitness) without implementing the *structure* of genomic information processing (regulation, transcription, alternative splicing, epigenetics).

DOGMA attempts to go deeper. It does not merely name its components after biological entities. It *instantiates* the computational logic of the Central Dogma as a formal pipeline with specific mathematical operations at each stage. The claim is not that biology is a good metaphor for AI. The claim is that biology has already solved the problem of organizing complex information processing—and that its solution can be formalized, generalized, and implemented.

1.8 Worked Examples

Example 1.8 (Complete workflow: DNA to protein)**.** Given the coding strand of a gene: 5′-ATGAAACCCGGATAA-3′.

Step 1: Find the template strand. Write the complement in the antiparallel direction:

Template strand: 3′-TACTTGGGCCTATT-5′.

Step 2: Transcribe to mRNA. The mRNA has the same sequence as the coding strand, but with U replacing T:

mRNA: 5′-AUGAAACCCGGAUAA-3′.

Step 3: Divide into codons.

AUG AAA CCC GGA UAA

Step 4: Translate using the codon table.

AUG	AAA	CCC	GGA	UAA
Met	Lys	Pro	Gly	Stop

Step 5: Result. The protein is: **Met–Lys–Pro–Gly** (a tetrapeptide).

Example 1.9 (Information content of a genome)**.** The bacterium *Mycoplasma genitalium* has one of the smallest known genomes: 580,076 base pairs.

(a) **Raw information content:** $580{,}076 \times 2 = 1{,}160{,}152$ bits ≈ 145 kilobytes. This is smaller than many JPEG images.

(b) **Number of proteins encoded:** *M. genitalium* has 482 protein-coding genes. On average, each gene is about $580{,}076/482 \approx 1{,}203$ bp (including non-coding regions between genes).

(c) **Comparison:** The Linux kernel source code (v5.0) is about 25 million lines, or roughly 800 megabytes. A minimal self-replicating organism fits in 145 kilobytes—a stunning compression ratio for the "program" that runs a living cell.

1.9 Exercises

1.1. [**Fundamentals**] Given the DNA sequence ATGCGATCAATG:

(a) Write the complementary strand.

(b) Write the reverse complement.

(c) Compute the GC content as a percentage.

(d) If this sequence were transcribed, what would the mRNA sequence be? (Replace T with U.)

1.2. [**Base Pairing**] Explain why adenine cannot pair with guanine in standard Watson–Crick base pairing. (Hint: consider the number of rings and the geometry of hydrogen bonds.)

1.3. **[Binary Encoding]** Using the encoding A=00, C=01, G=10, T=11:

(a) Encode the sequence `GATTACA` in binary.

(b) What is the hexadecimal representation?

(c) Verify that the bitwise NOT of your answer gives the complement (`CTAATGT`).

1.4. **[Information Theory]**

(a) Compute the Shannon entropy per base for a genome with base frequencies $p_A = 0.29$, $p_T = 0.29$, $p_C = 0.21$, $p_G = 0.21$.

(b) How does this compare to the maximum entropy of 2 bits? What does the deficit represent biologically?

(c) If a DNA storage system uses only {A, C} (avoiding G-T to reduce errors), what is the information density in bits per nucleotide?

1.5. **[Translation]** Translate the following mRNA sequences into amino acid chains using the codon table:

(a) 5′-`AUGUUUAAAUAG`-3′

(b) 5′-`AUGGGCUACGCUUGA`-3′

1.6. **[Replication]** Draw a diagram of a replication fork for the sequence 5′-`AACCTGGATC`-3′. Label the leading strand, lagging strand, Okazaki fragments, and the direction of fork movement.

1.7. **[Coding]** Write a Python function that takes a DNA sequence string and returns a dictionary with:

(a) The complement, reverse complement, GC content, and Shannon entropy.

(b) The k-mer frequency spectrum for a given k.

(c) Test your function on the first 1000 bases of any publicly available genome.

1.8. **[Conceptual]** The Central Dogma states that information cannot flow from protein back to nucleic acid. What would it mean computationally if this constraint were violated? Discuss the implications for DOGMA's architecture.

1.9. **[Redundancy]** Calculate the number of distinct DNA sequences that encode the peptide Met–Ala–Gly–Stop. (Hint: count the codons for each amino acid and multiply.)

1.10. **[Research]** Read Cherry and Qian [2018]. In their DNA-based neural network, what role does strand displacement play that is analogous to synaptic weighting in silicon neural networks? How does their winner-take-all mechanism compare to the softmax function?

1.11. **[Advanced]** Prove that the set of DNA sequences Σ^*_{DNA} under concatenation forms a free monoid. Why is the "free" property important for modeling biological sequence operations like restriction enzyme cutting and ligation?

1.10 Key Takeaways

Key Takeaways

- DNA is built from five atoms (C, H, O, N, P) arranged into nucleotides, which polymerize into strands, which pair into the double helix.
- Watson–Crick base pairing (A-T with 2 H-bonds, G-C with 3 H-bonds) provides complementarity, redundancy, and error detection.
- DNA is a four-letter alphabet with extraordinary information density ($\sim$2 bits/base, $\sim 10^9$ TB/cm^3). The binary encoding A=00, C=01, G=10, T=11 maps complementation to bitwise NOT.
- The Central Dogma (DNA $\rightarrow$ RNA $\rightarrow$ Protein) is not merely a chemical pipeline—it is a staged computational architecture separating persistent state from transient output.
- The genetic code maps 64 codons to 20 amino acids + stop, with degeneracy that provides error tolerance analogous to error-correcting codes.
- DNA replication achieves extraordinary fidelity ($\sim 10^{-10}$ error rate) through proofreading and mismatch repair.
- Genomic sequences share deep statistical properties with natural language: long-range correlations, hierarchical structure, and sub-maximal entropy.
- DNA can serve as both a storage medium and a computing substrate, with strand displacement as the key computational primitive and CRNs providing Turing-complete molecular computation.
- DOGMA translates these biological principles into a formal AI architecture, moving beyond shallow metaphor to deep structural correspondence.

Suggested Reading

- Alberts et al. [2022], Chapters 1, 4–7: Essential molecular biology background.
- Crick [1970]: The original refined statement of the Central Dogma.
- Shannon [1948]: Foundation of information theory.
- Church et al. [2012]: DNA data storage breakthrough.
- Zhang and Seelig [2011]: Comprehensive review of DNA strand displacement.

Chapter 2

A History of DNA Computing — From Adleman to Molecular Programming

The molecule does the work; the scientist sets up the problem.

— Leonard Adleman, 1994

On a November day in 1994, Leonard Adleman walked into his laboratory at the University of Southern California and solved an instance of the Hamiltonian path problem—not on a computer, but in a test tube. Using carefully designed DNA oligonucleotides, enzymatic reactions, and gel electrophoresis, he demonstrated that molecules could perform meaningful computation [Adleman, 1994]. The paper, published in *Science*, launched an entirely new field.

Three decades later, DNA computing has evolved from a provocative demonstration into a mature discipline with its own programming languages, compiler toolchains, and a growing library of implemented algorithms. This chapter traces that evolution in detail, building the intuition and technical foundations that underpin DOGMA's architecture.

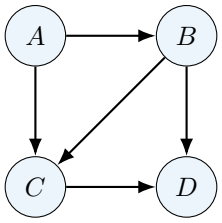
For the DOGMA reader, this chapter serves a specific purpose: it establishes that the computational primitives of DNA—pairing, displacement, catalysis, self-assembly—are not merely biological curiosities. They are *computational operations* that can be formalized, abstracted, and reimplemented on silicon. DOGMA is the architecture that performs that reimplementation.

2.1 Adleman's Breakthrough (1994)

2.1.1 The Hamiltonian Path Problem

The Hamiltonian path problem asks: given a directed graph $G = (V, E)$ with designated start and end vertices, does there exist a path that visits every vertex exactly once? The problem is NP-complete, meaning no polynomial-time algorithm is known for the general case.

Example 2.1 (A Simple Hamiltonian Path)**.** Consider a directed graph with four vertices $\{A, B, C, D\}$ and edges $A \to B$, $A \to C$, $B \to C$, $B \to D$, $C \to D$. Starting from A and ending at D, is there a path visiting every vertex exactly once?



The answer is yes: $A \to B \to C \to D$ visits all four vertices exactly once. But how would we find this using DNA?

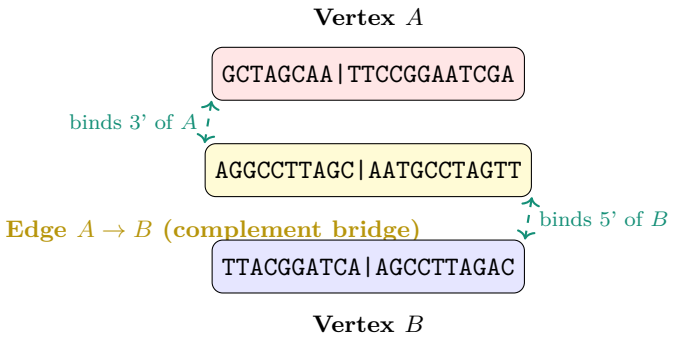
2.1.2 Encoding Graphs in DNA

Adleman's key insight was to encode the graph structure directly into DNA sequences, exploiting Watson-Crick complementarity to "explore" all possible paths simultaneously.

Step 1: Assign vertex sequences. Each vertex receives a random 20-nucleotide DNA sequence. For our four-vertex example:

Vertex	DNA Sequence (20-mer)
A	5'-GCTAGCAATTCCGGAATCGA-3'
B	5'-TTACGGATCAAGCCTTAGAC-3'
C	5'-CGGAATCCGTTAAGGCCATA-3'
D	5'-AAGCTTGATCCGGAATTACG-3'

Step 2: Encode edges as bridging strands. Each edge (v_i, v_j) is encoded as a 20-nucleotide oligo whose *first half* (10 nt) is complementary to the *3′ end* of v_i and whose *second half* (10 nt) is complementary to the *5′ end* of v_j. This creates a molecular "bridge" that physically links vertex v_i to vertex v_j:



Step 3: Mix and hybridize. When all vertex and edge oligonucleotides are mixed in solution, complementary sequences spontaneously hybridize. Each edge oligo acts as a "splint" that connects two vertex strands, creating double-stranded bridges. In a single test tube, *all possible paths through the graph assemble simultaneously.*

Step 4: Filter. Adleman then used three biochemical techniques to extract the correct answer:

1. **PCR amplification:** Using primers specific to vertex A (start) and vertex D (end), amplify only paths beginning at A and ending at D.

2. **Gel electrophoresis:** Select paths of the correct length. A path visiting all 4 vertices contains 4 vertex sequences = 80 nucleotides. Paths of the wrong length (too short = missing vertices; too long = cycles) are discarded.

3. **Affinity purification:** For each vertex, use a complementary strand bound to a magnetic bead to "fish out" only those paths containing that vertex. Repeat for all vertices. Only paths containing all vertices survive.

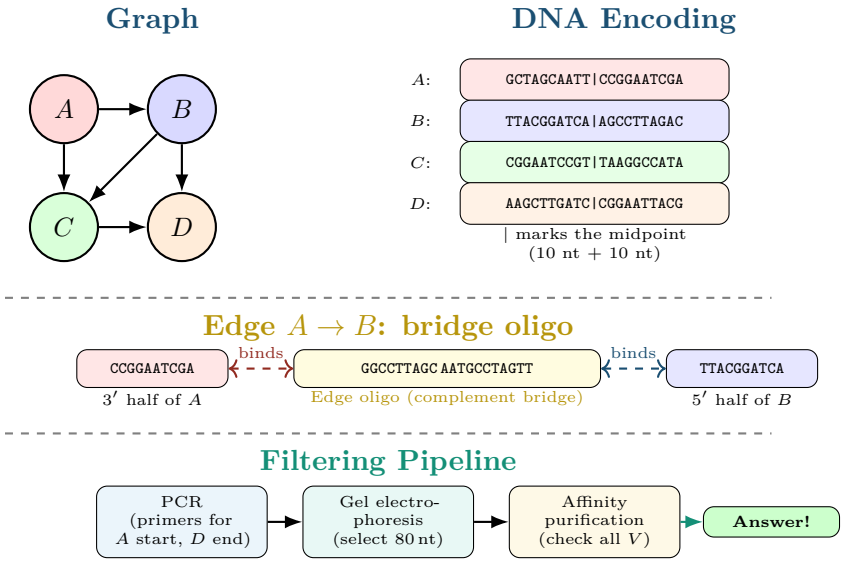


Figure 2.1: Adleman's experiment: a directed graph is encoded in DNA by assigning 20-mer sequences to vertices and complement-bridge oligos to edges. After self-assembly in solution, three filtering steps extract the Hamiltonian path [Adleman, 1994].

Massive Parallelism

The crucial advantage of DNA computation is *massive parallelism*. A test tube containing 10^{18} molecules simultaneously explores 10^{18} candidate solutions. This is a SIMD (Single Instruction, Multiple Data) architecture taken to an extreme that silicon cannot match in raw parallelism. Where a classical computer tries one path at a time, DNA tries them all at once.

2.1.3 Why It Worked—And Why It Didn't Scale

Adleman's experiment worked because the search space was small ($7! = 5040$ possible paths for his actual 7-vertex graph) and the molecular encoding was reliable at that scale. However, the approach faces fundamental scaling challenges:

- **Exponential material requirements.** For n vertices, the number of candidate paths grows exponentially. A 70-vertex graph would require more DNA mass than exists on Earth.

- **Error accumulation.** Hybridization errors, incomplete reactions, and PCR artifacts compound with problem size.

- **Readout bottleneck.** Extracting the answer from the molecular mixture requires multiple biochemical steps, each with finite yield.

Despite these limitations, Adleman's work established a fundamental principle: *molecules can compute*. The question became not whether molecular computation is possible, but how to make it reliable, scalable, and programmable.

2.2 DNA Logic Gates and Boolean Computation

Perhaps the most important question for understanding DOGMA is: *How can DNA implement the basic building blocks of all digital computation—logic gates?* In this section, we build up DNA logic gates from first principles, with complete truth tables and worked examples.

2.2.1 Binary Encoding in DNA

Before building logic gates, we need to represent binary values (0 and 1) in molecular terms. The convention used in DNA computing is *concentration-based encoding*:

Definition 2.1 (Molecular Binary Encoding)**.** A binary signal is encoded as the concentration of a specific DNA species S:

$$\text{Value} = \begin{cases} \mathbf{0} & \text{if } [S] < \theta \quad \text{(concentration below threshold)} \\ \mathbf{1} & \text{if } [S] \geq \theta \quad \text{(concentration at or above threshold)} \end{cases} \tag{2.1}$$

where $[S]$ denotes the molar concentration of species S and θ is a detection threshold (typically $\sim$100 nM).

This is analogous to voltage-based encoding in silicon: in CMOS logic, "0" corresponds to low voltage (< 0.8V) and "1" corresponds to high voltage (> 2.0V).

2.2.2 The AND Gate

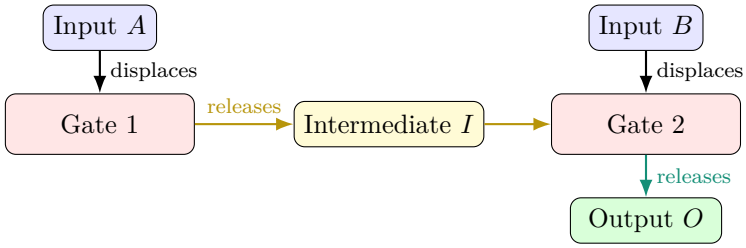
An AND gate produces output 1 only when *both* inputs are 1.

Truth table:

Input A	**Input** B	**Output** $A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	**1**

DNA implementation: The AND gate is implemented using *two sequential strand displacement reactions*. The output strand is released only after both input strands have participated:

1. **Gate complex 1:** Contains a toehold for input strand A. When A binds, it releases an intermediate strand I.
2. **Gate complex 2:** Contains a toehold for the intermediate strand I, but can only release the output strand O when input strand B is *also* present.



AND gate: Output O is only released when both A and B are present. Both displacement reactions must complete.

Figure 2.2: DNA AND gate using two sequential strand displacement reactions.

Step-by-step walkthrough:

1. **Both inputs absent ($A = 0, B = 0$):** No strand displacement occurs at Gate 1. Intermediate I is never released. Gate 2 remains closed. Output = 0. ✓
2. **Only A present ($A = 1, B = 0$):** Input A displaces Gate 1, releasing intermediate I. But I alone cannot open Gate 2 (which also requires B). Output = 0. ✓
3. **Only B present ($A = 0, B = 1$):** Gate 1 never fires (no A). Even though B is available, there is no intermediate I to co-trigger Gate 2. Output = 0. ✓
4. **Both present ($A = 1, B = 1$):** Input A displaces Gate 1, releasing I. Then I and B together open Gate 2, releasing output O. Output = 1. ✓

Concrete DNA sequences for an AND gate. To make this tangible, here are real DNA sequences that implement the AND gate above. Each strand has a specific nucleotide sequence—not abstract labels—so that Watson-Crick complementarity governs the computation [Daley and Kari, 2002, Seelig et al., 2006]:

Strand	Sequence ($5' \to 3'$)	Role
Input A	`CTAGGC ATTCGAATGCCTA`	6-nt toehold + 14-nt displacement
Input B	`GCCTAA TGGCAATTCCGGA`	6-nt toehold + 14-nt displacement
Gate 1 top	`TAGGCATTCGAATGCCTA`	Binds A; protects intermediate I
Intermediate I	`ATGCCTAAGGCCTTAGGA`	Released by Gate 1; triggers Gate 2
Gate 2 top	`TGGCAATTCCGGA TAGGA`	Binds B *and* I; protects output
Output O	`GCCTTAAGGCCTAATCCGG`	Released only when both gates fire

The 6-nucleotide toehold on each input (shown with a small space) initiates binding. Once the toehold domain hybridises, branch migration displaces the next strand along the gate complex. Gate 1 fires only when input A is present (its toehold `CTAGGC` is complementary to Gate 1's overhang `GATCCG`); Gate 2 requires *both* the intermediate strand released by Gate 1 *and* input B. No single strand alone can release the output. This is the same principle used in the abstract diagram of Figure 2.2, but now written in the language of real chemistry.

AND Gate in DOGMA

DOGMA's Strand Displacement path (Chapter 14) implements a differentiable AND gate using learned toehold scores. Two input features must both have high activation (analogous to high concentration) to produce a non-zero output through a multiplicative gating mechanism: $\text{out} = \sigma(W_A x_A) \odot \sigma(W_B x_B)$, where $\odot$ is elementwise multiplication.

2.2.3 The OR Gate

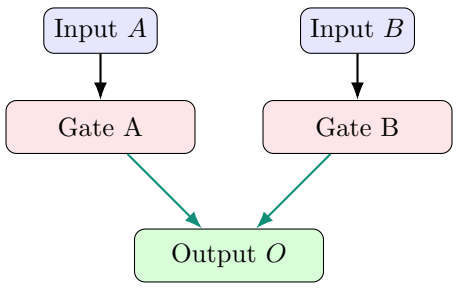
An OR gate produces output 1 when *at least one* input is 1.

Truth table:

Input A	**Input** B	**Output** $A \vee B$
0	0	0
0	1	**1**
1	0	**1**
1	1	**1**

DNA implementation: The OR gate uses *two parallel gates*, each producing the same output strand:

- **Gate A:** Has a toehold for input A. When A is present, it releases output strand O.
- **Gate B:** Has a toehold for input B. When B is present, it also releases output strand O.
- Since both gates produce the *same* output species O, the presence of *either* input (or both) results in high output concentration.



OR gate: Either Gate A or Gate B (or both) can produce the same output O.

Figure 2.3: DNA OR gate using two parallel strand displacement gates producing the same output.

Concrete DNA sequences for an OR gate. In the OR gate, both Gate A and Gate B release the *same* output species. Here is a concrete realisation:

Strand	Sequence ($5' \to 3'$)	Role
Input A	`TAGCCA GCCAATTGCCTAG`	Toehold binds Gate A overhang
Input B	`ACGTCG TTAACCGGAATCG`	Toehold binds Gate B overhang
Gate A top	`ATCGGT GCCAATTGCCTAG`	Protects shared output O_1
Gate B top	`TGCAGC TTAACCGGAATCG`	Protects shared output O_2
Output O	`CGGTTAACCGGATTAGCC`	*Same* strand released by either gate

Because O_1 and O_2 are identical sequences, the presence of *either* input produces high $[O]$. This is the molecular implementation of Boolean OR: the output reporter fluoresces whenever at least one input strand is at high concentration.

2.2.4 The NOT Gate (Inverter)

A NOT gate inverts its input: input 1 produces output 0, and input 0 produces output 1.

Truth table:

Input A	Output $\neg A$
0	**1**
1	0

DNA implementation: The NOT gate is more challenging because it requires producing a signal in the *absence* of an input. The solution uses a *threshold mechanism*:

1. A "fuel" strand continuously produces output O at a baseline rate.
2. When input A is present, A acts as an *annihilator*: it reacts with the output strand, consuming O and reducing its concentration below the threshold.
3. The fuel reaction is calibrated so that without A, the concentration of O exceeds θ (output = 1), and with A, the annihilation brings $[O]$ below θ (output = 0).

The NOT Gate Challenge

In molecular computing, NOT gates are fundamentally harder than AND/OR gates. AND and OR gates are *signal-triggered*: they respond to the *presence* of a signal. NOT gates must respond to the *absence* of a signal. This asymmetry has deep implications—it means molecular circuits naturally favor *positive logic* (signal presence), which influenced DOGMA's design choice to use activation-based regulation rather than inhibition-based regulation as the default pathway.

2.2.5 NAND and Universal Computation

A single gate type—NAND (NOT-AND)—is *functionally complete*: any Boolean function can be built from NAND gates alone.

NAND truth table:

Input A	Input B	Output $\overline{A \wedge B}$
0	0	**1**
0	1	**1**
1	0	**1**
1	1	0

Building other gates from NAND:

- **NOT from NAND:** $\neg A = A$ NAND A. Connect the same input to both terminals.
- **AND from NAND:** $A \wedge B = \neg(A \text{ NAND } B) = (A \text{ NAND } B)$ NAND $(A \text{ NAND } B)$.
- **OR from NAND:** $A \vee B = (A \text{ NAND } A)$ NAND $(B \text{ NAND } B)$.

This means that if DNA can implement a NAND gate, then *DNA can compute any Boolean function*. Since CRNs are Turing complete (Section 2.8), DNA computing is at least as powerful as silicon computing.

2.2.6 XOR Gate and Parity Detection

The XOR (exclusive OR) gate is particularly useful for arithmetic (it computes the sum bit in binary addition).

XOR truth table:

Input A	Input B	Output $A \oplus B$
0	0	0
0	1	**1**
1	0	**1**
1	1	0

XOR can be expressed using AND, OR, and NOT: $A \oplus B = (A \vee B) \wedge \neg(A \wedge B)$. In DNA, this requires combining an OR gate and an AND gate with an annihilation reaction that suppresses the output when both inputs are high.

2.3 DNA Arithmetic: Computing with Molecules

Now that we have logic gates, we can build arithmetic circuits. This section shows how DNA can perform binary addition, subtraction, and comparison—the foundations of all digital computation.

2.3.1 The Half Adder

A *half adder* adds two single-bit binary numbers and produces a *sum* bit and a *carry* bit.

Half adder truth table:

Input A	**Input** B	**Sum** S	**Carry** C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Notice that **Sum = XOR** and **Carry = AND**:

$$S = A \oplus B \tag{2.2}$$

$$C = A \wedge B \tag{2.3}$$

DNA implementation: A DNA half adder requires two sub-circuits operating in parallel:

- A DNA XOR circuit (from Section 2.2) produces the sum output S.
- A DNA AND circuit produces the carry output C.
- Both circuits share the same input strands A and B.

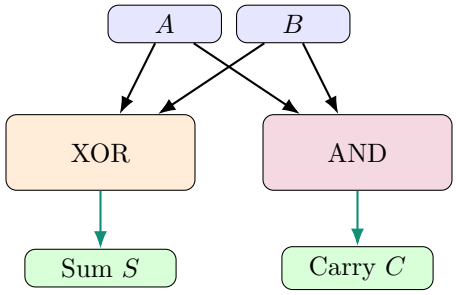


Figure 2.4: DNA half adder: parallel XOR and AND circuits sharing input strands.

2.3.2 The Full Adder

A *full adder* extends the half adder by accepting a carry-in bit C_{in} from a previous stage:

Full adder truth table:

A	B	C_{in}	**Sum** S	**Carry** C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The full adder equations are:

$$S = A \oplus B \oplus C_{\text{in}} \tag{2.4}$$

$$C_{\text{out}} = (A \wedge B) \vee (C_{\text{in}} \wedge (A \oplus B)) \tag{2.5}$$

A full adder can be built from two half adders and one OR gate. By chaining n full adders (with each carry-out feeding the next carry-in), we obtain an *n-bit ripple-carry adder*—capable of adding any two n-bit binary numbers.

Example 2.2 (4-Bit DNA Addition: $5+3=8$)**.** To compute $0101_2 + 0011_2 = 1000_2$ (i.e., $5+3=8$):

1. **Bit 0:** $A_0 = 1, B_0 = 1, C_{\text{in}} = 0$. Sum $= 0$, Carry $= 1$.
2. **Bit 1:** $A_1 = 0, B_1 = 1, C_{\text{in}} = 1$. Sum $= 0$, Carry $= 1$.
3. **Bit 2:** $A_2 = 1, B_2 = 0, C_{\text{in}} = 1$. Sum $= 0$, Carry $= 1$.
4. **Bit 3:** $A_3 = 0, B_3 = 0, C_{\text{in}} = 1$. Sum $= 1$, Carry $= 0$.

Result: $1000_2 = 8_{10}$. ✓

Each bit position would use a separate DNA full adder circuit, with the carry output strand of one circuit acting as the carry input strand of the next.

2.3.3 DNA Subtraction via Two's Complement

Subtraction can be implemented using addition and bitwise inversion. To compute $A - B$:

1. Invert each bit of B using NOT gates: $\bar{B}$.
2. Add 1 to the result: $\bar{B} + 1$ (this is the two's complement of B).
3. Add to A: $A + (\bar{B} + 1) = A - B$.

This means a DNA subtraction circuit requires only NOT gates, AND gates, OR gates, and XOR gates—all of which we have shown can be implemented in DNA.

2.3.4 Comparison and Decision Making

A comparator determines whether $A > B$, $A = B$, or $A < B$. For single-bit comparison:

A	B	$A > B$	$A = B$	$A < B$
0	0	0	**1**	0
0	1	0	0	**1**
1	0	**1**	0	0
1	1	0	**1**	0

These outputs can be computed as:

$$(A > B) = A \wedge \neg B \tag{2.6}$$

$$(A = B) = \neg(A \oplus B) = (A \wedge B) \vee (\neg A \wedge \neg B) \tag{2.7}$$

$$(A < B) = \neg A \wedge B \tag{2.8}$$

Biological Decision Making

Cells make "greater-than" comparisons all the time through competitive binding. When two transcription factors compete for the same promoter site, the one at higher concentration wins—implementing a molecular comparator. DOGMA's Regulation Gate (Chapter 7) uses exactly this principle: competing activation and suppression signals determine whether a genomic region is expressed.

2.4 Strand Displacement Cascades

The logic gates and arithmetic circuits above use a common molecular mechanism: *toehold-mediated strand displacement*. This section explains the mechanism in full detail.

2.4.1 The Three-Strand Mechanism

A strand displacement reaction involves three DNA strands:

- A *gate complex*: a double-stranded region with a short single-stranded *toehold* overhang.
- An *input strand*: complementary to both the toehold and the full gate strand.
- An *output strand*: released when the input strand displaces it from the gate.

The process occurs in three phases:

Phase 1: Toehold binding. The input strand recognizes and binds to the exposed toehold region. This initial binding is weak (5–8 base pairs) but sufficient to bring the input strand into close proximity with the gate complex.

Phase 2: Branch migration. Once the toehold is bound, the input strand begins replacing the output strand through a random walk process called *branch migration*. At each step, one base pair of the old duplex is broken and replaced by a base pair with the incoming strand. This is energetically neutral (breaking and forming equivalent base pairs), so it proceeds by diffusion.

Phase 3: Output release. When branch migration reaches the end of the gate, the output strand is completely displaced and released into solution. The gate now forms a stable duplex with the input strand.

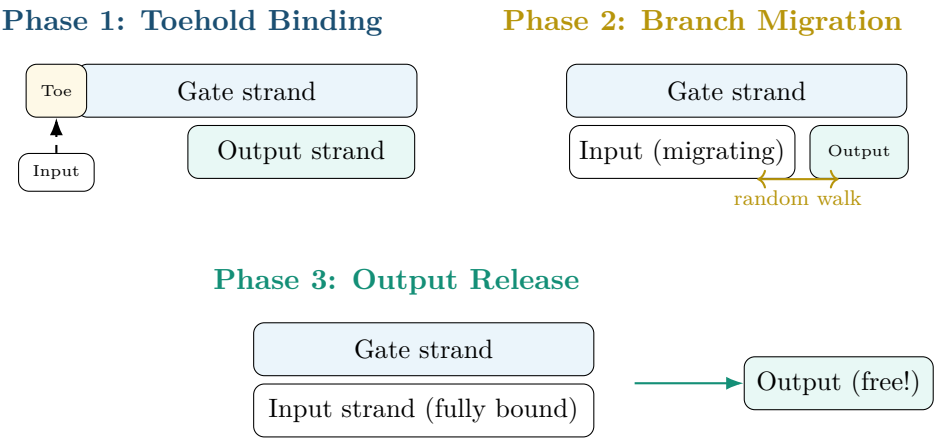


Figure 2.5: The three phases of toehold-mediated strand displacement.

2.4.2 Kinetic Control via Toehold Length

The reaction rate is *exponentially sensitive* to toehold length [Zhang and Seelig, 2011]:

$$k_{\text{forward}} \approx k_{\max} \cdot e^{-\Delta G_{\text{toehold}}/RT} \tag{2.9}$$

where $\Delta G_{\text{toehold}}$ is the free energy of toehold hybridization, R is the gas constant, and T is the temperature. Typical values:

Toehold Length (nt)	Approx. Rate ($\text{M}^{-1}\text{s}^{-1}$)
0 (no toehold)	~ 1 (background)
3	$\sim 10^2$
5	$\sim 10^4$
7	$\sim 10^6$ (maximum)
10+	$\sim 10^6$ (saturated)

This exponential control enables precise timing in molecular circuits—analogous to clock frequency control in digital electronics.

Toehold Length as Attention Score

In DOGMA's strand displacement path, the toehold binding energy ΔG becomes a *learned attention score*. Stronger "toeholds" (higher attention scores) produce faster "displacement" (stronger feature propagation). This maps the four-order-of-magnitude kinetic range of real toeholds into a continuous attention weight:

$$\alpha_{ij} = \text{softmax}\left(\frac{-\Delta G(q_i, k_j)}{RT}\right) \tag{2.10}$$

2.4.3 Seesaw Gates: Scalable Digital Logic

Qian and Winfree [2011] introduced *seesaw gates*, a simplified strand displacement motif that enables large-scale digital circuits. A seesaw gate has a threshold and an amplification mechanism:

- **Threshold:** A fixed amount of threshold strand absorbs input below a cutoff level. Only excess input above the threshold produces output (digital "1").
- **Amplification:** A catalytic mechanism (fuel strand) amplifies weak signals to full output level, restoring signal strength at each gate.
- **Integration:** Multiple input strands are summed (by hybridization to the gate strand), implementing weighted addition.

Qian and Winfree demonstrated circuits with over 100 DNA strands implementing a **4-bit square root computation**—the largest DNA circuit at the time. The circuit reads a 4-bit binary number and outputs its integer square root.

Example 2.3 (4-Bit Square Root Circuit)**.** The circuit computes $\lfloor\sqrt{n}\rfloor$ for 4-bit inputs $n \in \{0, 1, \ldots, 15\}$:

b_3	b_2	b_1	b_0	Decimal n	$\lfloor\sqrt{n}\rfloor$	Output (binary)
0	0	0	0	0	0	00
0	0	0	1	1	1	01
0	1	0	0	4	2	10
1	0	0	1	9	3	11
1	1	1	1	15	3	11

This was implemented entirely in DNA using 130 distinct DNA strands!

2.5 Restriction Enzyme Computing and Splicing Systems

While strand displacement uses designed synthetic strands, a complementary approach exploits *restriction enzymes*—nature's own "molecular scissors"—to cut DNA at specific recognition sites and recombine the fragments. This section follows the framework surveyed by Daley and Kari [2002].

2.5.1 Restriction Enzymes as Programmable Cutters

A Type II restriction endonuclease recognises a short palindromic DNA sequence (typically 4–8 bp) and cleaves both strands. The cut may leave *sticky ends* (overhangs) or *blunt ends*. Two commonly used enzymes:

Enzyme	Recognition (sense strand)	Cut Pattern	Overhang
TaqI	`5'-T↓CGA-3' / 3'-AGC↑T-5'`	5' overhang: `CG`	2-nt sticky
SciNI	`5'-G↓CGC-3' / 3'-CGC↑G-5'`	5' overhang: `GC`	2-nt sticky
EcoRI	`5'-G↓AATTC-3' / 3'-CTTAA↑G-5'`	5' overhang: `AATT`	4-nt sticky

The arrows (↓ / ↑) mark the phosphodiester bonds that are cleaved. After cutting, the sticky ends can hybridise with any compatible overhang, and DNA ligase seals the new backbone. This cut-and-paste cycle is the molecular basis of *splicing*.

2.5.2 Splicing: DNA Recombination as Computation

Head [1987] formalised the cut-and-paste action of restriction enzymes into a computational model called a *splicing system*. The key idea: start with a finite set of DNA strings (the *axioms*) and a set of *splicing rules* (each corresponding to a restriction enzyme + ligase pair); the *language* generated by the system is all strings obtainable by iterated splicing.

Example 2.4 (Restriction-Enzyme Splicing with Real Sequences)**.** Consider two double-stranded DNA molecules and two restriction enzymes (*TaqI* and *SciNI*), following the example of Daley and Kari [2002]:

Strand 1 contains a *TaqI* site (`TCGA`):

```
5'-CCCCCTCGACCCCC-3'
3'-GGGGGAGCTGGGGG-5'
```

Strand 2 contains a *SciNI* site (`GCGC`):

```
5'-AAAAAGCGCAAAAA-3'
3'-TTTTTCGCGTTTTT-5'
```

Step 1 — Cut. Each enzyme recognises its site and cleaves both backbones:

```
5'-CCCCCT  CGACCCCC-3'     5'-AAAAAG  CGCAAAAA-3'
3'-GGGGGAGC  TGGGGG-5'     3'-TTTTTCGC  GTTTTT-5'
```

After cutting, four fragments are produced. The sticky end `CG` from Strand 1's right fragment is complementary to the sticky end from Strand 2's left fragment (both are 2-nt 5' overhangs).

Step 2 — Crosswise ligation. DNA ligase seals the compatible sticky ends in a crosswise fashion, producing two *recombinant* molecules:

```
5'-CCCCCTCGCAAAAA-3'     5'-AAAAAGCGACCCCC-3'
3'-GGGGGAGCGTTTTT-5'     3'-TTTTTCGCTGGGGG-5'
```

The left half of Strand 1 is now joined to the right half of Strand 2, and vice versa. This is a single *splicing step* — and it is a computation, because the output strings encode information derived from both inputs.

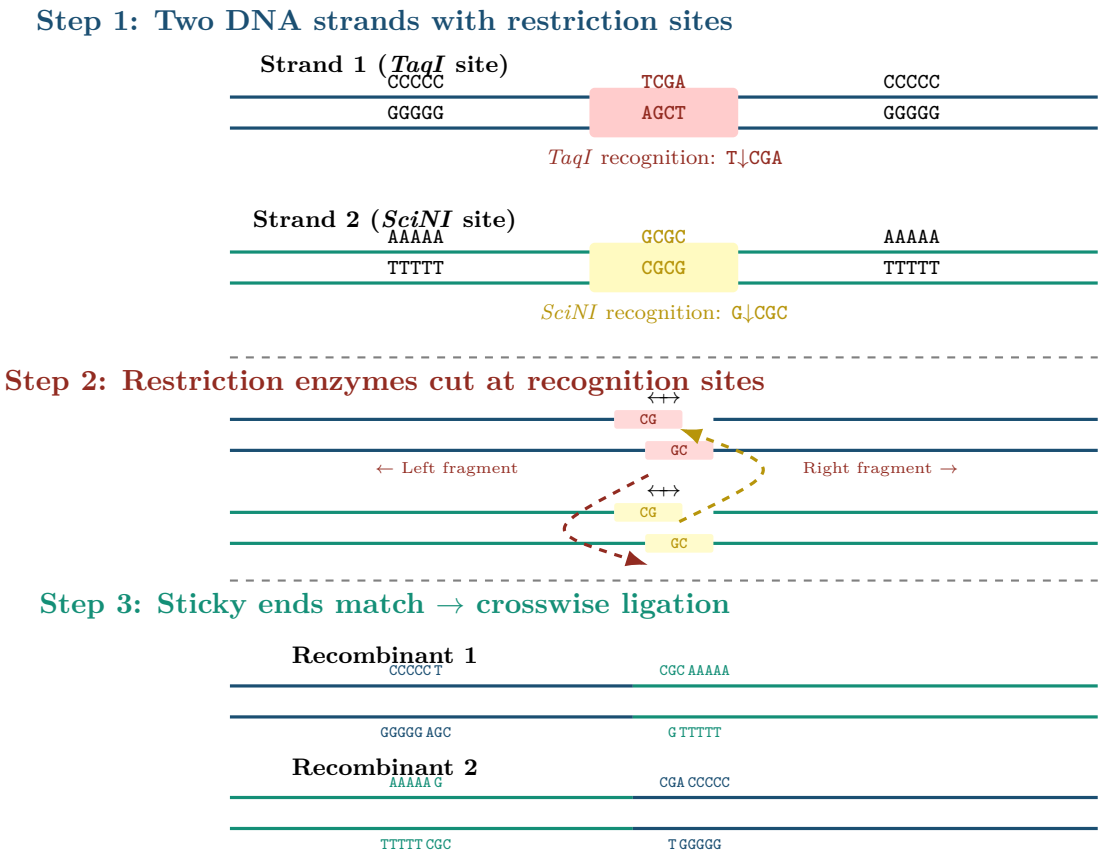


Figure 2.6: Step-by-step splicing: restriction enzymes cut two DNA strands at their recognition sites, producing sticky ends that recombine crosswise to form two new recombinant molecules.

2.5.3 How Splicing Achieves Turing Completeness

To understand *why* splicing is computationally powerful, we follow the argument of Head [1987], refined by Daley and Kari [2002].

Splicing System

A *splicing system* is a triple $\sigma = (\Sigma, I, R)$ where:

- Σ is a finite alphabet (e.g., $\{A, C, G, T\}$),
- $I \subseteq \Sigma^*$ is a finite set of *axiom* strings (initial DNA molecules),
- R is a finite set of *splicing rules* of the form $r = (u_1, u_2; u_3, u_4)$.

A rule r applies to strings $x = x_1 u_1 u_2 x_2$ and $y = y_1 u_3 u_4 y_2$, producing the recombinants $x_1 u_1 u_4 y_2$ and $y_1 u_3 u_2 x_2$. The *language* $L(\sigma)$ is the set of all strings obtainable by iterated application of rules in R to strings in I.

Concrete example: simulating a Turing machine transition. Consider a Turing machine with states $\{q_0, q_1, q_H\}$, tape alphabet $\{0, 1, B\}$ (where B is blank), and the transition rule $\delta(q_0, 1) =$

$(q_1, 0, R)$ (in state q_0, reading 1: write 0, move right, go to q_1). We show how a single splicing step simulates this transition.

Example 2.5 (Simulating a Turing Machine Step with Splicing)**. Step 1: Encode the tape configuration as a string.** The tape configuration "$\ldots B\,1\,\underline{1}\,0\,B\ldots$" with head on the underlined cell in state q_0 is encoded as:

$$w = B\,1\,q_0\,1\,0\,B$$

The state symbol q_0 is inserted immediately *before* the cell the head is reading.

Step 2: Define the splicing rule for this transition. We use the axiom string ("helper oligo"):

$$h = X\,q_1\,0\,Y$$

where X and Y are flanking markers. The splicing rule corresponding to $\delta(q_0, 1) = (q_1, 0, R)$ is:

$$r = (q_0,\ 1;\ q_1,\ 0)$$

Step 3: Apply the splice. We have $w = B\,1\,\underbrace{q_0}_{u_1}\underbrace{1}_{u_2}\,0\,B$ and $h = X\,\underbrace{q_1}_{u_3}\underbrace{0}_{u_4}\,Y$.

Cutting w between q_0 and 1, and cutting h between q_1 and 0, then crosswise ligation gives:

$$w' = B\,1\,q_1\,0\,Y \qquad \text{(new tape: head has moved right, wrote 0, now in } q_1\text{)}$$

The head is now before the cell containing 0 (the old value at the next position) and the state is q_1. After trimming the marker Y, the result encodes the tape after one Turing machine step:

$$B\,1\,0\,q_1\,0\,B$$

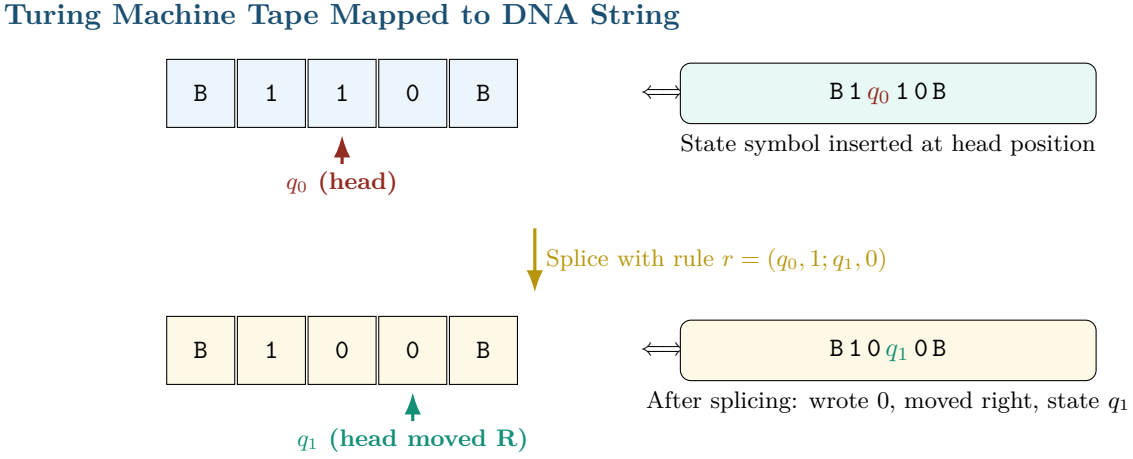


Figure 2.7: A Turing machine tape configuration encoded as a DNA string. One splicing step simulates one Turing machine transition: the state symbol moves, the tape cell is rewritten, and the head advances.

From single steps to Turing completeness. Each Turing machine transition $\delta(q, a) = (q', b, D)$ corresponds to one splicing rule. A Turing machine with k transitions requires k splicing rules and k helper axiom strings. Since every computation is a finite sequence of transitions, iterated splicing can simulate every computation that a Turing machine can perform.

Head [1987] showed that with a *finite* axiom set and *finite* splicing rules, the generated language is always *regular* (a relatively weak class). However, Daley and Kari [2002] observed that when the axiom set is allowed to be a *recursively enumerable language* (or when multisets with copy counts are used, reflecting the reality of molecular concentrations), splicing systems generate *all recursively enumerable languages*—making them Turing complete.

Why Splicing is Turing Complete

The key to Turing completeness lies in the *interaction* between splicing rules. A single rule performs a simple cut-and-paste. But when recombinant products from one rule become inputs to another rule, the system can chain together arbitrarily long computations. This is analogous to how a Turing machine chains transition steps: each step is trivial, but the composition of steps produces universal computation. In molecular terms, the test tube acts as a "memory" that accumulates intermediate results, and the enzyme-ligase cycles act as the "clock" that drives computation forward [Head, 1987, Daley and Kari, 2002].

Splicing Systems Are Turing Complete

Head [1987] proved that finite splicing systems (finite axioms + finite rules) generate regular languages. However, Daley and Kari [2002] note that splicing systems with *multisets* (where each string has a copy count that changes during splicing) generate the *entire class of computable languages*—making them Turing complete. This is remarkable: the cut-and-paste chemistry of restriction enzymes, formalised appropriately, has the same computational power as any silicon computer.

2.5.4 Rothemund's DNA Turing Machine (1996)

Rothemund [1996] proposed a direct implementation of a Turing machine using circular DNA and restriction enzymes. The Turing tape is encoded as a circular single-stranded DNA molecule; the head position and state are encoded as the spacing between a restriction enzyme recognition site and the "current symbol."

A single computation step consists of six biochemical operations:

1. **Cut** the circular DNA with two restriction enzymes (*state* enzyme and *invariant* enzyme), linearising the tape and exposing a sticky end unique to the current state + symbol.
2. **Ligate** the correct *transition oligonucleotide* (one of four per rule, each encoding a new state, symbol, and head direction) to the unique sticky end.
3. **Remove** the protective cap from the other end of the transition oligo.
4. **Re-circularise** the tape by ligating the exposed ends.
5. **Excise** the old symbol using a symbol-excision enzyme.
6. **Re-circularise** again, leaving the tape in the new configuration.

After each cycle, a small aliquot is amplified by **PCR** using a primer matching the *Halt* state. Only halted tapes are amplified, making detection straightforward. This elegant model proves that restriction enzymes, ligase, and PCR—three workhorse techniques of molecular biology—are sufficient for universal computation.

2.5.5 Surface-Based DNA Computing

Liu et al. [2000] demonstrated a practical alternative where DNA strands are affixed to a *gold surface* rather than floating freely in solution. The encoding uses single bases for bits: `A` and `C` for 0, `G` and `T` for 1.

The computation uses a simple set of operations:

- **Mark**(i, b)**:** Hybridise a complementary strand to all surface strands where bit i has value b (single $\rightarrow$ double stranded).
- **Destroy-marked / Destroy-unmarked:** Wash with exonucleases that selectively digest either double-stranded or single-stranded DNA.
- **Unmark:** Apply distilled water (low salinity denatures the marking strand).
- **Test-if-empty:** Remove remaining DNA, amplify with **PCR**, check for product.

Using these operations, Liu et al. [2000] solved a 4-variable, 4-clause 3-SAT instance. The approach was later scaled to a 20-variable SAT problem—the largest DNA computation of its era.

PCR as a Computational Amplifier

Polymerase Chain Reaction (PCR) appears in nearly every DNA computing protocol—not as a gate, but as a *readout amplifier*. Just as a sense amplifier in DRAM converts a tiny charge difference into a digital 1 or 0, PCR exponentially amplifies the molecular "answer" (2^n copies after n cycles) so it can be detected by gel electrophoresis or fluorescence. DOGMA abstracts this idea as the *Express* stage: amplifying selected features for downstream readout.

2.6 Whiplash PCR: Hairpin-Based State Machines

All the models above are *multi-strand*: the computation happens between distinct DNA molecules. *Whiplash PCR* [Sakamoto et al., 2000] takes a radically different approach: a *single* DNA strand acts as both the program *and* the data.

2.6.1 The Hairpin State Machine

The state machine is encoded in a single-stranded DNA molecule. The $3'$ end of the molecule represents the *current state*. The body of the strand contains the transition table, encoded as a series of subsequences in the format:

```
[stop]-[new_state]-[old_state]
```

Because the molecule is single-stranded, the $3'$ tail (current state) will attempt to fold back and hybridise with a complementary `old_state` subsequence within the same molecule, forming a *hairpin* structure:

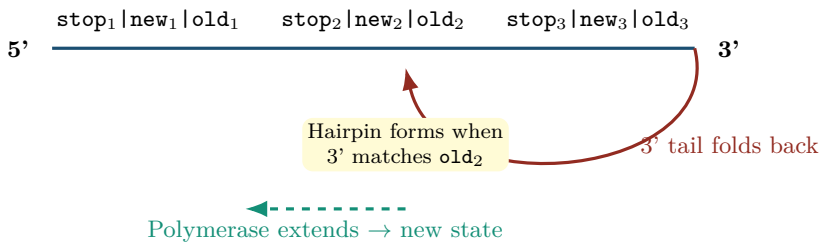


Figure 2.8: Whiplash PCR: a single DNA strand folds into a hairpin when the 3′ tail (current state) recognises a complementary transition entry. DNA polymerase extends the tail, copying the new-state sequence and advancing the computation.

2.6.2 Computation Cycle

Each "clock cycle" of the Whiplash PCR machine consists of three thermal steps:

1. **Anneal** (∼55° C): The 3′ tail folds back, scanning the strand for a complementary `old_state` subsequence. When found, a hairpin forms.

2. **Extend** (∼72° C): DNA polymerase extends the 3′ end, copying the `new_state` sequence. A `stop` signal (e.g., a non-standard base or a hairpin in the template) halts extension.

3. **Denature** (∼95° C): Heat melts the hairpin, releasing the 3′ tail—which now encodes the *new state*.

After one thermal cycle, the state has transitioned from `old_state` to `new_state`. By running n cycles in a standard PCR thermocycler, the machine executes n state transitions.

Massive Parallelism in a Single Pot

Whiplash PCR is *autonomous*: each molecule independently executes its own state machine. If 10^{12} copies of the same program strand are present in a 10 µL tube, all 10^{12} machines execute in parallel. Moreover, different program strands can coexist in the same tube, implementing distinct computations simultaneously—a molecular form of MIMD (Multiple Instruction, Multiple Data) computing.

Whiplash PCR and DOGMA's Synthesise Stage

DOGMA's Synthesise stage (Chapter 7) mirrors the Whiplash PCR principle. The model maintains an internal "state" (hidden vector) that is iteratively updated by scanning learned transition rules (weight matrices). Each forward-pass step is analogous to one thermal cycle: the current state "folds back" to find the best-matching rule, which then extends the state to produce the next hidden representation.

2.7 DNA Nanotechnology and Structural Computing

Beyond logic circuits, DNA can serve as a structural material for building nanoscale architectures.

2.7.1 DNA Origami

Rothemund [2006] demonstrated *DNA origami*: a long single-stranded "scaffold" strand (∼7000 nucleotides, typically from the M13 bacteriophage genome) is folded into a target shape by ∼200 short "staple" strands (∼32 nt each). Each staple binds to specific non-contiguous regions of the scaffold, physically pulling those regions together.

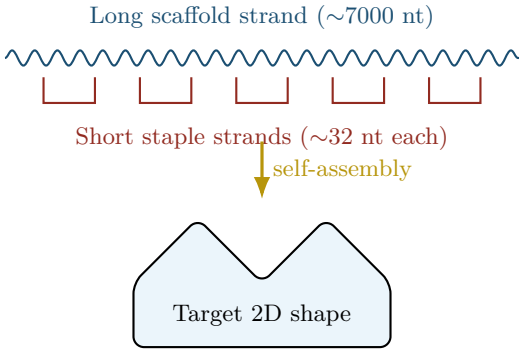


Figure 2.9: DNA origami: staple strands fold a long scaffold into a prescribed 2D shape.

DNA origami demonstrates that *structure can be programmed through sequence*: a designer specifies the desired geometry, and a compiler computes the staple sequences needed to fold the scaffold into that shape.

2.7.2 Tile Self-Assembly and Turing Completeness

Winfree et al. [1998] showed that DNA tiles—small DNA structures with programmable sticky ends—can self-assemble into two-dimensional crystals that implement computational patterns.

Definition 2.2 (Wang Tile System)**.** A *Wang tile system* consists of a finite set of tile types T, each with four labeled edges (north, south, east, west). Tiles are placed on the integer lattice $\mathbb{Z}^2$ such that adjacent edges must carry the same label. A tile system is *computationally universal* if it can simulate any Turing machine through its growth pattern.

Winfree showed that DNA tiles physically instantiate Wang tiles, proving that *self-assembly can compute*. The growth pattern of a DNA crystal can encode the execution trace of a Turing machine.

This principle—*local interactions producing global computation*—reappears in DOGMA's TetraMemory (Chapter 10), where local tetrahedral interactions propagate information without global attention.

2.8 Chemical Reaction Networks: Turing-Complete Chemistry

2.8.1 Formal Definition

Chemical Reaction Networks (CRNs) provide a mathematical abstraction that captures the computational essence of molecular systems.

Definition 2.3 (Chemical Reaction Network)**.** A *Chemical Reaction Network* (CRN) is a pair $(\mathcal{S}, \mathcal{R})$ where:

- $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ is a finite set of *species*.

- $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ is a finite set of *reactions*, each of the form:

$$\alpha_1 S_1 + \alpha_2 S_2 + \cdots \xrightarrow{k} \beta_1 S_1 + \beta_2 S_2 + \cdots \tag{2.11}$$

  where $\alpha_i, \beta_i \in \mathbb{N}_0$ are stoichiometric coefficients and $k > 0$ is the rate constant.

Under *mass-action kinetics*, the rate of each reaction is proportional to the product of reactant concentrations:

$$v_j = k_j \prod_{i:\alpha_{ij}>0} [S_i]^{\alpha_{ij}} \tag{2.12}$$

2.8.2 Computing with CRNs: Step-by-Step Examples

Addition: The reaction $A + B \xrightarrow{k} C$ computes $[C] = [A]_0 + [B]_0$ when the reaction goes to completion (assuming $[C]_0 = 0$ and A, B are not consumed elsewhere).

Actually, proper CRN addition uses *catalytic reactions*:

$$A \xrightarrow{k} A + C \quad \text{(A produces C without being consumed)} \tag{2.13}$$

$$B \xrightarrow{k} B + C \quad \text{(B produces C without being consumed)} \tag{2.14}$$

After time t, $[C](t) \approx k \cdot t \cdot ([A]_0 + [B]_0)$. By calibrating time, this computes addition.

Multiplication: The catalytic reaction system:

$$A + B \xrightarrow{k} A + B + C \tag{2.15}$$

produces C at a rate proportional to $[A] \cdot [B]$. At equilibrium, $[C] \propto [A]_0 \cdot [B]_0$, computing multiplication.

Max (comparison): A "winner-take-all" annihilation reaction:

$$A + B \xrightarrow{k} \emptyset \tag{2.16}$$

When $[A]_0 > [B]_0$, species B is completely consumed and $[A]$ remains at $[A]_0 - [B]_0 > 0$. When $[B]_0 > [A]_0$, species A is consumed. The surviving species indicates which initial concentration was larger.

Example 2.6 (CRN Toggle Switch (Memory))**.** A bistable toggle switch remembers a binary state using two mutually repressing species:

$$A + B \xrightarrow{k_1} A \quad (A \text{ consumes } B) \tag{2.17}$$

$$B + A \xrightarrow{k_2} B \quad (B \text{ consumes } A) \tag{2.18}$$

The system settles into one of two stable states: high $[A]$/low $[B]$ (storing "1") or low $[A]$/high $[B]$ (storing "0"). External signals can flip the state, implementing molecular memory [Gardner et al., 2000].

2.8.3 Turing Completeness of CRNs

Theorem 2.1 (CRN Turing Completeness [Soloveichik et al., 2010])**.** For any Turing machine M, there exists a CRN $(\mathcal{S}, \mathcal{R})$ that simulates M with arbitrarily high probability. The CRN can be made *rate-independent*: the final output depends only on initial species concentrations and the reaction graph, not on the specific rate constants.

This result is profound: *chemistry itself is computationally universal*. Any computation that a silicon chip can perform, a properly designed set of molecular reactions can also perform (though much more slowly).

2.8.4 Deterministic vs. Stochastic Semantics

CRNs can be analyzed under two semantics:

1. **Deterministic:** Species concentrations evolve as continuous quantities governed by ordinary differential equations (Equation 2.12). Accurate for large molecule counts ($> 10^3$).

2. **Stochastic:** Individual molecular events are modeled as a continuous-time Markov chain via the Chemical Master Equation. Accurate for small molecule counts, where individual reaction events matter.

Why CRNs Matter for DOGMA

DOGMA uses CRNs as neural network layers (Chapter 14, Paper VI). The key insight is that mass-action kinetics (Equation 2.12) is *differentiable*—we can compute gradients through it. By parameterizing the rate constants k_j and learning them via backpropagation, DOGMA turns chemistry into a trainable computation. The reactions encode the *structure* of computation; the rate constants encode the *learned parameters*.

2.9 DNA Neural Networks

Cherry and Qian [2018] achieved a landmark result: a DNA-based neural network that performs pattern recognition entirely in molecular form.

2.9.1 Architecture

The network classifies handwritten digits using three layers:

1. **Input layer:** Each pixel of a simplified 10×10 binary image is encoded as the concentration of a specific DNA species. A "bright" pixel has high concentration; a "dark" pixel has zero concentration.

2. **Weight layer:** Each weight w_{ij} connecting input i to output j is encoded as the concentration of a gate strand. The gate strand releases output species j when triggered by input species i, with the amount released proportional to w_{ij}.

3. **Winner-take-all (WTA) layer:** Output species compete through mutual annihilation reactions. The species with the highest weighted sum survives, representing the network's classification.

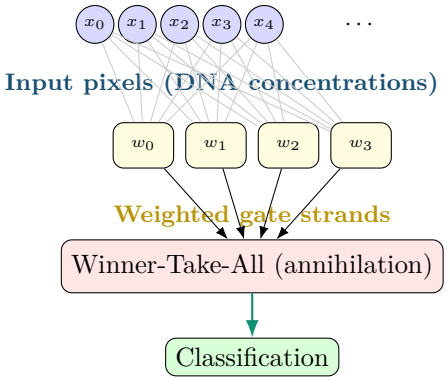


Figure 2.10: Architecture of a DNA neural network for pattern classification.

2.9.2 How the Winner-Take-All Works

The WTA mechanism is particularly elegant. Suppose we have three output species Y_1, Y_2, Y_3 representing three possible digits:

1. After the weight layer, concentrations might be: $[Y_1] = 80$ nM, $[Y_2] = 40$ nM, $[Y_3] = 20$ nM.

2. Annihilation reactions run: $Y_1 + Y_2 \to$ waste, $Y_1 + Y_3 \to$ waste, $Y_2 + Y_3 \to$ waste.

3. Because $[Y_1]$ is highest, it survives the competition. After all reactions complete: $[Y_1] = 20$ nM, $[Y_2] = 0$, $[Y_3] = 0$.

4. A reporter fluorescent strand bound to Y_1 glows green → the digit is classified as "1".

WTA in DOGMA

DOGMA's Regulation Gate implements a differentiable analogue of WTA. Competing expression and suppression signals determine which genomic regions are "expressed" (active) and which are "silenced". The competitive dynamics are captured by softmax: $\text{softmax}(\mathbf{z})_i = e^{z_i} / \sum_j e^{z_j}$, which is the continuous-valued generalization of WTA.

2.9.3 Significance for DOGMA

The Cherry–Qian DNA neural network demonstrates three principles central to DOGMA:

1. **Computation through molecular interaction.** Strand displacement implements weighted summation and competitive selection—the core operations of neural networks—without any silicon.

2. **Thermodynamic binding as attention.** The specificity of strand displacement (controlled by toehold sequence and length) is functionally equivalent to attention scores in transformers. Strong toeholds produce high attention; weak toeholds produce low attention.

3. **Winner-take-all as selection.** The competitive annihilation mechanism is analogous to softmax followed by argmax.

2.10 Molecular Programming Languages

As DNA computing matured, researchers developed software tools that abstract away molecular details.

2.10.1 Visual DSD

The DNA Strand Displacement (DSD) language provides a domain-specific programming language for specifying strand displacement circuits [Zhang and Seelig, 2011]. A DSD program describes species, domains, and reactions; a compiler translates this into specific DNA sequences and predicts system dynamics through simulation.

2.10.2 NUPACK

NUPACK (Nucleic Acid Package) performs thermodynamic analysis of nucleic acid systems [Zadeh et al., 2011]:

- Minimum free energy (MFE) secondary structures
- Partition functions and base-pairing probabilities
- Equilibrium concentrations of multi-strand complexes
- Sequence design to achieve target structures

2.10.3 The Compiler Analogy

The molecular programming toolchain follows the same architecture as a software compiler:

Software Compiler	**Molecular Compiler**	**DOGMA Analogue**
High-level language	Circuit specification (DSD)	Prompt bundle
Intermediate repr.	Domain-level design	Genomic sequence
Assembly	Nucleotide sequences	Codon opcodes
Machine code	Synthesized DNA	Compiled execution plan
Hardware execution	Wet-lab experiment	Neural forward pass

DOGMA's CodonForge compiler (Chapter 11) follows this architecture explicitly.

2.11 From Molecules to Silicon: The DOGMA Bridge

This chapter has established that DNA can implement logic gates, arithmetic circuits, neural networks, and Turing-complete computation. The following table summarizes how each molecular computing primitive maps to a DOGMA component:

DNA Primitive	Molecular Mechanism	DOGMA Component
Watson-Crick pairing	Complementary base binding	Thermodynamic attention scores
Toehold binding	Kinetically-controlled initiation	Learned attention weights
Strand displacement	Signal transduction cascade	Strand Displacement path
Branch migration	Random walk replacement	Feature propagation
AND gate	Sequential displacement	Multiplicative gating
OR gate	Parallel displacement	Additive pooling
NOT gate	Threshold + annihilation	Regulation gate suppression
CRN computation	Mass-action ODEs	CRN neural layers
Winner-take-all	Competitive annihilation	Softmax selection
Self-assembly	Local tile rules	TetraMemory propagation
Codon translation	$64 \rightarrow 21$ mapping	CodonForge opcode table
Gene regulation	Promoter/enhancer control	Epigenetic memory

The DOGMA Thesis

DOGMA does not simulate DNA computing on silicon. It *abstracts* the computational principles of molecular systems—massive parallelism, thermodynamic binding, competitive selection, self-organized structure—and reimplements them using linear algebra, differentiable programming, and gradient-based optimization. The *architecture* is biological; the *substrate* is silicon.

2.12 The State of the Art and Open Challenges

As of 2026, DNA computing has achieved:

- Circuits with hundreds of distinct molecular species
- Neural network inference on molecular substrates [Cherry and Qian, 2018]
- Programmable molecular robots that walk on DNA origami surfaces
- In vivo computation inside living cells (using CRISPR-based logic)
- DNA data storage with random access at the petabyte scale

Fundamental challenges remain:

- **Speed:** DNA reactions operate on timescales of minutes to hours, vs. nanoseconds for silicon.

- **Error rates:** Molecular interactions have inherent stochasticity.
- **Scalability:** Crosstalk between strands increases with circuit complexity.
- **Readout:** Extracting results from molecular mixtures remains slow and expensive.

2.13 Exercises

2.1. [**Truth Table**] Complete the truth table for a 3-input majority gate (output is 1 when at least 2 of 3 inputs are 1). Express the output as a Boolean formula using AND and OR. How many DNA strand displacement gates would you need?

2.2. [**Design**] Design (on paper) a DNA strand displacement AND gate. Specify the sequences for two input strands, a gate complex, and the expected output strand. Explain how the gate fails if only one input is present.

2.3. [**Arithmetic**] Using the half adder and full adder circuits described in this chapter, trace the computation of $7 + 5 = 12$ in binary ($0111 + 0101 = 1100$). Show the sum and carry at each bit position.

2.4. [**Analysis**] In Adleman's 1994 experiment, how many distinct path encodings existed for his 7-vertex graph? Estimate the mass of DNA needed if the experiment were scaled to 20 vertices.

2.5. [**CRN Programming**] Design a CRN that computes the minimum of two concentrations: given initial concentrations $[A]_0$ and $[B]_0$, produce an output species C with $[C] = \min([A]_0, [B]_0)$. *Hint:* Use mutual annihilation followed by catalytic production.

2.6. [**Theory**] Prove that a CRN consisting of the reactions $A+B \to C$ and $C \to A+B$ implements a reversible binary counter for species C (in the deterministic regime).

2.7. [**Coding**] Implement a simple CRN simulator in Python that takes a set of reactions and initial concentrations, and integrates the mass-action ODEs using Euler's method. Test it on a toggle switch circuit with two mutually repressing species.

2.8. [**Conceptual**] Compare the "winner-take-all" mechanism in Cherry and Qian's DNA neural network with the softmax function in transformers. What are the key similarities and differences?

2.9. [**Research**] Read Qian and Winfree [2011]. How does the seesaw gate architecture achieve modularity? Why is modularity important for scaling molecular circuits?

2.10. [**Challenge**] Design a DNA circuit that computes the XOR of *three* inputs ($A \oplus B \oplus C$). Provide the complete truth table and sketch the gate-level architecture.

2.11. [**Splicing**] Given the two double-stranded DNA molecules `5'-AAAGAATTCBBB-3'` and `5'-CCCGAATTCDDD-3'`, and the restriction enzyme *EcoRI* (recognition site `G↓AATTC`), write out all four fragments produced by cutting, then show the two recombinant molecules produced by crosswise ligation. Which parts of the original strands are now combined?

2.12. **[Whiplash PCR]** Design a Whiplash PCR program strand that implements a 3-state counter (states $q_0 \to q_1 \to q_2 \to q_0$). Write out the transition table segments (`stop-new-old`) and sketch the hairpin that forms when the $3'$ tail is in state q_1. How many thermal cycles are needed to return to q_0?

2.14 Key Takeaways

Key Takeaways

- DNA computing began with Adleman's 1994 Hamiltonian path experiment, demonstrating massive molecular parallelism with 10^{18} simultaneous computations.
- DNA logic gates (AND, OR, NOT, NAND, XOR) are implemented using real DNA sequences with toehold-mediated strand displacement; each gate uses specific nucleotide strands with 6–8 nt toeholds controlling reaction kinetics.
- From logic gates, DNA can build complete arithmetic circuits: half adders, full adders, and multi-bit ripple-carry adders capable of binary addition, subtraction, and comparison.
- Strand displacement is the key computational primitive: toehold length provides exponential kinetic control (analogous to clock speed in silicon).
- Restriction enzymes provide programmable molecular scissors: splicing systems (Head, 1987) formalise the cut-and-paste recombination of real enzyme sites (TaqI, EcoRI, SciNI) into a Turing-complete computational model.
- Whiplash PCR enables autonomous single-molecule state machines: each DNA strand encodes both program and data, executing transitions through hairpin formation and polymerase extension in a standard thermocycler.
- Chemical Reaction Networks (CRNs) are provably Turing complete, establishing molecular computation as fundamentally universal.
- DNA neural networks demonstrate that thermodynamic binding can implement attention-like weighted computation with winner-take-all classification.
- DOGMA reimplements these biological computational principles on silicon, mapping each molecular mechanism to a differentiable neural component.

Suggested Reading

- Adleman [1994]: The founding paper of DNA computing.
- Daley and Kari [2002]: Comprehensive survey of DNA computing models and implementations (splicing systems, Whiplash PCR, surface-based computing, equality machines).
- Zhang and Seelig [2011]: Comprehensive review of strand displacement.
- Soloveichik et al. [2010]: CRN Turing completeness proof.
- Cherry and Qian [2018]: DNA-based neural networks.

- Qian and Winfree [2011]: Seesaw gates and the 4-bit square root circuit.
- Head [1987]: The original formal model of splicing systems.
- Seeman [2003]: DNA nanotechnology foundations.
- Rothemund [2006]: DNA origami.
- Sakamoto et al. [2000]: Whiplash PCR and hairpin-based molecular computation.

Chapter 3

Gene Regulation as Computation

> *The genome is not a blueprint; it is a regulatory program that produces different outputs depending on context.*
>
> — After François Jacob

Every cell in a human body contains the same genome—approximately 3.2 billion base pairs, encoding roughly 20,000 protein-coding genes. Yet a neuron looks and functions nothing like a liver cell, a muscle fiber, or an immune cell. The difference is not in what genes are *present*, but in which genes are *expressed*—and when, where, how much, and in response to what signals.

Gene regulation is the computational system that controls this selective expression. It is, arguably, the most sophisticated information-processing system in biology—and the one that DOGMA most directly emulates. In this chapter, we examine gene regulation not as a topic in biochemistry, but as a *computational architecture*: one with logic gates, conditional branching, feedback loops, memory, and context-dependent program execution.

Understanding gene regulation is essential for understanding DOGMA's design. The six-stage pipeline (Chapter 7) is a direct formalization of the regulatory logic that cells use to convert persistent genomic state into context-appropriate behavior.

3.1 The Regulatory Grammar of DNA

Gene regulation operates through *cis-regulatory elements*—DNA sequences that control transcription of nearby genes—and *trans-acting factors*—proteins and RNAs that bind these elements. Together, they form a regulatory grammar with well-defined syntax and semantics [Alberts et al., 2022, ENCODE Project Consortium, 2012].

Think of it this way: if the genome is a book, then cis-regulatory elements are the punctuation, chapter headings, and margin notes that tell the reader *how to read* the text. Trans-acting factors are the readers themselves—molecular machines that interpret these instructions.

3.1.1 Promoters: The Entry Point

A *promoter* is a DNA sequence (typically 100–1000 bp) located upstream of a gene. It serves as the binding site for RNA polymerase and associated transcription factors. The promoter determines *whether* a gene can be transcribed and sets the *basal* transcription rate.

Promoter as Conditional Gate

In computational terms, a promoter is a *conditional gate*: it permits downstream execution (transcription) only when the appropriate activation signals (transcription factors) are present. The strength of the promoter determines the gate's threshold—how much signal is needed to initiate transcription.

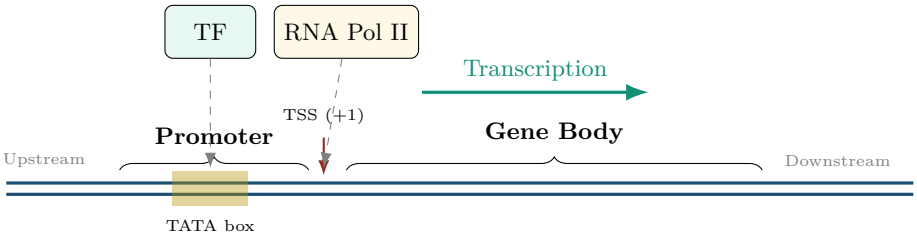


Figure 3.1: A gene promoter region. Transcription factors (TFs) bind to specific sites within the promoter, recruiting RNA Polymerase II to the transcription start site (TSS). Transcription proceeds downstream through the gene body.

3.1.2 Enhancers: Remote Amplifiers

Enhancers are regulatory sequences that can increase transcription of a target gene from distances of up to one megabase (1 million bp). They function by looping the intervening DNA to bring the enhancer-bound proteins into contact with the promoter [ENCODE Project Consortium, 2012].

Enhancers implement *long-range conditional amplification*: they increase the activation of specific genes based on cellular context, regardless of their linear distance on the chromosome. This is analogous to an attention mechanism that selectively amplifies relevant features regardless of sequence position—a parallel that DOGMA's regulation stage exploits.

3.1.3 Silencers and Insulators

Silencers are the complement of enhancers: they recruit repressor proteins that suppress transcription. *Insulators* create boundaries that prevent regulatory signals from crossing into adjacent domains, functioning as scope delimiters in the regulatory program.

Element	Biological Function	Computational Analogue	DOGMA Mapping
Promoter	Initiates transcription	Conditional entry point	Activation gate in Regulation stage
Enhancer	Amplifies transcription	Remote attention / boost	Context-dependent amplification
Silencer	Represses transcription	Inhibitory gate	Suppression in activation map
Insulator	Blocks cross-domain effects	Scope delimiter	Region boundary in genome

3.2 Transcription Factor Binding: Step by Step

Transcription factors are the molecular workhorses of gene regulation. Understanding *how* they work—at a mechanistic level—reveals the computational logic that DOGMA formalizes.

3.2.1 How Transcription Factors Bind DNA

A transcription factor (TF) is a protein with two key domains:

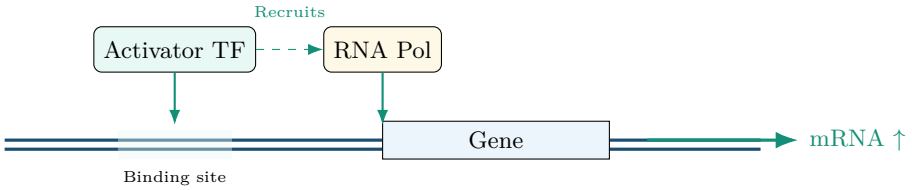
1. **DNA-binding domain (DBD):** A structural motif that recognizes and binds a specific short DNA sequence (typically 6–12 bp), called a *binding motif* or *response element*.
2. **Effector domain:** A region that interacts with other proteins to either *activate* or *repress* transcription.

The binding process works as follows:

1. The TF diffuses through the nucleus, sampling DNA sequences by sliding along the double helix.
2. When the DBD encounters its target motif, it forms hydrogen bonds and van der Waals contacts with the DNA bases in the major groove.
3. The TF-DNA complex is stabilized, and the effector domain recruits co-activators or co-repressors.
4. If the TF is an *activator*, it helps recruit RNA polymerase to the promoter, increasing transcription. If it is a *repressor*, it blocks polymerase access or recruits chromatin-modifying enzymes that compact the DNA.

3.2.2 Activators vs. Repressors

(a) Activator



(b) Repressor

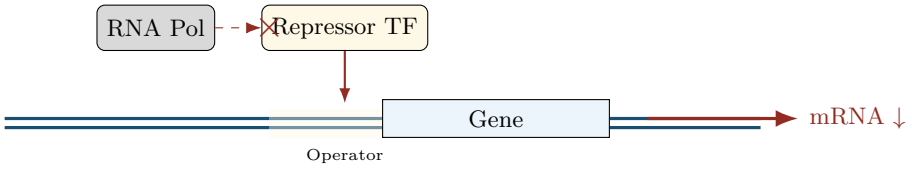


Figure 3.2: (a) An activator TF binds its target site, recruits RNA Polymerase, and increases transcription. (b) A repressor TF binds the operator, physically blocking RNA Polymerase and decreasing transcription.

3.2.3 Cooperative Binding and the Hill Function

A single transcription factor binding event is often not enough to switch a gene on or off. In practice, multiple copies of a TF must bind cooperatively—each binding event makes the next more likely. This produces a *sigmoidal* (S-shaped) response curve, described by the **Hill equation**:

$$f(x) = \frac{x^n}{K^n + x^n}, \tag{3.1}$$

where:

- x = concentration of the transcription factor,
- K = dissociation constant (the concentration at which the gene is half-maximally activated),
- n = Hill coefficient (controls the steepness of the response).

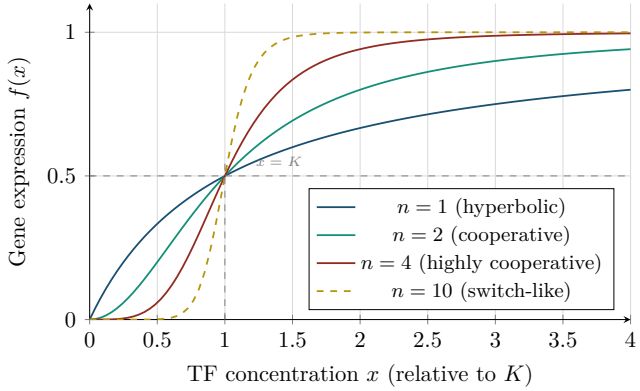


Figure 3.3: The Hill function for different Hill coefficients n. As n increases, the response becomes increasingly switch-like. At $x = K$, the output is always 0.5 (half-maximal). For $n = 1$, the response is gradual (Michaelis–Menten kinetics). For large n, the response approximates a digital step function.

Example 3.1 (Worked Example: Hill Function Calculation)**.** Consider a gene regulated by a transcription factor with $K = 10$ nM and $n = 3$. What fraction of maximal expression do we get at concentrations $x = 5, 10, 15, 20$ nM?

Using $f(x) = \frac{x^n}{K^n + x^n} = \frac{x^3}{10^3 + x^3}$:

x (nM)	x^3	$K^3 + x^3$	$f(x)$	
5	125	1125	$125/1125 = 0.111$	(11%)
10	1000	2000	$1000/2000 = 0.500$	(50%)
15	3375	4375	$3375/4375 = 0.771$	(77%)
20	8000	9000	$8000/9000 = 0.889$	(89%)

Notice the steep transition: doubling x from 5 to 10 nM raises expression from 11% to 50%, and going from 10 to 20 nM raises it from 50% to 89%. This steep, switch-like behavior is what allows cells to make sharp decisions from graded chemical signals.

Why Cooperativity Matters

Without cooperativity ($n = 1$), a gene responds gradually to TF concentration—like a dimmer switch. With cooperativity ($n \geq 2$), the response becomes sigmoidal—more like a toggle switch. This is biologically essential because cells must make *discrete decisions* (divide or not, differentiate or not, die or not) from *continuous signals*. The Hill function is biology's analog-to-digital converter. DOGMA's activation functions in the Regulation stage play an analogous role.

3.3 Transcription Factors as Logic Gates

Transcription factors (TFs) are proteins that bind specific DNA sequences and either activate or repress transcription. They are the *logic gates* of gene regulation, implementing combinatorial Boolean functions over input signals [Alon, 2019].

3.3.1 Combinatorial Regulation

Most genes are regulated by multiple TFs simultaneously. The promoter and enhancer regions of a gene contain binding sites for several different TFs, and the transcriptional output depends on the *combination* of TFs present. This is *combinatorial logic*:

- **AND logic:** Gene X is expressed only if both TF-A **and** TF-B are present. This occurs when the promoter requires cooperative binding of both factors.
- **OR logic:** Gene X is expressed if TF-A **or** TF-B (or both) are present. This occurs when either factor alone can recruit RNA polymerase.
- **NOT logic:** Gene X is expressed unless repressor TF-C is present. The repressor physically blocks polymerase access or recruits chromatin-modifying enzymes.
- **Complex functions:** Real promoters implement complex Boolean functions like $(A \wedge B) \vee (C \wedge \neg D)$—requiring multiple TF binding sites with specific spatial arrangements and cooperative interactions.

3.3.2 The Lac Operon: A Biological AND Gate

The *lac operon* in *E. coli*, discovered by Jacob and Monod [Jacob and Monod, 1961], is the canonical example of gene regulation as computation. It controls the expression of genes needed to metabolize lactose.

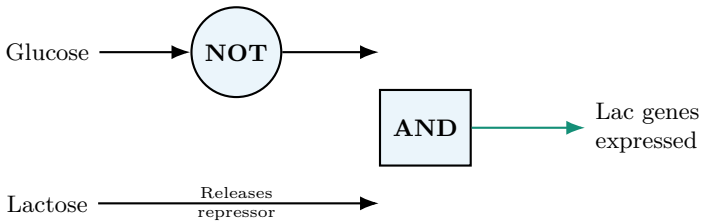
The logic is elegant: the cell should only invest energy in making lactose-digesting enzymes when two conditions are *both* true: (1) lactose is available, and (2) the preferred sugar, glucose, is absent. This is an AND gate with one inverted input.

```
# Biological lac operon logic (simplified)
def lac_operon_expression(glucose_present: bool, lactose_present: bool)
    -> bool:
    """Returns True if lac genes are expressed."""
    # CAP activator binds only when glucose is absent (cAMP high)
    cap_active = not glucose_present
    # Lac repressor releases only when lactose (allolactose) is present
```

```
    repressor_inactive = lactose_present
    # Expression requires BOTH conditions
    return cap_active and repressor_inactive
```

Listing 3.1: The lac operon expressed as pseudocode.

The lac operon implements $express = (\neg glucose) \wedge lactose$—a two-input logic gate with biological NOT and AND operations. This is not a metaphor: the molecular interactions are functionally identical to a logic circuit.



Truth Table

Glu	Lac	Express
0	0	0
0	1	**1**
1	0	0
1	1	0

Figure 3.4: The lac operon as a logic circuit. Glucose is inverted (NOT gate) because its *absence* activates CAP. The AND gate requires both CAP activation and lactose presence. The truth table confirms: expression occurs only when glucose is absent *and* lactose is present.

Biological Economics

The lac operon logic is economically optimal: the cell expends energy to make lactose-metabolizing enzymes only when lactose is available and glucose (the preferred sugar) is not. This cost-aware regulation is a form of *resource-optimal computation*—a principle that DOGMA inherits through its efficiency pressure in the Tera regime (Chapter 9).

Example 3.2 (Worked Example: Lac Operon Conditions)**.** Consider *E. coli* growing in a medium. Predict lac operon expression:

(a) **Glucose only:** Glucose = 1, Lactose = 0. Expression = $(\neg 1) \wedge 0 = 0 \wedge 0 = 0$. No expression—the cell uses glucose, its preferred carbon source.

(b) **Lactose only:** Glucose = 0, Lactose = 1. Expression = $(\neg 0) \wedge 1 = 1 \wedge 1 = 1$. Full expression—the cell makes enzymes to digest the only available sugar.

(c) **Both sugars:** Glucose = 1, Lactose = 1. Expression = $(\neg 1) \wedge 1 = 0 \wedge 1 = 0$. No expression—why waste energy on lactose enzymes when glucose is available?

(d) **Neither sugar:** Glucose = 0, Lactose = 0. Expression = $(\neg 0) \wedge 0 = 1 \wedge 0 = 0$. No expression—no substrate to metabolize.

This is a biological optimization: the cell only manufactures proteins when they are both *needed* and *useful*.

3.4 Gene Regulatory Networks

Individual regulatory interactions connect into *gene regulatory networks* (GRNs)—directed graphs where nodes are genes and edges represent regulatory relationships (activation or repression). GRNs exhibit recurring structural motifs with characteristic dynamical behaviors [Alon, 2019].

3.4.1 Network Motifs

Alon [2019] identified several common network motifs:

1. **Negative autoregulation:** A gene represses its own transcription. This motif speeds up response time and reduces cell-to-cell variability in gene expression levels. *Computational analogue:* self-normalizing activation function.

2. **Positive feedback loop:** A gene activates its own transcription. This creates bistability—the cell commits to one of two stable states (on or off). *Computational analogue:* binary memory element.

3. **Feed-forward loop:** Gene A activates gene B, and both A and B regulate gene C. Depending on the signs (activation or repression), this motif can implement persistence detection (filtering out transient signals) or pulse generation. *Computational analogue:* temporal filter / edge detector.

4. **Toggle switch:** Two genes mutually repress each other, creating a bistable switch [Gardner et al., 2000]. *Computational analogue:* SR latch / binary memory.

5. **Oscillator:** Three (or more) genes form a repression ring, producing oscillatory expression [Elowitz and Leibler, 2000]. *Computational analogue:* clock circuit.

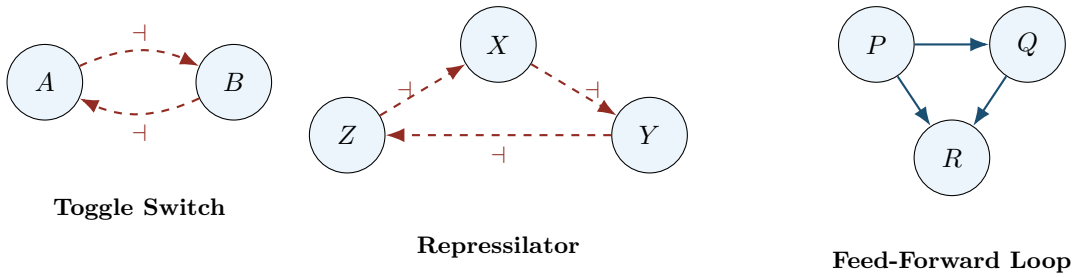


Figure 3.5: Common gene regulatory network motifs. Solid arrows indicate activation; dashed arrows indicate repression. Each motif implements a distinct computational function.

3.4.2 The Repressilator: A Biological Ring Oscillator

The *repressilator* [Elowitz and Leibler, 2000] is one of the most elegant examples of biological computation. It consists of three genes arranged in a ring, where each gene represses the next:

$$X \dashv Y \dashv Z \dashv X$$

This is precisely a *ring oscillator*—the same circuit used in electronics to generate clock signals. In electronics, three NOT gates (inverters) connected in a loop produce oscillation because the output is always the complement of the input, and the signal chases itself around the loop.

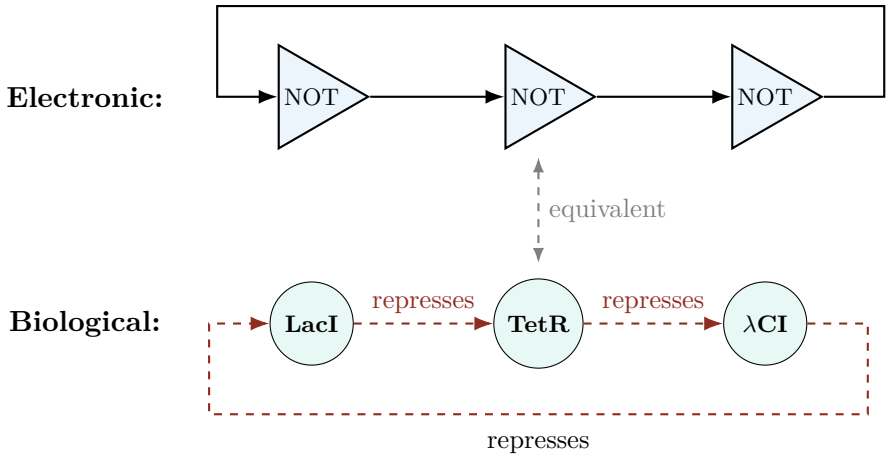


Figure 3.6: The repressilator is the biological equivalent of an electronic ring oscillator. Three repressor genes form a loop where each represses the next, producing sustained oscillations in protein expression levels.

Example 3.3 (Worked Example: Repressilator Dynamics)**.** Trace the logic step by step, starting with gene X highly expressed:

1. X is HIGH $\Rightarrow$ X protein represses Y $\Rightarrow$ Y goes LOW.
2. Y is LOW $\Rightarrow$ Y protein cannot repress Z $\Rightarrow$ Z goes HIGH.
3. Z is HIGH $\Rightarrow$ Z protein represses X $\Rightarrow$ X goes LOW.
4. X is LOW $\Rightarrow$ X protein cannot repress Y $\Rightarrow$ Y goes HIGH.
5. Y is HIGH $\Rightarrow$ Y protein represses Z $\Rightarrow$ Z goes LOW.
6. Z is LOW $\Rightarrow$ Z protein cannot repress X $\Rightarrow$ X goes HIGH.
7. Return to step 1. The cycle repeats indefinitely!

The period of oscillation depends on protein production and degradation rates. In the original Elowitz and Leibler [2000] experiment, the period was approximately 150 minutes per cycle, visible as blinking fluorescent *E. coli* cells under a microscope.

3.4.3 The Toggle Switch: A Biological Flip-Flop

The *genetic toggle switch* [Gardner et al., 2000] consists of two genes that mutually repress each other. This creates *bistability*: the system has two stable states and can be flipped between them by transient signals.

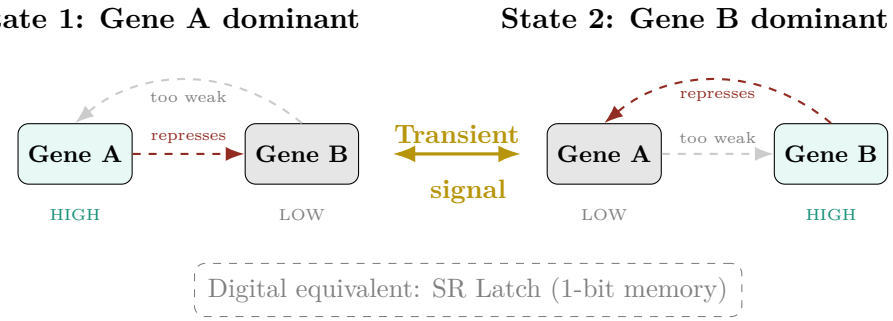


Figure 3.7: The toggle switch as a bistable flip-flop. In State 1, Gene A is dominant and represses Gene B. In State 2, Gene B is dominant and represses Gene A. A transient signal can flip between states, but each state is self-maintaining—a 1-bit biological memory.

The toggle switch truth table looks like a memory element:

Signal 1 (Set)	Signal 2 (Reset)	Output State
0	0	*Retains previous state*
1	0	Gene A dominant
0	1	Gene B dominant
1	1	Undefined (race condition)

This is identical to an SR latch in digital electronics—the simplest form of memory. Biology discovered flip-flops billions of years before engineers did.

3.4.4 GRNs as Programs

A gene regulatory network can be formalized as a dynamical system. Let $x_i(t) \geq 0$ denote the expression level (mRNA or protein concentration) of gene i at time t. The regulatory dynamics are:

$$\frac{dx_i}{dt} = f_i(x_1, x_2, \ldots, x_n) - \gamma_i x_i, \tag{3.2}$$

where f_i is the *regulatory function* (determined by the network topology and TF binding kinetics) and γ_i is the degradation rate. A common model for the regulatory function uses Hill kinetics:

$$f_i(\mathbf{x}) = \beta_i \cdot \prod_{j \in \text{activators}} \frac{x_j^{n_j}}{K_j^{n_j} + x_j^{n_j}} \cdot \prod_{k \in \text{repressors}} \frac{K_k^{n_k}}{K_k^{n_k} + x_k^{n_k}}, \tag{3.3}$$

where β_i is the maximal production rate, K_j are dissociation constants, and n_j are Hill coefficients controlling the steepness of the regulatory response.

Example 3.4 (Worked Example: Toggle Switch Steady States)**.** For the toggle switch, we have two genes x and y that mutually repress each other:

$$\frac{dx}{dt} = \frac{\beta}{1 + y^n} - \gamma x,$$
$$\frac{dy}{dt} = \frac{\beta}{1 + x^n} - \gamma y.$$

At steady state, $\frac{dx}{dt} = \frac{dy}{dt} = 0$. Let $\beta = 10$, $\gamma = 1$, and $n = 2$:

$$x = \frac{10}{1 + y^2}, \qquad y = \frac{10}{1 + x^2}.$$

Substituting the first into the second:

$$y = \frac{10}{1 + \left(\frac{10}{1+y^2}\right)^2}.$$

This equation has three solutions: (1) a symmetric unstable fixed point at $x = y \approx 2.62$ (found numerically), and (2,3) two stable states where one gene dominates: approximately $(x, y) \approx (9.6, 0.98)$ and $(x, y) \approx (0.98, 9.6)$. The two asymmetric fixed points are the "flip" and "flop" of the toggle switch.

From GRNs to DOGMA Regulation

DOGMA's Regulation stage computes an activation map $A_t = \text{Regulate}(\Gamma_t, c_t)$ that determines which genomic regions participate in the current computation. This is a direct formalization of Equation 3.2: the genome Γ_t plays the role of the gene network, the context c_t plays the role of external signals, and the activation map A_t plays the role of the expression profile $\mathbf{x}(t)$.
The key design insight borrowed from biology is that *regulation should be sparse and context-dependent*. In a human cell, only 10–20% of genes are expressed at any time. DOGMA inherits this sparsity: the activation map activates only the genomic regions relevant to the current task, rather than engaging the full genome uniformly.

3.5 Boolean Networks in Biology

In the 1960s, Stuart Kauffman proposed a radical simplification of gene regulatory networks: treat each gene as a *Boolean variable* (ON or OFF), and model regulation as a *Boolean function* of the gene's inputs. Despite the drastic simplification, Boolean network models capture essential features of real biological networks.

3.5.1 Kauffman's Random Boolean Networks

A *random Boolean network* (RBN) with N genes is defined as follows:

1. Each gene i has a state $s_i(t) \in \{0, 1\}$ at time t.
2. Each gene receives inputs from K randomly chosen other genes.
3. Each gene's next state is determined by a randomly assigned Boolean function of its K inputs.
4. All genes update synchronously: $s_i(t + 1) = f_i(s_{j_1}(t), \ldots, s_{j_K}(t))$.

The total state of the network at time t is a binary vector $\mathbf{s}(t) = (s_1(t), \ldots, s_N(t))$, which can take at most 2^N distinct values. Since the update rule is deterministic, the network must eventually revisit a state, entering a *cycle*—called an *attractor*.

3.5.2 Attractors as Cell Types

Kauffman's key insight was to identify attractors with cell types. Consider:

- A human has ∼20,000 genes and ∼200 cell types.
- A random Boolean network with N nodes typically has $\sim\sqrt{N}$ attractors.
- For $N = 20{,}000$: $\sqrt{20{,}000} \approx 141$—remarkably close to ∼200.

While this numerical coincidence is not proof, the conceptual framework is powerful: *cell differentiation corresponds to the network settling into different attractors*, and the number of stable cell types is determined by the network's dynamical structure, not by a special genetic program for each cell type.

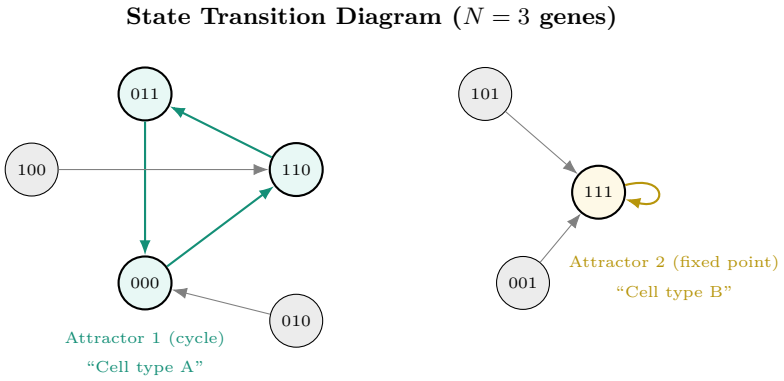


Figure 3.8: State transition diagram for a hypothetical 3-gene Boolean network. The $2^3 = 8$ possible states are connected by the network's update rule. States eventually reach *attractors*—either fixed points or cycles. Each attractor corresponds to a stable cell type. Transient states represent differentiating cells "flowing" toward their fate.

Boolean Network Attractors and DOGMA

In DOGMA, the genome state Γ_t can be viewed as a high-dimensional dynamical system whose attractors correspond to stable computational strategies. The Tera regime state nudges the system between attractor basins, analogous to external signals triggering cell fate transitions. The regulation stage determines the basin of attraction for each input context.

3.6 Signal Transduction as Computation

Cells do not just regulate individual genes in isolation—they process complex signals from their environment through elaborate *signal transduction pathways*. These pathways are biochemical circuits that convert extracellular signals into intracellular responses, and they implement sophisticated computational operations.

3.6.1 Kinase Cascades as Amplifiers

A *kinase* is an enzyme that adds a phosphate group to a target protein, typically activating it. In *kinase cascades*, the signal is passed through a chain of kinases, where each activated kinase activates many copies of the next kinase in the chain:

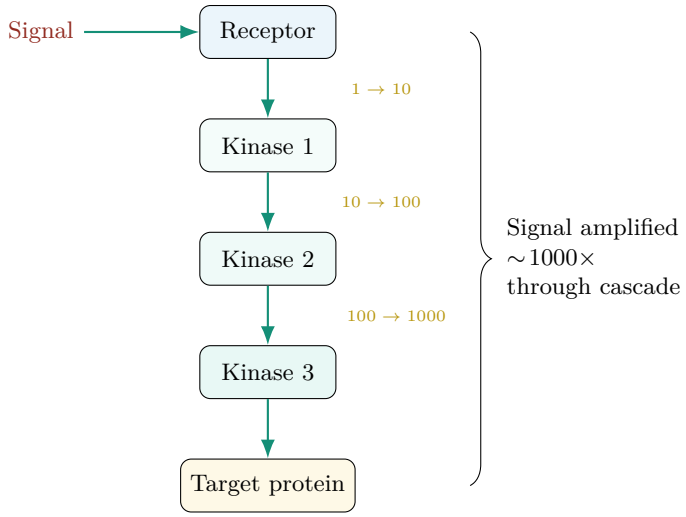


Figure 3.9: A kinase cascade amplifies a weak extracellular signal into a strong intracellular response. Each level activates many copies of the next kinase, producing exponential amplification—analogous to a transistor amplifier chain.

3.6.2 The MAPK Pathway: A Signal Processing Pipeline

The *mitogen-activated protein kinase* (MAPK) pathway is one of the most important and well-studied signal transduction cascades. It converts growth factor signals at the cell surface into gene expression changes in the nucleus, controlling cell growth, division, and differentiation.

The MAPK pathway is a *three-tier kinase cascade*: Raf → MEK → ERK. Each tier amplifies the signal and adds a layer of regulation. The pathway has striking parallels to a *signal processing pipeline*:

MAPK Layer	**Signal Processing Analogue**	**DOGMA Stage**
Growth factor receptor	Input transducer (antenna)	Context encoding
Ras (G-protein switch)	Binary switch / multiplexer	Regulation gate
Raf (MAP3K)	First amplifier stage	Activation map
MEK (MAP2K)	Second amplifier + filter	Transcription
ERK (MAPK)	Output driver	Expression
Transcription factors	Output device (speaker)	Folding / output

3.6.3 Feedback Loops in Signaling

Signal transduction pathways are not simple linear chains—they contain extensive feedback loops that shape the signal:

- **Negative feedback** attenuates the signal, preventing overactivation. Example: ERK phosphorylates and inactivates Sos (an upstream activator), creating a self-limiting circuit. *Computational analogue:* gain control / automatic volume adjustment.
- **Positive feedback** amplifies the signal and can create bistability (all-or-none responses). Example: ERK activates its own upstream kinase Raf through an indirect path. *Computational analogue:* switch / latch with hysteresis.
- **Combined feedback** creates complex behaviors like oscillations, adaptation (returning to baseline despite persistent stimulus), and ultrasensitivity (very steep dose-response curves).

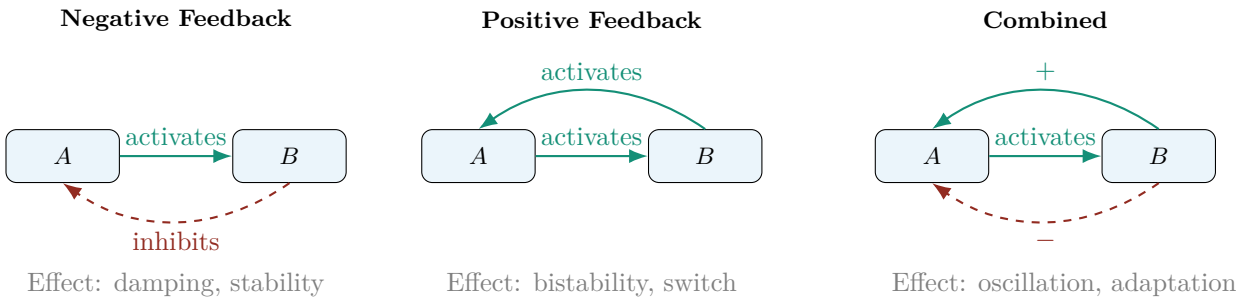


Figure 3.10: Three types of feedback in signal transduction. Negative feedback creates stability and damping. Positive feedback creates bistability and switching. Combined feedback can produce oscillations or adaptation.

Signal Transduction and DOGMA

DOGMA's six-stage pipeline (Regulation → Transcription → Translation → Folding → Expression → Tera feedback) mirrors the signal transduction paradigm. Each stage transforms and refines the signal, with feedback from downstream stages (especially Tera) modulating upstream behavior. The MAPK cascade's multi-tier amplification with feedback is a direct biological precedent for DOGMA's multi-stage processing with regime-driven adaptation.

3.7 Epigenetics: Computation That Modifies the Reader

Epigenetics refers to heritable changes in gene expression that do not alter the DNA sequence itself [Allis and Jenuwein, 2016]. If the genome is a book, epigenetics is the system of bookmarks, highlights, and sticky notes that tell the reader which pages to read and which to skip—without changing a single word of the text.

3.7.1 DNA Methylation

DNA methylation is the addition of a methyl group (CH_3) to the cytosine base in DNA, typically at *CpG dinucleotides* (a C followed by a G). Methylation generally *silences* gene expression through two mechanisms:

1. **Direct blocking:** The methyl group physically prevents transcription factors from binding to the DNA.

2. **Recruitment of repressors:** Methylated DNA attracts *methyl-binding proteins* that recruit histone-modifying enzymes, compacting the chromatin and making the DNA inaccessible.

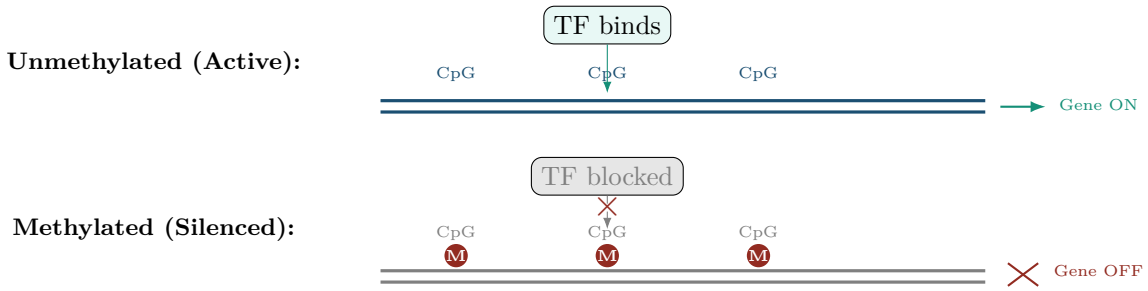


Figure 3.11: DNA methylation silences genes. Methyl groups (M) added to CpG sites block transcription factor binding, effectively switching the gene off without changing the DNA sequence.

3.7.2 Histone Modification

DNA in eukaryotic cells is wound around protein spools called *histones*. Chemical modifications to histone "tails" (protruding amino acid chains) alter chromatin structure:

- **Histone acetylation** (adding acetyl groups): *Opens* chromatin, making DNA accessible for transcription. Think of it as "loosening the spool."
- **Histone methylation** (adding methyl groups): Can either *open* or *close* chromatin, depending on which amino acid is modified and how many methyl groups are added.
- **Histone phosphorylation:** Involved in DNA damage response and chromosome condensation during cell division.

3.7.3 The Histone Code

The combination of histone modifications at a given genomic locus forms a *histone code*—a combinatorial signal that determines the regulatory state of that region [Allis and Jenuwein, 2016]. This code is "read" by effector proteins with specific binding domains.

Modification	Effect	Chromatin State	Computational Analogue
H3K4me3	Activation	Open	Feature flag = ON
H3K27me3	Repression	Closed	Feature flag = OFF
H3K9ac	Activation	Open	Access permission = READ
H3K9me3	Strong repression	Heterochromatin	Access permission = DENIED

3.7.4 Epigenetic Memory: State Without Sequence Change

The most remarkable property of epigenetic marks is that they are *maintained through cell division*. When a cell divides, its methylation pattern and histone modifications are copied to both daughter cells. This creates *persistent state without sequence modification*:

- A liver cell remains a liver cell not because its DNA is different from a neuron's, but because its epigenetic state keeps liver-specific genes active and neuron-specific genes silent.
- The same genome, with different epigenetic marks, produces completely different cell types—just as the same program, with different configuration files, produces different behaviors.

Epigenetics as Configuration State

From a computational perspective, epigenetics is a *configuration layer* that modifies the behavior of the underlying program (genome) without changing its source code. It is analogous to environment variables, runtime configuration files, or feature flags that alter program behavior at deployment time.
In DOGMA, the Tera regime state (Chapter 9) plays this role: it modifies how the genome is regulated and expressed without altering the genome itself. Just as epigenetic marks can persist across cell divisions, Tera's meta-policy memory persists across training runs.

3.8 Alternative Splicing: One Genome, Many Programs

3.8.1 The Mechanism

In eukaryotes, protein-coding genes are interrupted by non-coding sequences called *introns*. During transcription, introns are removed and the remaining *exons* are joined together in a process called *splicing*. Crucially, different combinations of exons can be joined, producing different mRNAs—and therefore different proteins—from the same gene.

This is *alternative splicing*: the ability to produce multiple distinct outputs from a single genetic locus, depending on context. The human gene *DSCAM* (Down Syndrome Cell Adhesion Molecule) in *Drosophila* can produce over 38,000 distinct mRNA variants through alternative splicing—more than twice the total number of genes in the fly's genome.

3.8.2 Computational Significance

Alternative splicing demonstrates that the relationship between genome and output is *many-to-many*, not one-to-one. A single genomic region can produce different outputs depending on regulatory context. DOGMA's Transcription stage implements this principle: the same genome Γ_t can yield different transcripts T_t depending on the activation map A_t and context c_t.

Alternative Splicing in DOGMA

In the EVO-TRAINER implementation, alternative splicing corresponds to the ability of the transcription engine to select different subsets of activated genomic regions and combine them in different orders. This is implemented through context-dependent region selection and composition operators, producing task-specific intermediate representations from a shared persistent genome.

3.9 The ENCODE Project: Quantifying the Regulatory Genome

The ENCODE (Encyclopedia of DNA Elements) project [ENCODE Project Consortium, 2012] systematically mapped functional elements across the human genome. Key findings relevant to DOGMA include:

- **80% of the genome is functional** (by biochemical assay), far more than the ∼1.5% that encodes proteins. Most functional sequence is regulatory.
- **Over 400,000 enhancer-like regions** were identified, outnumbering genes by 20:1. The regulatory program is far more complex than the structural program.
- **Gene expression is combinatorially regulated:** the average gene is influenced by multiple distal enhancers, and the average enhancer influences multiple genes.

The ENCODE findings reinforce a central DOGMA thesis: *in sophisticated information-processing systems, regulation is more important than content.* A genome's power lies not in its coding sequences alone, but in the regulatory architecture that controls their expression. Similarly, DOGMA's power lies not in its base representations alone, but in the regulation, transcription, and expression stages that control how those representations are activated and combined.

3.10 Synthetic Biology: Engineering Biological Circuits

Synthetic biology applies engineering principles to design and construct biological systems with novel functions. Two landmark achievements demonstrate that gene regulation can be *designed*, not just observed:

1. **The repressilator** [Elowitz and Leibler, 2000]: an engineered genetic oscillator consisting of three mutually repressing genes arranged in a ring (Section 3.4). The circuit produces regular oscillations in fluorescent protein expression, visible as blinking cells under a microscope.
2. **The genetic toggle switch** [Gardner et al., 2000]: an engineered bistable circuit consisting of two mutually repressing genes (Section 3.4). The switch can be flipped between two stable states by transient chemical signals, implementing a one-bit memory.

These achievements confirm that gene regulation is not merely an evolved accident but a *designable computational substrate*. If we can engineer biological circuits, we can certainly formalize their computational principles and implement them in silicon—which is precisely what DOGMA does.

3.11 From Biological Regulation to DOGMA

This chapter has established that gene regulation is a computational system with:

1. **Logic:** Transcription factors implement combinatorial Boolean functions (Section 3.3).
2. **Memory:** Epigenetic marks and toggle switches maintain persistent state (Sections 3.7, 3.4).
3. **Dynamics:** Feed-forward loops, oscillators, and feedback circuits implement temporal computation (Sections 3.4, 3.6).
4. **Context-dependence:** Alternative splicing and combinatorial regulation produce different outputs from the same genome (Section 3.8).
5. **Sparsity:** Only 10–20% of genes are active in any given cell, creating efficient task-specific programs.

6. **Hierarchy:** Regulation operates at multiple scales (nucleotide, gene, operon, chromosome, genome).

DOGMA inherits all six properties:

1. **Logic → Regulation stage:** The activation map A_t implements context-dependent gating.
2. **Memory → Tera meta-policy:** Persistent regime state survives across runs.
3. **Dynamics → Evolutionary training:** Multi-generational mutation and selection implement temporal optimization.
4. **Context-dependence → Transcription:** The same genome produces different transcripts for different tasks.
5. **Sparsity → Regulated activation:** Only relevant regions participate in each forward pass.
6. **Hierarchy → Genomic organization:** Bases → motifs → regions → subgenomes.

With Part I complete, we have established the biological foundations: DNA as information (Chapter 1), DNA as computation (Chapter 2), and gene regulation as a sophisticated programming system (this chapter). Part II builds the complementary AI/ML foundations before Part III unites them in the DOGMA architecture.

3.12 Exercises

3.1. **[Logic]** Design a gene regulatory circuit (using transcription factor interactions) that implements the Boolean function $f(A, B, C) = (A \wedge B) \vee (\neg C)$. Draw the circuit diagram and specify which interactions are activating vs. repressing.

3.2. **[Hill Function]** A gene is regulated by an activator with Hill coefficient $n = 4$ and $K = 50$ nM. The maximal expression rate is $\beta = 200$ mRNA/min and the degradation rate is $\gamma = 0.1$ min^{-1}.

(a) What is the steady-state mRNA level when the activator concentration is $x = 25$ nM?

(b) At what activator concentration does expression reach 90% of its maximum?

(c) Plot the dose-response curve for $x \in [0, 200]$ nM.

3.3. **[Dynamics]** For the toggle switch system with mutual repression:

$$\frac{dx}{dt} = \frac{\beta}{1 + y^n} - \gamma x, \qquad \frac{dy}{dt} = \frac{\beta}{1 + x^n} - \gamma y,$$

find the steady states for $\beta = 10$, $\gamma = 1$, $n = 2$. Show that the system is bistable by analyzing the nullclines.

3.4. **[Boolean Networks]** Construct a Boolean network with $N = 4$ genes, where each gene's next state depends on two inputs with the following rules:

- Gene 1: AND(Gene 2, Gene 4)
- Gene 2: OR(Gene 1, Gene 3)

- Gene 3: NOT(Gene 2)
- Gene 4: XOR(Gene 1, Gene 3)

Enumerate all $2^4 = 16$ states and draw the complete state transition diagram. How many attractors exist? What are their lengths?

3.5. **[Signal Transduction]** Consider a three-tier kinase cascade where each active kinase activates 10 copies of the next. If the initial signal activates 5 copies of Kinase 1:

(a) How many copies of Kinase 3 are active after one round of signaling?

(b) If each tier also has a phosphatase that inactivates kinases at a rate of 20% per step, what is the net amplification?

(c) Explain how negative feedback from Kinase 3 to Kinase 1 would change the dynamics.

3.6. **[Coding]** Implement a gene regulatory network simulator that:

(a) Takes a network topology (adjacency matrix with activation/repression signs) and Hill parameters.

(b) Simulates the dynamics using Equation 3.2 with Hill kinetics (Equation 3.3).

(c) Plots the time course of all gene expression levels.

(d) Test with the repressilator circuit: three genes in a mutual repression ring.

3.7. **[Analysis]** The ENCODE project found >400,000 enhancer-like regions but only ∼20,000 protein-coding genes. What does this 20:1 ratio of regulatory to coding elements suggest about the relative importance of "content" vs. "control" in biological information processing? How does this inform DOGMA's design philosophy?

3.8. **[Conceptual]** Compare epigenetic memory to the Tera regime state in DOGMA. What properties do they share? Where does the analogy break down?

3.9. **[Advanced]** Prove that a feed-forward loop with coherent type-1 logic (both direct and indirect paths are activating) acts as a persistence detector: it responds to sustained input signals but filters out transient pulses. Under what parameter conditions does filtering occur?

3.10. **[Synthesis]** Design a synthetic biological circuit that implements a 2-bit counter (counts 00 $\rightarrow$ 01 $\rightarrow$ 10 $\rightarrow$ 11 $\rightarrow$ 00). Specify the required number of genes, the regulatory interactions between them, and the external clock signal. How does this compare to the digital logic implementation?

3.13 Key Takeaways

Key Takeaways

- Gene regulation implements combinatorial logic through transcription factors, with promoters, enhancers, silencers, and insulators forming a complete regulatory grammar.
- Transcription factor binding follows Hill kinetics, producing sigmoidal (switch-like) responses that convert analog chemical signals into digital-like decisions. The Hill coefficient n controls steepness.
- The lac operon is a biological AND gate: expression requires lactose present AND glucose absent—an economically optimal computation.
- Gene regulatory networks exhibit recurring motifs (toggle switches, oscillators, feed-forward loops) that implement specific computational functions: memory, timing, and signal filtering.
- Boolean networks model gene regulation as discrete state machines, with attractors corresponding to stable cell types—providing a bridge between molecular biology and computation theory.
- Signal transduction pathways implement amplification (kinase cascades), signal processing (MAPK pipeline), and feedback control (positive and negative loops).
- Epigenetics provides a configuration layer that modifies gene expression without altering the DNA sequence—the biological precedent for DOGMA's Tera regime state.
- Alternative splicing demonstrates that one genome can produce many different outputs depending on context—the principle behind DOGMA's context-dependent transcription.
- The ENCODE project revealed that ∼80% of the functional genome is regulatory, confirming that control is more complex and important than content.
- Synthetic biology demonstrates that biological regulatory circuits can be designed from first principles, validating DOGMA's approach of formalizing and reimplementing them computationally.

Suggested Reading

- Jacob and Monod [1961]: The original discovery of gene regulation (lac operon).
- Alon [2019]: Systems biology of gene networks, with emphasis on motifs and design principles.
- ENCODE Project Consortium [2012]: The ENCODE project's comprehensive regulatory map.
- Allis and Jenuwein [2016]: Epigenetic mechanisms and the histone code.
- Elowitz and Leibler [2000] and Gardner et al. [2000]: Foundational synthetic biology circuits.

Part II

The Intelligence Stack

Chapter 4

Machine Learning Foundations for Biological AI

A learning machine is a machine that can change its behavior in response to experience.

— After Alan Turing, 1950

Parts I established that biology encodes, regulates, and evolves information through sophisticated computational mechanisms. This chapter begins Part II by building the machine learning foundations needed to translate those biological principles into working AI systems. We cover neural networks, optimization, representation learning, and evolutionary algorithms—focusing specifically on the concepts that reappear in DOGMA's architecture.

This is not a comprehensive ML textbook (we recommend Goodfellow et al. [2016] for that). Instead, it is a targeted introduction that equips the reader with the precise ML vocabulary and intuitions needed for the DOGMA chapters that follow.

4.1 The Learning Problem

4.1.1 Supervised Learning as Function Approximation

At its core, machine learning is about finding patterns in data. The simplest formulation is *supervised learning*: given a dataset of input-output pairs, find a function that maps inputs to outputs.

Supervised Learning

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ of input-output pairs drawn from some unknown distribution $p(\mathbf{x}, y)$, find a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ that minimizes expected loss:

$$\theta^* = \arg\min_{\theta} \; \mathbb{E}_{(\mathbf{x},y)\sim p}\big[\mathcal{L}(f_\theta(\mathbf{x}), y)\big], \tag{4.1}$$

where $\mathcal{L}$ is a loss function and θ are learnable parameters. In practice, we approximate the expectation with the empirical mean over $\mathcal{D}$.

There are three crucial choices in any supervised learning setup: (1) the *model class* (what

family of functions f_θ can we represent?), (2) the *loss function* (what does "good" mean?), and (3) the *optimizer* (how do we search for θ^*?). The rest of this chapter develops each of these in detail.

Worked example: DNA GC-content classification. Suppose we want to classify DNA sequences of length 10 into "GC-rich" ($\geq 60\%$ G/C content) or "GC-poor" ($< 60\%$). Our input is a sequence like `ATCGGCTAGC`, represented as a vector $\mathbf{x} \in \{0,1\}^{40}$ using one-hot encoding (4 bases $\times$ 10 positions), and our output is a binary label $y \in \{0,1\}$. We seek a function f_θ that predicts y from $\mathbf{x}$. A simple approach is logistic regression: $f_\theta(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$, where σ is the sigmoid function.

Let us trace through this concretely. For the sequence `ATCGGCTAGC`, the one-hot encoding assigns a 4-dimensional vector to each position: A$\to [1,0,0,0]$, T$\to [0,0,0,1]$, C$\to [0,1,0,0]$, G$\to [0,0,1,0]$. Concatenating all 10 positions yields a 40-dimensional binary vector. The GC content is $6/10 = 60\%$, so the label is $y = 1$ (GC-rich). If we initialize $\mathbf{w}$ so that the weights corresponding to G and C positions are positive ($+0.3$ each) and A and T positions are negative (-0.2 each), the pre-activation $z = \mathbf{w}^\top \mathbf{x} + b$ will be positive for GC-rich sequences, and $\sigma(z)$ will be close to 1. Training adjusts these weights to maximize classification accuracy on the training set.

4.1.2 Loss Functions

The choice of loss function $\mathcal{L}$ determines what "good prediction" means. The most common loss functions are:

Loss Function	Formula	Use Case
Mean Squared Error (MSE)	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	Regression (continuous outputs)
Binary Cross-Entropy	$-\frac{1}{N}\sum_i [y_i \log \hat{y}_i + (1 - y_i)\log(1-\hat{y}_i)]$	Binary classification
Categorical Cross-Entropy	$-\frac{1}{N}\sum_i \sum_c y_{ic} \log \hat{y}_{ic}$	Multi-class classification
Negative Log-Likelihood	$-\frac{1}{N}\sum_i \log p_\theta(y_i \mid \mathbf{x}_i)$	General probabilistic models

Why cross-entropy and not MSE for classification? For classification tasks, cross-entropy is preferred over MSE for two reasons. First, the gradient of cross-entropy with respect to the logits simplifies to $\hat{\mathbf{y}} - \mathbf{y}$ (as you will prove in Exercise 4.1), producing a gradient that is large when the prediction is wrong and small when it is right. MSE gradients, by contrast, can become very small near $\hat{y} = 0$ or $\hat{y} = 1$ due to the saturating sigmoid, slowing learning. Second, cross-entropy has an information-theoretic interpretation: it measures the extra bits needed to encode the true distribution using the predicted distribution, connecting machine learning to the information theory discussed in Chapter 1.

Worked example: cross-entropy for nucleotide prediction. Suppose our model predicts the next nucleotide in a DNA sequence. The true next base is G, encoded as $\mathbf{y} = [0,0,1,0]$ (for A, C, G, T). Our model predicts probabilities $\hat{\mathbf{y}} = [0.1, 0.2, 0.6, 0.1]$. The cross-entropy loss is:

$$\mathcal{L} = -(0 \cdot \log 0.1 + 0 \cdot \log 0.2 + 1 \cdot \log 0.6 + 0 \cdot \log 0.1) = -\log 0.6 \approx 0.511. \tag{4.2}$$

A perfect prediction of $\hat{y}_G = 1.0$ would give $\mathcal{L} = 0$. A uniform prediction of $\hat{y}_G = 0.25$ would give $\mathcal{L} = -\log 0.25 \approx 1.386$. The loss measures how "surprised" the model is by the true answer.

4.1.3 The Bias-Variance Tradeoff

A fundamental tension in machine learning is between models that are too simple (high bias) and too complex (high variance).

The Bias-Variance Decomposition

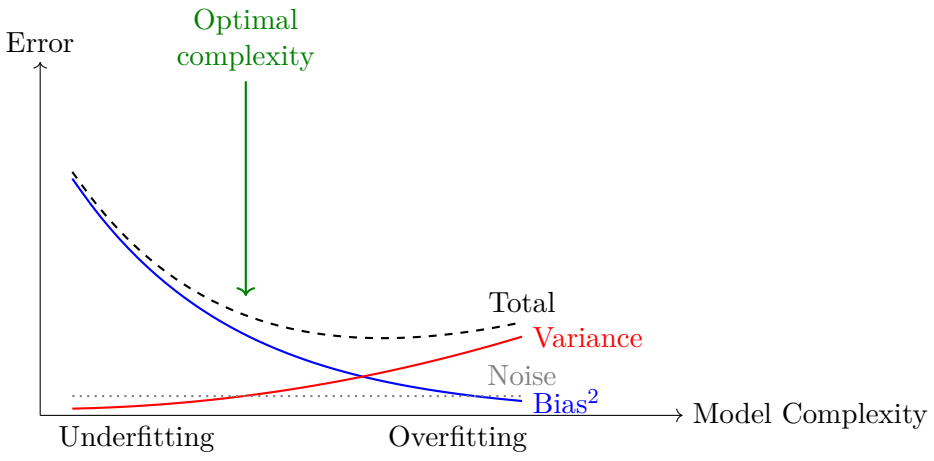
The expected error of a model can be decomposed as:

$$\text{Error} = \underbrace{\text{Bias}^2}_{\substack{\text{systematic error from} \\ \text{model assumptions}}} + \underbrace{\text{Variance}}_{\substack{\text{sensitivity to} \\ \text{training data}}} + \underbrace{\text{Irreducible noise}}_{\substack{\text{inherent randomness} \\ \text{in the data}}}. \tag{4.3}$$

Intuition. Imagine fitting a curve to noisy data:

- A straight line (linear model) may miss the true pattern (high bias, low variance).
- A degree-20 polynomial may fit every noise point (low bias, high variance).
- A degree-3 polynomial may capture the true pattern without overfitting (good tradeoff).

Why the tradeoff matters quantitatively. Consider a concrete scenario. Suppose the true function is $f(x) = \sin(x)$ and we observe data with Gaussian noise $\epsilon \sim \mathcal{N}(0, 0.1^2)$. A linear model $\hat{f}(x) = ax + b$ has high bias (it cannot represent the curvature of sine) but low variance (the estimated line changes little between different training samples). A degree-15 polynomial has low bias (it can approximate $\sin(x)$ well) but high variance: with only 10 training points, the polynomial wiggles wildly between data points, and different training sets produce drastically different fits. A degree-3 polynomial $\hat{f}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ provides a good tradeoff: the Taylor expansion of $\sin(x)$ is $x - x^3/6 + \cdots$, so degree 3 captures the dominant pattern while having few enough parameters to be stably estimated.



Biological Priors Reduce Variance

DOGMA's structured architecture embeds biological priors—typed bases, region structure, regulatory logic—that constrain the hypothesis space. This is analogous to choosing a degree-3 polynomial when you know the underlying process is smooth: the prior reduces variance without excessive bias, because the structure matches the true data-generating process. In contrast, a flat transformer must discover all structure from data alone, requiring more data to achieve the same generalization.

4.1.4 Unsupervised and Self-Supervised Learning

Not all learning requires labeled data. In *unsupervised learning*, the model discovers structure in unlabeled data (clustering, dimensionality reduction). In *self-supervised learning*, the model creates its own supervisory signal from the data structure.

The dominant self-supervised paradigm for language models is *next-token prediction*:

$$\mathcal{L}_{\text{LM}}(\theta) = -\sum_{t=1}^{T} \log p_\theta(x_t \mid x_1, \ldots, x_{t-1}). \tag{4.4}$$

This objective requires no human-labeled data—the text itself provides the supervision. The same principle applies to genomic sequences: predicting the next nucleotide given preceding context is a natural self-supervised objective for DNA language models (Chapter 6).

Why self-supervised learning is so powerful. The key advantage is *data efficiency at the labeling stage*. Consider genomic data: we have billions of base pairs of sequenced DNA, but only a tiny fraction has been functionally annotated. Self-supervised pre-training on raw sequence data learns representations that capture sequence grammar, motif structure, and evolutionary constraints—all without a single label. These pre-trained representations can then be fine-tuned with a small number of labeled examples for specific tasks like promoter prediction or variant effect classification. This *pre-train then fine-tune* paradigm is the foundation of modern biological AI.

Other self-supervised objectives.

- **Masked language modeling (MLM):** Randomly mask tokens and predict them from surrounding context. Used in BERT [Devlin et al., 2019] and DNABERT [Ji et al., 2021b]. For example, given "ATG`[MASK]`GC", the model must predict that the masked token is most likely C or T based on context.

- **Contrastive learning:** Learn embeddings where similar items are close and dissimilar items are far. Used in protein representation learning [Lin et al., 2023]. The objective pulls together embeddings of augmented views of the same protein while pushing apart embeddings of different proteins.

- **Denoising:** Corrupt input and train the model to reconstruct the original. Used in denoising autoencoders and diffusion models. For DNA, corruption might involve random base substitutions, and the model learns to "correct" them.

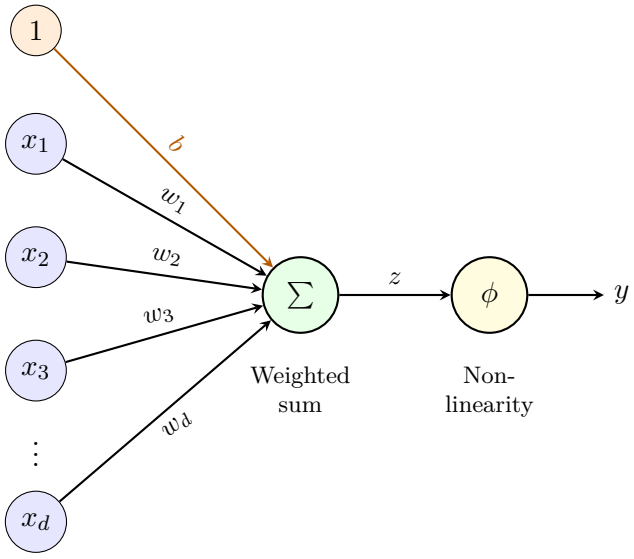
4.2 Neural Networks from First Principles

4.2.1 The Perceptron: A Single Neuron

The simplest neural network is a single neuron (perceptron), which computes a weighted sum of its inputs, adds a bias, and applies a nonlinear activation function:

$$y = \phi(\mathbf{w}^\top \mathbf{x} + b) = \phi\left(\sum_{i=1}^{d} w_i x_i + b\right). \tag{4.5}$$

The weights $\mathbf{w}$ determine how much each input contributes to the output, and the bias b shifts the decision boundary. Together, $\mathbf{w}$ and b define a hyperplane in d-dimensional space: the neuron outputs different values on each side of this hyperplane.



Worked example. Consider a neuron with 3 inputs ($d = 3$), weights $\mathbf{w} = [0.5, -0.3, 0.8]$, bias $b = 0.1$, and ReLU activation $\phi(z) = \max(0, z)$. For input $\mathbf{x} = [1.0, 2.0, 0.5]$:

$$z = 0.5 \times 1.0 + (-0.3) \times 2.0 + 0.8 \times 0.5 + 0.1 \tag{4.6}$$

$$= 0.5 - 0.6 + 0.4 + 0.1 = 0.4, \tag{4.7}$$

$$y = \max(0, 0.4) = 0.4. \tag{4.8}$$

If instead the input were $\mathbf{x} = [0.1, 2.0, 0.5]$, we would get $z = 0.05 - 0.6 + 0.4 + 0.1 = -0.05$, and $y = \max(0, -0.05) = 0$. The neuron is "off" for this input—ReLU has completely silenced it. This sparsity (many neurons outputting exactly zero) is a key property of ReLU networks and contributes to computational efficiency.

A single perceptron can only represent *linear decision boundaries*. It can compute AND and OR but *cannot* compute XOR—this limitation motivated the development of multi-layer networks.

x_1	x_2	AND	OR	XOR	NAND
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	1	0	0

AND and OR are linearly separable (a single line separates the 0s from the 1s), but XOR is not. This is the historical motivation for *deep* networks.

4.2.2 Activation Functions

The activation function ϕ introduces nonlinearity, enabling the network to learn complex patterns. Without nonlinearity, a stack of linear layers would collapse into a single linear transformation. To see why, note that if $f_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x}+\mathbf{b}_1$ and $f_2(\mathbf{x}) = \mathbf{W}_2\mathbf{x}+\mathbf{b}_2$, then $f_2(f_1(\mathbf{x})) = \mathbf{W}_2(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)+\mathbf{b}_2 = (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2)$, which is just another linear function. No matter how many linear layers you stack, the result is equivalent to a single linear layer.

Function	**Formula**	**Range**	**Properties**
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	$(0, 1)$	Smooth, saturates at extremes; vanishing gradient problem
Tanh	$\tanh(z) = \frac{e^z-e^{-z}}{e^z+e^{-z}}$	$(-1, 1)$	Zero-centered; still saturates
ReLU	$\max(0, z)$	$[0, \infty)$	Fast, sparse; "dead neuron" problem
Leaky ReLU	$\max(\alpha z, z)$, $\alpha \ll 1$	$(-\infty, \infty)$	Fixes dead neurons
GELU	$z \cdot \Phi(z)$	$\approx (-0.17, \infty)$	Smooth ReLU; used in transformers
Swish	$z \cdot \sigma(z)$	$\approx (-0.28, \infty)$	Self-gated; strong empirical performance

The vanishing gradient problem in detail. Sigmoid and tanh saturate for large $|z|$: when $z = 10$, $\sigma'(10) \approx 0.0000454$. During backpropagation, gradients are multiplied through layers, so if each layer contributes a factor near zero, the gradient shrinks exponentially. For a network with L layers, the gradient at the first layer is proportional to $\prod_{\ell=1}^{L} \sigma'(z^{(\ell)})$. If each factor is 0.25 (the maximum of σ'), then after just 10 layers: $0.25^{10} \approx 9.5 \times 10^{-7}$. This makes training deep networks with sigmoid activations extremely difficult.

ReLU solves this by having a derivative of exactly 1 for $z > 0$, allowing gradients to flow unchanged through active neurons. The cost is the "dead neuron" problem: if a neuron's pre-activation z is always negative for all training inputs, the gradient is always zero and the neuron can never recover. Leaky ReLU mitigates this by using a small positive slope α (typically 0.01) for negative inputs.

Activation Functions and Biological Neurons

Biological neurons fire when their membrane potential exceeds a threshold—a process well approximated by the sigmoid function. However, real neurons exhibit much richer dynamics: refractory periods, spike-rate adaptation, bursting patterns. DOGMA's regulatory activation weights ρ_i generalize simple thresholding to context-dependent, multi-factor activation, more closely mirroring biological reality.

4.2.3 Multi-Layer Perceptrons (MLPs)

A *multi-layer perceptron* (MLP) composes multiple layers of neurons:

$$\mathbf{h}^{(\ell)} = \phi^{(\ell)}(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}), \quad \ell = 1, \ldots, L. \tag{4.9}$$

Here, $\mathbf{h}^{(0)} = \mathbf{x}$ is the input, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix connecting layer $\ell - 1$ to layer ℓ, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$ is the bias vector, and d_ℓ is the width (number of neurons) of layer ℓ. The total number of learnable parameters is $\sum_{\ell=1}^{L}(d_\ell \times d_{\ell-1} + d_\ell)$.

Worked example: 2-layer MLP for XOR. We solve XOR using a network with 2 inputs, a hidden layer of 2 neurons (ReLU activation), and 1 output neuron (sigmoid activation):

$$\text{Hidden layer:} \quad \mathbf{W}^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{4.10}$$

$$\text{Output layer:} \quad \mathbf{w}^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b^{(2)} = 0 \tag{4.11}$$

Let us verify all four cases:

x_1	x_2	$h_1 = \text{ReLU}(x_1 + x_2)$	$h_2 = \text{ReLU}(x_1 + x_2 - 1)$	$z = h_1 - 2h_2$	$\hat{y} = \sigma(z)$
0	0	0	0	0	0.50
0	1	1	0	1	0.73
1	0	1	0	1	0.73
1	1	2	1	0	0.50

The network outputs high values (0.73) for XOR=1 cases and 0.5 for XOR=0 cases. With further weight tuning (e.g., scaling $\mathbf{w}^{(2)}$ by a factor of 10), the separation becomes sharper. The key insight: the hidden layer creates a *new representation* in which the XOR problem becomes linearly separable. Neuron h_1 detects "at least one input is 1" and neuron h_2 detects "both inputs are 1"; the output layer then computes "h_1 but not h_2."

Theorem 4.1 (Universal Approximation [Goodfellow et al., 2016])**.** A feedforward network with a single hidden layer of sufficient width can approximate any continuous function on a compact domain to arbitrary accuracy.

The theorem guarantees *existence* but says nothing about *efficiency*. Deep networks achieve the same approximation quality with exponentially fewer parameters than shallow ones for many function classes—this is why depth matters.

Why depth helps: a counting argument. A function that requires 2^n neurons in a single hidden layer may require only $O(n)$ neurons distributed across $O(n)$ layers. For example, computing the parity of n binary inputs requires exponentially many neurons in one layer but only $n - 1$ neurons across $\log_2 n$ layers of XOR gates.

4.3 Training Neural Networks

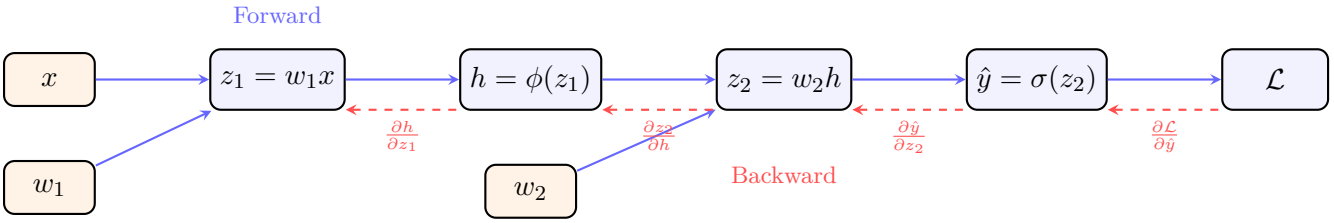
4.3.1 Backpropagation: Computing Gradients Efficiently

Parameters are optimized via gradient descent, and the gradient of the loss with respect to each parameter is computed efficiently by *backpropagation*—application of the chain rule through the computational graph.

The chain rule in action. Consider a simple 2-layer network: $z_1 = w_1 x$, $h = \phi(z_1)$, $z_2 = w_2 h$, $\hat{y} = \sigma(z_2)$, $\mathcal{L} = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$. The gradient with respect to w_1 is:

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} \cdot \frac{\partial z_2}{\partial h} \cdot \frac{\partial h}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}. \tag{4.12}$$

Each factor in this chain is a simple local derivative. Backpropagation computes all gradients in a single backward pass through the network, with computational cost proportional to the forward pass.



Step-by-step backpropagation example. Let $x = 2$, $w_1 = 0.5$, $w_2 = 1.0$, $y = 1$ (true label), and $\phi = \text{ReLU}$:

1. **Forward pass:** $z_1 = 0.5 \times 2 = 1.0$; $h = \max(0, 1.0) = 1.0$; $z_2 = 1.0 \times 1.0 = 1.0$; $\hat{y} = \sigma(1.0) = 0.731$.
2. **Loss:** $\mathcal{L} = -\log(0.731) = 0.313$.
3. **Backward pass:**
   - $\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} = -\frac{1}{0.731} = -1.368$
   - $\frac{\partial \hat{y}}{\partial z_2} = \hat{y}(1-\hat{y}) = 0.731 \times 0.269 = 0.197$
   - $\frac{\partial \mathcal{L}}{\partial z_2} = -1.368 \times 0.197 = -0.269$ (this is $\hat{y} - y = 0.731 - 1$)
   - $\frac{\partial z_2}{\partial w_2} = h = 1.0$, so $\frac{\partial \mathcal{L}}{\partial w_2} = -0.269 \times 1.0 = -0.269$
   - $\frac{\partial z_2}{\partial h} = w_2 = 1.0$, $\frac{\partial h}{\partial z_1} = 1$ (ReLU derivative for $z_1 > 0$)
   - $\frac{\partial \mathcal{L}}{\partial w_1} = -0.269 \times 1.0 \times 1 \times x = -0.269 \times 2 = -0.538$
4. **Update:** With learning rate $\eta = 0.1$: $w_1 \leftarrow 0.5 - 0.1 \times (-0.538) = 0.554$; $w_2 \leftarrow 1.0 - 0.1 \times (-0.269) = 1.027$.

Both weights increased, which will push $\hat{y}$ closer to $y = 1$ on the next forward pass. Let us verify: with the updated weights, the forward pass gives $z_1 = 0.554 \times 2 = 1.108$, $h = 1.108$, $z_2 = 1.027 \times 1.108 = 1.138$, $\hat{y} = \sigma(1.138) = 0.757$. Indeed, $\hat{y}$ has moved from 0.731 to 0.757, closer to the target $y = 1$, and the loss decreased from 0.313 to $-\log(0.757) = 0.278$.

4.3.2 Gradient Descent Variants

The basic gradient descent update rule is:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta), \tag{4.13}$$

where η is the learning rate. In practice, several variants improve convergence:

Optimizer	**Key Idea**	**Update Rule (simplified)**
SGD	Random mini-batch gradient estimates	$\theta \leftarrow \theta - \eta \hat{\nabla} \mathcal{L}$
SGD + Momentum	Accumulate past gradients to smooth updates	$\mathbf{v} \leftarrow \beta \mathbf{v} + \hat{\nabla} \mathcal{L}$; $\theta \leftarrow \theta - \eta \mathbf{v}$
AdaGrad	Per-parameter adaptive learning rates	$\theta_i \leftarrow \theta_i - \frac{\eta}{\sqrt{G_i + \epsilon}} \hat{\nabla}_i$
Adam	Adaptive moments: combines momentum + AdaGrad	$\theta \leftarrow \theta - \eta \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$

Worked example: gradient descent on a simple function. To build intuition, consider minimizing $f(\theta) = (\theta - 3)^2$ starting from $\theta_0 = 0$ with learning rate $\eta = 0.2$. The gradient is $f'(\theta) = 2(\theta - 3)$.

Step t	θ_t	$f(\theta_t)$	$f'(\theta_t)$	$\theta_{t+1} = \theta_t - 0.2 \cdot f'(\theta_t)$
0	0.000	9.000	-6.000	1.200
1	1.200	3.240	-3.600	1.920
2	1.920	1.166	-2.160	2.352
3	2.352	0.420	-1.296	2.611
4	2.611	0.151	-0.778	2.767
5	2.767	0.054	-0.467	2.860

The parameter θ converges toward the minimum at $\theta^* = 3$. Notice that each step reduces the loss by a factor of approximately $0.36 = (1 - 2\eta)^2 = (1 - 0.4)^2$, which is a geometric convergence rate characteristic of gradient descent on quadratic functions. If we set $\eta = 0.5$, each step would be $\theta_{t+1} = \theta_t - 2(\theta_t - 3) = 6 - \theta_t$, causing the parameter to oscillate: $0 \to 6 \to 0 \to 6 \to \cdots$. If $\eta > 0.5$, the oscillations diverge. This illustrates why the learning rate must be chosen carefully.

Adam [Kingma and Ba, 2015] is the most widely used optimizer in modern deep learning. It maintains exponential moving averages of the gradient (first moment $\hat{m}$) and the squared gradient (second moment $\hat{v}$), providing adaptive learning rates for each parameter. The full update rule is:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{4.14}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{4.15}$$

$$\hat{m}_t = m_t / (1 - \beta_1^t), \quad \hat{v}_t = v_t / (1 - \beta_2^t), \tag{4.16}$$

$$\theta_{t+1} = \theta_t - \eta\, \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon), \tag{4.17}$$

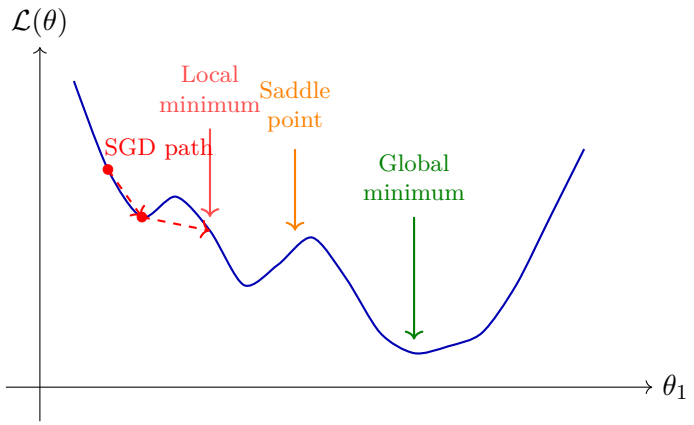
where $g_t = \nabla_\theta \mathcal{L}(\theta_t)$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ are the standard defaults. The bias correction terms ($\hat{m}_t$ and $\hat{v}_t$) are essential in early training when the moving averages are biased toward zero due to initialization at $m_0 = v_0 = 0$. Most DOGMA foundation model training uses Adam with learning rate warmup and cosine decay.

Implementation Note

When training DOGMA models, use Adam with $\eta = 3 \times 10^{-4}$, linear warmup over the first 2000 steps, and cosine decay to $\eta_{\min} = 10^{-5}$. The warmup prevents early instabilities when the randomly initialized model produces large gradients, and cosine decay allows fine-grained convergence in later training.

4.3.3 The Loss Landscape

Understanding optimization in neural networks requires thinking about the *loss landscape*—the surface defined by the loss function over the parameter space. For a network with d parameters, this is a surface in $(d+1)$-dimensional space, but we can visualize 2D slices.



Key features of neural network loss landscapes:

- **Local minima** are points where the loss is lower than all nearby points but not the global minimum. In high-dimensional spaces, most local minima have loss values close to the global minimum [Goodfellow et al., 2016].
- **Saddle points** are far more common than local minima in high dimensions. At a saddle point, the gradient is zero, but the Hessian has both positive and negative eigenvalues (loss increases in some directions and decreases in others).
- **Plateaus** are flat regions where the gradient is near zero, causing slow learning. Momentum helps escape plateaus by accumulating velocity from previous gradients.

4.3.4 Regularization: Preventing Overfitting

A model that memorizes the training data (overfitting) fails to generalize. Regularization techniques constrain the model to prefer simpler solutions:

1. **L2 regularization (weight decay):** Add $\lambda\|\theta\|^2$ to the loss, penalizing large weights:

$$\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{data}} + \lambda \sum_i \theta_i^2. \tag{4.18}$$

   This is equivalent to a Gaussian prior on the weights in a Bayesian framework. Concretely, $\lambda = 0.01$ means we are saying "we believe the weights should be small, with a prior variance of $1/(2\lambda) = 50$." Larger λ imposes a stronger prior, yielding a simpler model.

2. **Dropout** [Srivastava et al., 2014]: During training, randomly set each neuron's output to zero with probability p (typically $p = 0.1$ to 0.5). This forces the network to learn redundant representations, preventing co-adaptation of neurons. At test time, all neurons are active but outputs are scaled by $(1-p)$ to compensate. Dropout can be interpreted as training an ensemble of 2^N sub-networks (where N is the number of neurons) that share parameters.

3. **Early stopping:** Monitor performance on a held-out validation set and stop training when validation loss begins to increase, even if training loss continues to decrease. The gap between training loss (decreasing) and validation loss (increasing) is a signature of overfitting.

4. **Data augmentation:** Artificially increase the training set by applying transformations. For DNA sequences, augmentations include reverse complementation, random mutation, and subsequence extraction.

5. **Layer normalization** [Ba et al., 2016]: Normalize activations within each layer to have zero mean and unit variance, stabilizing the distribution of hidden representations. For a vector $\mathbf{h}$ of activations: $\text{LayerNorm}(\mathbf{h}) = \gamma \cdot \frac{\mathbf{h}-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta$, where μ and σ^2 are the mean and variance of $\mathbf{h}$, and γ, β are learned scale and shift parameters.

Gradient Descent vs. Evolution

Gradient descent optimizes parameters along the loss landscape using local gradient information. Evolution optimizes genomes through population-based search using fitness evaluation. The table below highlights key differences:

Property	Gradient Descent	Evolution
Search signal	Gradient (local)	Fitness (global)
Update rule	Continuous, small steps	Discrete mutations
Parallelism	Mini-batch	Population
Exploration	Limited (local minima)	Broad (mutation diversity)
Requires	Differentiable loss	Only evaluable fitness
State	Parameters ($\mathbb{R}^d$)	Genomes (structured)
Memory	None (memoryless)	Population history

DOGMA uses both: gradient descent for training neural components, and evolutionary search for optimizing prompt genomes and architectural structure (Chapter 12).

4.4 Convolutional Neural Networks

Before transformers dominated, *convolutional neural networks* (CNNs) were the architecture of choice for structured data. CNNs remain important in biological AI for processing DNA sequences and protein structures.

4.4.1 The Convolution Operation

A 1D convolution slides a learned filter (kernel) $\mathbf{k} \in \mathbb{R}^w$ across an input sequence $\mathbf{x} \in \mathbb{R}^T$ to produce a feature map:

$$(\mathbf{x} * \mathbf{k})_t = \sum_{i=0}^{w-1} k_i \cdot x_{t+i}. \tag{4.19}$$

The convolution has two important properties that make it efficient and effective:

- **Parameter sharing:** The same filter weights are used at every position. A filter with width w has only w parameters regardless of sequence length T, compared to $T \times T$ for a fully connected layer.
- **Translation equivariance:** If the input shifts by k positions, the output feature map also shifts by k positions. A motif detector works identically regardless of where the motif appears in the sequence.

Worked example: motif detection in DNA. Suppose we have a DNA sequence encoded numerically: $\mathbf{x} = [1, 0, 0, 1, 1, 0, 1, 0]$ (where 1 = purine, 0 = pyrimidine). A filter $\mathbf{k} = [1, 1, 1]$ (width 3) detects runs of three purines:

$$(\mathbf{x} * \mathbf{k})_1 = 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 = 1 \tag{4.20}$$
$$(\mathbf{x} * \mathbf{k})_2 = 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 = 1 \tag{4.21}$$
$$(\mathbf{x} * \mathbf{k})_3 = 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 2 \tag{4.22}$$
$$(\mathbf{x} * \mathbf{k})_4 = 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 = 2 \tag{4.23}$$
$$(\mathbf{x} * \mathbf{k})_5 = 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 = 2 \tag{4.24}$$
$$(\mathbf{x} * \mathbf{k})_6 = 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 = 1 \tag{4.25}$$

The highest activations (value 2) occur at positions 3–5, where purine density is highest. In practice, DNA models like DeepSEA [Zhou and Troyanskaya, 2015] use hundreds of learned filters to detect regulatory motifs.

4.4.2 Key CNN Concepts

- **Stride:** The step size between filter applications. Stride > 1 reduces output length (downsampling). With stride s, an input of length T with filter width w produces an output of length $\lfloor (T - w)/s \rfloor + 1$.
- **Padding:** Adding zeros around the input to control output size. "Same" padding preserves input length; "valid" padding (no padding) reduces length by $w - 1$.
- **Pooling:** Aggregating neighboring activations (max pooling or average pooling) to reduce dimensionality and provide translation invariance. Max pooling with window 2 halves the sequence length while keeping the strongest activations.

- **Multiple channels:** Each input can have multiple channels (e.g., 4 channels for one-hot encoded DNA), and each filter produces one output channel. A CNN layer with C_{in} input channels and C_{out} filters has $C_{\text{out}} \times C_{\text{in}} \times w$ learnable weights.
- **Dilated convolutions:** Inserting gaps between filter elements allows a small filter to cover a large receptive field. A filter with dilation d and width w has an effective receptive field of $w + (w-1)(d-1)$ positions. This is useful for capturing long-range patterns in DNA without increasing parameters.

Receptive field growth. Stacking L convolutional layers with filter width w gives a total receptive field of $L(w-1)+1$. With dilated convolutions using dilation rates $1, 2, 4, \ldots, 2^{L-1}$, the receptive field grows exponentially as $2^L - 1 + w - 1$. For DNA, where regulatory elements can influence expression from thousands of bases away, this exponential growth is essential.

CNNs in DOGMA

DOGMA's strand displacement computational path (Chapter 7) is conceptually related to 1D convolution: it slides local operations along the genomic sequence, detecting and transforming local patterns. However, DOGMA adds *typed* convolution: the operation depends on the base types being processed, analogous to how different enzymes recognize different DNA motifs.

4.5 Representation Learning

4.5.1 Embeddings: Mapping Symbols to Geometry

The core idea of representation learning is to map discrete symbols into continuous vector spaces where geometric relationships capture semantic relationships. An *embedding* is a learned function $e : \Sigma \to \mathbb{R}^d$ that assigns each symbol a dense vector.

For a vocabulary V of size $|V|$, an embedding layer is a matrix $\mathbf{E} \in \mathbb{R}^{|V| \times d}$ where row i is the embedding of token i. The embedding is learned end-to-end through backpropagation.

Why embeddings rather than one-hot encoding? A one-hot vector for a vocabulary of size $|V|$ is $|V|$-dimensional with a single 1 and all other entries 0. This representation has several problems: (1) it is very high-dimensional for large vocabularies ($|V| = 64$ for DNA 3-mers, $|V| \approx 50{,}000$ for text), (2) all pairs of symbols are equidistant (cosine similarity is 0 between any two different one-hot vectors), so the representation encodes no similarity structure, and (3) it is extremely sparse. Embeddings solve all three problems: they are low-dimensional ($d \ll |V|$), they capture similarity (similar symbols have similar embeddings), and they are dense.

Worked example: DNA k-mer embeddings. Consider a vocabulary of all DNA 3-mers (64 possible): ATG, AAA, CCC, etc. An embedding dimension of $d = 16$ gives an embedding matrix $\mathbf{E} \in \mathbb{R}^{64 \times 16}$. After training on genomic data, we might find:

- Start codons (ATG) and other coding-related 3-mers cluster together.
- GC-rich 3-mers (GCG, CGC, GGC) cluster separately from AT-rich 3-mers (AAT, TAT, ATA).
- Reverse complement pairs (e.g., ATG and CAT) have related but not identical embeddings.

The embedding captures both chemical properties (GC content) and functional properties (coding vs. regulatory) without being explicitly told about either.

Worked example: cosine similarity computation. Suppose after training, the embeddings (simplified to $d = 4$) are: ATG $= [0.8, 0.3, -0.1, 0.5]$ and GTG $= [0.7, 0.4, -0.2, 0.6]$. The cosine similarity is:

$$\mathbf{u} \cdot \mathbf{v} = 0.8 \times 0.7 + 0.3 \times 0.4 + (-0.1)(-0.2) + 0.5 \times 0.6 = 0.56 + 0.12 + 0.02 + 0.30 = 1.00 \tag{4.26}$$

$$\|\mathbf{u}\| = \sqrt{0.64 + 0.09 + 0.01 + 0.25} = \sqrt{0.99} \approx 0.995 \tag{4.27}$$

$$\|\mathbf{v}\| = \sqrt{0.49 + 0.16 + 0.04 + 0.36} = \sqrt{1.05} \approx 1.025 \tag{4.28}$$

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \frac{1.00}{0.995 \times 1.025} \approx 0.980. \tag{4.29}$$

The high similarity (0.98) reflects that ATG and GTG are functionally related—both are alternative start codons in some organisms.

Embeddings in DOGMA

DOGMA's genomic bases carry embeddings as their content component σ_i, but they also carry *type* (τ_i), *regulatory weight* (ρ_i), and *compatibility state* (ω_i). Formally, each base is a tuple $b_i = (\sigma_i, \tau_i, \rho_i, \omega_i)$. This is richer than a standard token embedding because each base is not merely a point in semantic space—it is a typed, weighted, interaction-annotated computational atom.

4.5.2 The Geometry of Meaning

Good embeddings exhibit meaningful geometric structure. Classic examples include Word2Vec's vector arithmetic: king $-$ man $+$ woman $\approx$ queen.

For biological sequences, protein language models learn embeddings where proteins with similar function cluster together, even without explicit functional labels [Lin et al., 2023]. DNA language models learn embeddings where regulatory elements cluster by function, coding regions by gene family, and promoters by expression level [Ji et al., 2021b].

Measuring embedding quality. Common metrics include:

- **Cosine similarity:** $\text{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$. Values near 1 indicate similar vectors.
- **k-nearest neighbors accuracy:** What fraction of an embedding's k nearest neighbors share its class label?
- **Linear probing:** Train a linear classifier on frozen embeddings. High accuracy indicates the embeddings capture relevant structure. For example, if a linear probe on DNABERT embeddings can predict whether a sequence is a promoter with 85% accuracy, the embeddings have learned promoter-relevant features even though the model was never trained on promoter labels.

4.5.3 Positional Encoding

Embeddings alone do not capture *position*—"ATG at position 1" and "ATG at position 100" have identical embeddings. Positional encoding injects position information:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad \text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right). \tag{4.30}$$

These sinusoidal functions create unique position-dependent patterns. The frequency decreases with dimension index i, so the first dimensions encode fine-grained position and later dimensions encode coarser position.

Why sinusoidal? The sinusoidal encoding has two useful properties. First, each position gets a unique encoding (no two positions have the same pattern across all dimensions). Second, the encoding supports relative position computation: for any fixed offset k, $\text{PE}(pos+k)$ can be expressed as a linear function of $\text{PE}(pos)$, allowing the model to learn attention patterns that depend on relative position. An alternative is *learned* positional embeddings, which train a separate embedding vector for each position. Learned embeddings are more flexible but cannot extrapolate to positions longer than seen during training.

Positional Encoding and Genomic Coordinates

In biology, position matters enormously: the same sequence TATAAA functions as a promoter only at specific distances upstream of a gene. DOGMA's genomic bases encode position implicitly through their region membership and regulatory context, rather than through additive positional encoding. This is more biologically realistic: a cell does not "add" a position signal to each nucleotide; instead, the 3D structure of chromatin determines which regions are accessible.

4.6 Sequence Models: From RNNs to State Spaces

Biological data is inherently sequential: DNA sequences, protein chains, gene expression time series. Sequence models are therefore central to biological AI.

4.6.1 Recurrent Neural Networks

RNNs process sequences by maintaining a hidden state $\mathbf{h}_t$ that is updated at each time step:

$$\mathbf{h}_t = \phi(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}). \tag{4.31}$$

The hidden state acts as a "memory" that accumulates information from previous time steps. At each step, the RNN takes the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$, combines them through a linear transformation, applies a nonlinearity, and produces a new hidden state $\mathbf{h}_t$. This hidden state serves two purposes: it is the representation used for prediction at time t, and it is the memory passed to time $t+1$.

However, vanilla RNNs struggle with long sequences due to the *vanishing gradient problem*: gradients shrink exponentially as they propagate backward through many time steps.

Vanishing gradients: a concrete example. If the gradient at each time step is multiplied by a factor $\alpha < 1$, then after T steps the gradient is α^T. For $\alpha = 0.9$ and $T = 100$: $0.9^{100} \approx 2.7 \times 10^{-5}$. The gradient has effectively vanished, making it impossible to learn long-range dependencies. For DNA sequences, which can be hundreds of thousands of bases long, this is catastrophic: an RNN cannot learn that a promoter 10,000 bases upstream affects a gene's expression.

4.6.2 Long Short-Term Memory (LSTM)

The LSTM architecture [Hochreiter and Schmidhuber, 1997] addresses vanishing gradients by adding a *cell state* $\mathbf{c}_t$ that flows through time with minimal transformation, gated by three learned gates:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \quad \text{(forget gate)} \tag{4.32}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \quad \text{(input gate)} \tag{4.33}$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \quad \text{(candidate cell state)} \tag{4.34}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad \text{(cell state update)} \tag{4.35}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \quad \text{(output gate)} \tag{4.36}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad \text{(hidden state)} \tag{4.37}$$

where $\odot$ denotes element-wise multiplication and $[\cdot, \cdot]$ denotes concatenation.

The key idea is the cell state $\mathbf{c}_t$: information flows along the cell state with only multiplicative gating (the forget gate $\mathbf{f}_t$) and additive updates (the input gate $\mathbf{i}_t \odot \tilde{\mathbf{c}}_t$). If the forget gate is close to 1 and the input gate close to 0, information is preserved indefinitely. This creates a "gradient highway" that allows gradients to flow backward through time without shrinking.

LSTM Gates and Biological Regulation

The LSTM's gating mechanism has a striking parallel to gene regulation:

LSTM Component	Biological Analogue	Function
Forget gate $\mathbf{f}_t$	Silencer element	Suppresses previously stored information
Input gate $\mathbf{i}_t$	Promoter/enhancer	Controls which new information to store
Output gate $\mathbf{o}_t$	Transcription regulation	Controls which information to express
Cell state $\mathbf{c}_t$	Epigenetic memory	Persistent information across time

DOGMA's regulation stage generalizes this parallel into a full architectural principle with typed bases and region-level control. Rather than three gates operating on a single vector, DOGMA applies context-dependent activation weights to entire genomic regions with rich type information.

4.6.3 State Space Models

An emerging alternative to attention-based sequence models is the *structured state space model* (SSM) [Gu et al., 2022, Gu and Dao, 2023]. SSMs process sequences through a linear dynamical system:

$$\mathbf{h}_t = \mathbf{A}\mathbf{h}_{t-1} + \mathbf{B}\mathbf{x}_t, \tag{4.38}$$
$$\mathbf{y}_t = \mathbf{C}\mathbf{h}_t + \mathbf{D}\mathbf{x}_t, \tag{4.39}$$

where $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are structured matrices. The key innovation of SSMs is that the recurrence in Equation 4.38 can be computed as a *convolution* during training (enabling parallelism) and as a *recurrence* during inference (enabling efficient autoregressive generation).

Why the dual computation matters. To see how the recurrence becomes a convolution, unroll the state equation:

$$\mathbf{h}_0 = \mathbf{B}\mathbf{x}_0, \tag{4.40}$$

$$\mathbf{h}_1 = \mathbf{A}\mathbf{B}\mathbf{x}_0 + \mathbf{B}\mathbf{x}_1, \tag{4.41}$$

$$\mathbf{h}_2 = \mathbf{A}^2\mathbf{B}\mathbf{x}_0 + \mathbf{A}\mathbf{B}\mathbf{x}_1 + \mathbf{B}\mathbf{x}_2. \tag{4.42}$$

The output $\mathbf{y}_t = \mathbf{C}\mathbf{h}_t$ depends on all past inputs through the kernel $\bar{K}_t = \mathbf{C}\mathbf{A}^t\mathbf{B}$, yielding $\mathbf{y} = \bar{K} * \mathbf{x}$—a convolution that can be computed in $O(T \log T)$ via FFT during training. During inference, only one step of the recurrence is needed per token, giving $O(1)$ per-token cost.

Complexity comparison.

Architecture	Training	Inference (per token)
RNN / LSTM	$O(T)$ sequential	$O(1)$
Transformer	$O(T^2)$ parallel	$O(T)$ (KV cache)
SSM (S4/Mamba)	$O(T \log T)$ parallel	$O(1)$

The Mamba architecture [Gu and Dao, 2023] makes the SSM matrices *input-dependent*, achieving selective state-space modeling: the model can choose which information to remember or forget based on the input, similar to LSTM gating but with $O(T)$ complexity.

SSMs in DOGMA

DOGMA's ParallelScan computational path implements a variant of selective state-space processing. The state transition matrix $\mathbf{A}$ is parameterized using thermodynamic binding energies from the SantaLucia nearest-neighbor model, providing a biophysically grounded state dynamics. This provides linear-time sequence processing for long genomic sequences while maintaining biological interpretability.

4.7 Evolutionary Algorithms

Evolutionary algorithms (EAs) are optimization methods inspired by biological evolution [Holland, 1975, De Jong, 2006]. They maintain a *population* of candidate solutions and iteratively apply *selection*, *mutation*, and *recombination* to improve population fitness.

4.7.1 The Genetic Algorithm: Step by Step

A standard genetic algorithm (GA) proceeds through the following steps:

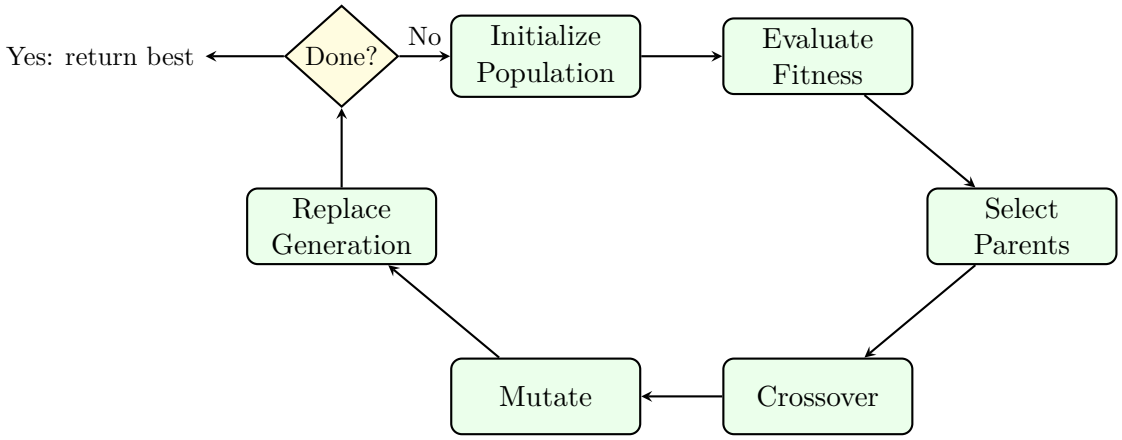
1. **Initialize:** Create a random population of N candidate solutions ("individuals"), each encoded as a genome (e.g., a bit string, a real-valued vector, or a structured object).

2. **Evaluate:** Compute a fitness score $f(\mathbf{x}_i)$ for each individual $\mathbf{x}_i$.

3. **Select parents:** Choose individuals for reproduction, with probability proportional to fitness. Common selection methods:

- *Tournament selection:* Randomly pick k individuals; the fittest wins.
- *Roulette wheel:* Select with probability $p_i = f(\mathbf{x}_i)/\sum_j f(\mathbf{x}_j)$.
- *Rank selection:* Sort by fitness; select by rank rather than raw fitness value.

4. **Crossover (recombination):** Combine two parents to produce offspring:
   - *One-point crossover:* Choose a random split point; child gets first part from parent A and second part from parent B.
   - *Uniform crossover:* For each gene, randomly choose from parent A or B.
5. **Mutate:** Randomly modify offspring with small probability p_m (e.g., flip a bit, add Gaussian noise to a real-valued gene).
6. **Replace:** Form the next generation. Options include generational replacement (all individuals replaced) or steady-state (only a few replaced per iteration).
7. **Repeat** steps 2–6 until convergence or a computational budget is exhausted.



Worked example: evolving a 6-bit binary string. Suppose our fitness function counts the number of 1-bits (the "OneMax" problem). We want to find the string 111111.

1. **Generation 0 (initialize):** Random population of $N = 4$:

Individual	Genome	Fitness
$\mathbf{x}_1$	100110	3
$\mathbf{x}_2$	011001	3
$\mathbf{x}_3$	110100	3
$\mathbf{x}_4$	001011	3

2. **Selection (tournament, $k = 2$):**
   - Pair 1: $\mathbf{x}_1$ (3) vs $\mathbf{x}_3$ (3) $\to$ $\mathbf{x}_3$ wins (tie-break).
   - Pair 2: $\mathbf{x}_2$ (3) vs $\mathbf{x}_4$ (3) $\to$ $\mathbf{x}_2$ wins.
3. **Crossover (one-point at position 3):**

- Parent $\mathbf{x}_3 = \mathbf{110}|100$, Parent $\mathbf{x}_2 = \mathbf{011}|001$
- Child 1: $110|001 = 110001$ (fitness 3)
- Child 2: $011|100 = 011100$ (fitness 3)

4. **Mutation ($p_m = 0.1$ per bit):** Suppose bit 6 of child 1 flips: $110001 \to 110011$ (fitness 4). Child 2 has no mutations.

5. **Generation 1:** Replace worst two individuals. New population: $\{110100, 011001, 110011, 011100\}$. Best fitness improved from 3 to 4.

After several more generations, mutation and crossover create individuals with progressively more 1-bits until the optimum 111111 is found. Note that no gradient was computed—the search is guided entirely by fitness evaluation and random variation.

Worked example: evolving a DNA promoter score. Suppose we want to find a 10-base DNA sequence that maximizes a promoter strength predictor $f : \{A, C, G, T\}^{10} \to [0, 1]$. We initialize a population of $N = 50$ random sequences, evaluate each with the predictor, apply tournament selection ($k = 3$), one-point crossover, and point mutations (each base mutated with probability $p_m = 0.05$). After 100 generations, the population converges toward high-scoring promoter sequences—without ever computing a gradient.

4.7.2 Quality-Diversity Algorithms

Standard optimization seeks a single best solution. *Quality-diversity* (QD) algorithms seek a diverse collection of high-quality solutions that cover the space of possible behaviors [Mouret and Clune, 2015].

MAP-Elites

MAP-Elites maintains a grid of behavioral niches, indexed by a *behavioral descriptor* (BD). For each niche, only the best-performing individual is stored. The algorithm:

1. Randomly generate initial individuals and place in grid by BD.
2. Select a random occupied niche; mutate its occupant.
3. Evaluate the mutant's fitness and BD.
4. If the mutant's niche is empty or the mutant beats the current occupant, insert it.
5. Repeat until the grid is filled with diverse, high-quality solutions.

Intuition for MAP-Elites. Consider designing DNA aptamers (short DNA sequences that bind to specific targets). The behavioral descriptor might be a 2D space: (GC content, predicted secondary structure stability). MAP-Elites would fill a grid where each cell contains the best-binding aptamer with that particular GC content and stability combination. This gives the designer a menu of high-quality options spanning different properties—far more useful than a single "best" aptamer.

Novelty search [Lehman and Stanley, 2011] abandons explicit fitness objectives entirely, rewarding solutions that are behaviorally different from previously encountered ones. This can discover solutions that objective-driven search misses due to deceptive fitness landscapes.

Evolutionary Algorithms in DOGMA

DOGMA's evolutionary training loop (Chapter 12) combines multiple EA principles:

- **Mutation:** Prompt genome fragments are modified through targeted edits (point mutations, insertions, deletions, region swaps).
- **Selection:** Multi-objective fitness evaluation using Pareto dominance selects improvements.
- **Novelty:** HelixHash novelty scoring (Chapter 8) prevents premature convergence by measuring structural novelty.
- **Quality-diversity:** The HelixHash archive maintains diverse structural variants, analogous to MAP-Elites.
- **Meta-learning:** Cross-run meta-policy memory enables the system to improve at evolving itself—learning which mutation strategies work best in which contexts.

4.8 Multi-Objective Optimization

Real-world learning problems rarely have a single objective. DOGMA's training loop simultaneously optimizes accuracy, schema compliance, distillation agreement, efficiency, and latency.

4.8.1 Pareto Optimality

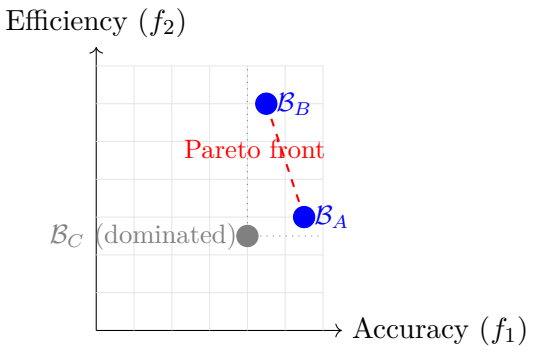
Definition 4.1 (Pareto Dominance and Optimality)**.** Given two solutions $\mathbf{x}$ and $\mathbf{y}$ evaluated on m objectives $f_1, \ldots, f_m$ (all to be maximized):

- $\mathbf{x}$ *Pareto dominates* $\mathbf{y}$ (written $\mathbf{x} \succ \mathbf{y}$) if $f_i(\mathbf{x}) \geq f_i(\mathbf{y})$ for all i and $f_j(\mathbf{x}) > f_j(\mathbf{y})$ for at least one j.
- $\mathbf{x}$ is *Pareto optimal* if no feasible solution dominates it.
- The set of all Pareto-optimal solutions is the *Pareto front*.

Worked example. Consider three genomes evaluated on accuracy (f_1) and efficiency (f_2):

Genome	Accuracy (f_1)	Efficiency (f_2)
$\mathcal{B}_A$	0.95	0.60
$\mathcal{B}_B$	0.85	0.90
$\mathcal{B}_C$	0.80	0.55

$\mathcal{B}_A$ and $\mathcal{B}_B$ are both Pareto optimal—neither dominates the other (A is better on accuracy, B on efficiency). $\mathcal{B}_C$ is dominated by $\mathcal{B}_B$ (worse on both objectives), so C is *not* Pareto optimal. Note that $\mathcal{B}_A$ does not dominate $\mathcal{B}_C$ either: although A is better on accuracy ($0.95 > 0.80$) and efficiency ($0.60 > 0.55$), it does dominate C since it is better on *both* objectives. The Pareto front is therefore $\{\mathcal{B}_A, \mathcal{B}_B\}$.



4.8.2 Scalarization Methods

The simplest approach to multi-objective optimization is *scalarization*: combine multiple objectives into a single scalar using weights:

$$F_{\text{scalar}}(\mathbf{x}) = \sum_{i=1}^{m} w_i \cdot f_i(\mathbf{x}), \quad \sum_i w_i = 1. \tag{4.43}$$

The choice of weights w_i determines which point on the Pareto front is found. Different weight vectors trace out different points on the front: $w = [1.0, 0.0]$ finds the solution with maximum accuracy (ignoring efficiency), $w = [0.0, 1.0]$ maximizes efficiency, and $w = [0.5, 0.5]$ seeks a balanced compromise.

DOGMA's multi-objective scoring uses a weighted combination where the weights are *dynamically adjusted* by the Tera regime state (Chapter 9), creating an adaptive scalarization that shifts emphasis based on the current evolutionary pressure landscape.

$$w_i(t) = \text{softmax}(\mathbf{W}_{\text{tera}} \cdot \mathbf{r}(t) + \mathbf{b})_i \tag{4.44}$$

where $\mathbf{r}(t)$ is the Tera regime state vector at time t. When the system is stagnating on accuracy, Tera increases the accuracy weight; when efficiency is lagging, it shifts emphasis to efficiency. This dynamic reweighting is analogous to how biological organisms shift resource allocation under different environmental pressures.

Implementation Note

In practice, DOGMA's Tera state smoothly interpolates between weight configurations rather than jumping between them. The softmax temperature is annealed during training: high temperature early on (exploring the full Pareto front) and low temperature later (focusing on the most promising region). This prevents the optimizer from oscillating between objectives and ensures stable convergence.

4.9 Bringing It Together: The DOGMA Learning Stack

The ML concepts introduced in this chapter form the foundation of DOGMA's learning stack. The following table maps each ML concept to its role in DOGMA:

ML Concept	DOGMA Component	Role
Embeddings	Genomic base content σ_i	Represent symbols as typed vectors
Activation functions	Regulatory weights ρ_i	Context-dependent activation
CNNs	Strand displacement path	Local pattern detection
RNNs/LSTMs	Epigenetic memory (v2)	Persistent state across time
SSMs	ParallelScan path	Linear-time sequence processing
Backpropagation	Foundation model training	Train neural components
Evolutionary search	Evolutionary training loop	Optimize prompt genomes
Multi-objective optimization	Tera regime dynamics	Balance competing objectives
Quality-diversity	HelixHash archive	Maintain structural diversity
Regularization	Type compatibility constraints	Prevent degenerate solutions

This mapping illustrates DOGMA's synthetic approach: rather than choosing between gradient-based and evolutionary optimization, between local and global computation, or between structured and unstructured representations, DOGMA combines the best of each paradigm within a biologically grounded framework.

Two-Level Optimization

DOGMA operates a two-level optimization process that mirrors biology's own dual optimization:

- **Inner loop (gradient descent):** Trains neural network parameters (weights, embeddings) through backpropagation on differentiable losses. This is analogous to *learning within an organism's lifetime*—adjusting synaptic weights through experience.
- **Outer loop (evolutionary search):** Evolves prompt genomes, architectural configurations, and regulatory structures through population-based search. This is analogous to *evolution across generations*—selecting genomes that produce fit organisms.

The inner loop handles what is differentiable; the outer loop handles what is not. Together, they cover the full optimization landscape that DOGMA must navigate.

4.10 Exercises

4.1. **[Fundamentals]** Derive the gradient of the cross-entropy loss $\mathcal{L} = -\sum_i y_i \log \hat{y}_i$ with respect to the logits $\mathbf{z}$, where $\hat{y}_i = \text{softmax}(\mathbf{z})_i$. Show that the result simplifies to $\hat{\mathbf{y}} - \mathbf{y}$. *Hint:* Start by computing $\partial \hat{y}_i / \partial z_j$ for the cases $i = j$ and $i \neq j$ separately, using the quotient rule on the softmax definition.

4.2. **[Coding]** Implement a single-layer neural network from scratch (NumPy only) that classifies

DNA 3-mers into categories based on GC content (low: 0–1 G/C, medium: 2, high: 3 G/C). Train with SGD and plot the learning curve. Report final accuracy.

4.3. [**Analysis**] Compare the LSTM cell state $\mathbf{c}_t$ with a biological gene's expression level $x_i(t)$ governed by Equation 3.2 from Chapter 3. Identify the biological analogues of the forget gate, input gate, and output gate. Are there regulatory mechanisms that have no LSTM analogue?

4.4. [**Backpropagation**] Perform the complete forward and backward pass for a 2-layer MLP with weights $W^{(1)} = \begin{bmatrix} 0.3 & -0.2 \\ 0.4 & 0.1 \end{bmatrix}$, $\mathbf{b}^{(1)} = [0, 0]$, $\mathbf{w}^{(2)} = [0.5, -0.3]$, $b^{(2)} = 0$, input $\mathbf{x} = [1, 2]$, and true label $y = 1$. Use ReLU for the hidden layer and sigmoid for the output. Compute all gradients and the updated weights after one step of SGD with $\eta = 0.1$.

4.5. [**Convolutions**] Design a set of 1D convolutional filters (specify the weights) that detect the following DNA motifs: (a) the start codon ATG, (b) any purine-pyrimidine alternating pattern, (c) a CpG dinucleotide. Assume 4-channel one-hot encoded input (A, C, G, T).

4.6. [**Theory**] Prove that the Pareto front of a bi-objective optimization problem with convex feasible set is a connected curve. Give a counterexample for non-convex problems.

4.7. [**Design**] Design a quality-diversity algorithm for evolving DNA primer sequences. Define the behavioral descriptor space, the quality metric, and the mutation operators. Explain how your design prevents the population from converging prematurely.

4.8. [**Research**] Read Hansen and Ostermeier [2001] on CMA-ES. How does CMA-ES's covariance adaptation relate to DOGMA's Tera regime state adaptation? Both systems learn about the local landscape to guide search—compare their mechanisms.

4.9. [**Activation Functions**] Implement sigmoid, ReLU, GELU, and Swish in Python. For each, plot the function and its derivative on $[-5, 5]$. Which derivative has the largest maximum value? Which never saturates?

4.10. [**Embeddings**] Train Word2Vec-style embeddings on DNA 4-mers using the skip-gram objective on a small genome (e.g., E. coli). Visualize the embeddings using t-SNE. Do reverse complement pairs cluster together?

4.11. [**Gradient Descent**] Implement gradient descent on $f(\theta_1, \theta_2) = (\theta_1 - 2)^2 + 10(\theta_2 - \theta_1^2)^2$ (a scaled Rosenbrock function). Starting from $\theta_0 = (-1, 1)$, compare the trajectories of plain SGD ($\eta = 0.001$), SGD with momentum ($\beta = 0.9$), and Adam. Plot the trajectories overlaid on a contour plot of f. Which optimizer converges fastest? Why?

4.12. [**Evolutionary Algorithms**] Implement a genetic algorithm to find the DNA 8-mer with maximum GC content (trivial optimal: `GCGCGCGC`). Use tournament selection ($k = 2$), uniform crossover, and point mutation ($p_m = 0.1$ per base). Track the population's average and best fitness over 50 generations. How does increasing the population size from $N = 10$ to $N = 100$ affect convergence speed?

4.11 Key Takeaways

Key Takeaways

- Machine learning is function approximation from data; the loss function defines what "good" means, and the optimizer finds parameters that minimize loss.
- Neural networks gain expressiveness from depth and nonlinearity; the universal approximation theorem guarantees expressiveness, but depth provides efficiency.
- Backpropagation computes gradients efficiently via the chain rule; Adam is the standard optimizer for modern deep learning.
- Regularization (L2, dropout, early stopping) prevents overfitting by constraining model complexity.
- Convolutional networks detect local patterns through learned filters—DOGMA's strand displacement path generalizes this with typed convolution.
- Self-supervised learning (next-token prediction) is the dominant paradigm for both language and genomic foundation models.
- Representation learning maps discrete symbols to continuous geometry—DOGMA extends this with typed, weighted genomic bases carrying regulatory state.
- LSTM gating parallels biological gene regulation; DOGMA generalizes this to full region-level regulatory control.
- State space models offer $O(T)$ sequence processing—DOGMA's ParallelScan path adopts this efficiency with biophysical parameterization.
- Evolutionary algorithms provide gradient-free optimization—DOGMA combines gradient training with evolutionary prompt genome optimization.
- Multi-objective optimization with dynamic Tera-driven weight adaptation enables balanced improvement across competing objectives.

Suggested Reading

- Goodfellow et al. [2016]: Comprehensive deep learning reference—the standard textbook.
- Hochreiter and Schmidhuber [1997]: LSTM—foundational for sequence modeling and gating mechanisms.
- Kingma and Ba [2015]: Adam optimizer—the workhorse of modern deep learning.
- Gu and Dao [2023]: Mamba—selective state spaces as an attention alternative.
- Srivastava et al. [2014]: Dropout regularization—simple but effective.
- Holland [1975]: Genetic algorithms—the original EA framework.
- Mouret and Clune [2015]: MAP-Elites—quality-diversity algorithms.

- Lehman and Stanley [2011]: Novelty search—objectives are not always necessary.
- Ji et al. [2021b]: DNABERT—applying BERT to genomic sequences.
- Zhou and Troyanskaya [2015]: DeepSEA—CNNs for predicting chromatin effects from DNA sequence.

Chapter 5

Transformers and Large Language Models

Attention is all you need—but perhaps not all you want.

— After Vaswani et al., 2017

Chapter 4 established the foundational building blocks of machine learning: neural networks, optimization, embeddings, and evolutionary algorithms. This chapter focuses on the single most consequential architecture of the modern deep learning era: the *transformer*. We trace its development from the attention mechanism through the scaling revolution to the emergent capabilities of large language models (LLMs), and close with a careful analysis of the transformer paradigm's structural limitations—limitations that motivate the DOGMA architecture introduced in Chapter 7.

The transformer is not merely one architecture among many. It is, at the time of writing, the computational substrate upon which virtually all frontier AI systems are built. Understanding it deeply—its mathematical structure, its scaling behavior, its surprising emergent properties, and its fundamental constraints—is prerequisite to understanding why DOGMA proposes a different organizational principle for intelligence.

This chapter is deliberately more detailed than a typical survey. We will work through every matrix multiplication, trace every gradient, and build intuition from concrete numerical examples. If you can reconstruct scaled dot-product attention on a napkin by the end of this chapter, you are ready for DOGMA.

5.1 The Attention Mechanism

5.1.1 From Alignment to Attention

The attention mechanism was originally introduced for neural machine translation, where it allowed the decoder to dynamically focus on relevant parts of the source sentence rather than compressing the entire input into a fixed-length vector [Bahdanau et al., 2015]. The key insight: not all input positions are equally relevant to producing each output position, and the model should learn to *selectively attend* to the most informative sources.

Consider translating the French sentence "Le chat noir dort sur le tapis" into English. When generating the word "black," the model should focus on "noir"; when generating "sleeps," it should

focus on "dort." Before attention, sequence-to-sequence models compressed the entire source sentence into a single fixed-length vector, forcing all information through a bottleneck. Attention removes this bottleneck by allowing the decoder to look back at the full source sequence at every generation step.

Attention and Transcription Factor Binding

Biological transcription factors selectively bind to specific promoter and enhancer regions, activating only the relevant genes for a given cellular context. Attention performs an analogous function: given a query (the current context), it selectively weights source positions (the "genome" of input tokens) to determine which information flows forward. DOGMA's regulation stage (Chapter 7) generalizes this idea beyond pairwise token comparisons to typed, region-level regulatory control.

5.1.2 Intuition: Queries, Keys, and Values

Before diving into the mathematics, let us build intuition for the three central objects in attention: **queries**, **keys**, and **values**. The analogy to a database lookup is instructive:

- **Query (Q):** "What am I looking for?" Each token produces a query vector that encodes what information it needs from other tokens. Think of it as a search query in a database.
- **Key (K):** "What do I contain?" Each token also produces a key vector that advertises what information it has available. Think of it as the index entry in a database.
- **Value (V):** "What do I actually return?" Each token produces a value vector that is the content to be retrieved. Think of it as the data record that the index points to.

The attention mechanism computes a *soft lookup*: each query is compared against all keys to produce a relevance score, and the output is a weighted combination of values, where the weights are proportional to the relevance scores. Unlike a hard database lookup that returns a single record, attention returns a blended combination of all records, weighted by relevance.

Remark 5.1. The query, key, and value vectors are *all derived from the same input* in self-attention. Each token simultaneously asks a question (query), advertises its content (key), and provides retrievable information (value). The three roles are distinguished only by the learned projection matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$.

5.1.3 Scaled Dot-Product Attention

The transformer's attention mechanism operates on three learned projections of the input: *queries* $\mathbf{Q}$, *keys* $\mathbf{K}$, and *values* $\mathbf{V}$. Given an input matrix $\mathbf{X} \in \mathbb{R}^{T \times d}$ representing T tokens of dimension d, these projections are:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V, \tag{5.1}$$

where $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d \times d_k}$ and $\mathbf{W}_V \in \mathbb{R}^{d \times d_v}$ are learned parameter matrices. Scaled dot-product attention then computes:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}. \tag{5.2}$$

Let us break this computation into four explicit steps:

1. **Compute raw attention scores:** $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{T\times T}$. Entry S_{ij} measures how much token i's query aligns with token j's key—i.e., how relevant token j's content is to token i's current need.

2. **Scale:** $\mathbf{S}' = \mathbf{S}/\sqrt{d_k}$. Without scaling, the dot products grow in magnitude proportionally to d_k (since each is a sum of d_k terms). Large magnitudes push the softmax into saturated regions where gradients vanish. Dividing by $\sqrt{d_k}$ keeps the variance of the scores approximately 1, regardless of the dimension.

3. **Normalize with softmax:** $\mathbf{A} = \text{softmax}(\mathbf{S}')$, applied row-wise. Each row of $\mathbf{A}$ is a probability distribution over all tokens, with $A_{ij} \geq 0$ and $\sum_j A_{ij} = 1$. This converts raw scores into normalized attention weights.

4. **Aggregate values:** $\mathbf{O} = \mathbf{A}\mathbf{V} \in \mathbb{R}^{T\times d_v}$. The output for token i is a weighted average of all value vectors: $\mathbf{o}_i = \sum_j A_{ij}\mathbf{v}_j$.

Attention Complexity

The attention matrix $\mathbf{A} = \text{softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d_k})$ has shape $T \times T$, making the time and space complexity of self-attention $O(T^2 \cdot d_k)$. For a sequence of length $T = 128{,}000$ tokens with $d_k = 128$, the attention matrix alone contains $\approx 1.6 \times 10^{10}$ entries—a fundamental bottleneck that limits the transformer's ability to process long sequences efficiently.

5.1.4 Why the Scaling Factor Matters

To understand the scaling factor $1/\sqrt{d_k}$ more concretely, suppose the entries of $\mathbf{q}$ and $\mathbf{k}$ are independent random variables with mean 0 and variance 1. Then the dot product $\mathbf{q}\cdot\mathbf{k} = \sum_{j=1}^{d_k} q_j k_j$ has mean 0 and variance d_k (since each product $q_j k_j$ contributes variance 1, and the terms are independent). For $d_k = 64$, the standard deviation of $\mathbf{q}\cdot\mathbf{k}$ is $\sqrt{64} = 8$. Dot products of magnitude 8 or larger push softmax outputs toward 0 or 1, creating nearly one-hot attention patterns that produce vanishingly small gradients. Dividing by $\sqrt{d_k} = 8$ rescales the standard deviation back to 1, keeping softmax in a well-behaved regime.

5.1.5 Worked Example: Self-Attention with 3 Tokens

Let us trace through a complete self-attention computation with $T = 3$ tokens and $d_k = d_v = 4$. Suppose after the linear projections, we have:

$$\mathbf{Q} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{K} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \quad \mathbf{V} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}. \tag{5.3}$$

Step 1: Raw attention scores. Compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$:

$$\mathbf{S} = \begin{pmatrix} \mathbf{q}_1\cdot\mathbf{k}_1 & \mathbf{q}_1\cdot\mathbf{k}_2 & \mathbf{q}_1\cdot\mathbf{k}_3 \\ \mathbf{q}_2\cdot\mathbf{k}_1 & \mathbf{q}_2\cdot\mathbf{k}_2 & \mathbf{q}_2\cdot\mathbf{k}_3 \\ \mathbf{q}_3\cdot\mathbf{k}_1 & \mathbf{q}_3\cdot\mathbf{k}_2 & \mathbf{q}_3\cdot\mathbf{k}_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{pmatrix}. \tag{5.4}$$

For example, $S_{11} = (1)(0)+(0)(1)+(1)(1)+(0)(0) = 1$, and $S_{33} = (1)(1)+(1)(1)+(0)(0)+(0)(0) = 2$.

Step 2: Scale. With $d_k = 4$, we divide by $\sqrt{4} = 2$:

$$\mathbf{S}' = \frac{\mathbf{S}}{\sqrt{d_k}} = \begin{pmatrix} 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & 0.5 & 1.0 \end{pmatrix}. \tag{5.5}$$

Step 3: Softmax (row-wise). For rows 1 and 2, all entries are equal (0.5), so softmax gives uniform weights: $A_{1j} = A_{2j} = 1/3$ for all j. For row 3:

$$A_{31} = A_{32} = \frac{e^{0.5}}{2e^{0.5} + e^{1.0}} \approx \frac{1.649}{3.298 + 2.718} \approx 0.274, \quad A_{33} = \frac{e^{1.0}}{2e^{0.5} + e^{1.0}} \approx \frac{2.718}{6.016} \approx 0.452. \tag{5.6}$$

The full attention matrix is:

$$\mathbf{A} \approx \begin{pmatrix} 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \\ 0.274 & 0.274 & 0.452 \end{pmatrix}. \tag{5.7}$$

Step 4: Weighted value aggregation. The output $\mathbf{O} = \mathbf{AV}$:

$$\mathbf{o}_1 = 0.333 \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} + 0.333 \cdot \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} + 0.333 \cdot \begin{pmatrix} 9 \\ 10 \\ 11 \\ 12 \end{pmatrix} = \begin{pmatrix} 5.0 \\ 6.0 \\ 7.0 \\ 8.0 \end{pmatrix}. \tag{5.8}$$

Token 1 attends uniformly, so its output is the average of all value vectors. Token 3, however, attends more heavily to itself (weight 0.452 vs. 0.274), so its output is shifted toward $\mathbf{v}_3$:

$$\mathbf{o}_3 \approx 0.274 \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} + 0.274 \cdot \begin{pmatrix} 5 \\ 6 \\ 7 \\ 8 \end{pmatrix} + 0.452 \cdot \begin{pmatrix} 9 \\ 10 \\ 11 \\ 12 \end{pmatrix} \approx \begin{pmatrix} 5.70 \\ 6.70 \\ 7.70 \\ 8.70 \end{pmatrix}. \tag{5.9}$$

Example 5.1 (Interpreting Attention Weights)**.** In the worked example above, tokens 1 and 2 attend uniformly to all tokens—they have no strong preference. Token 3 attends preferentially to itself because $\mathbf{q}_3 \cdot \mathbf{k}_3 = 2 > 1 = \mathbf{q}_3 \cdot \mathbf{k}_j$ for $j \neq 3$. In practice, attention heads learn to produce query-key alignments that reflect semantic relationships: a pronoun's query might strongly match the key of its antecedent, producing a high attention weight that copies the antecedent's information into the pronoun's representation.

5.1.6 Visualizing Self-Attention

Figure 5.1 illustrates the self-attention mechanism with colored arrows showing how tokens selectively attend to each other.

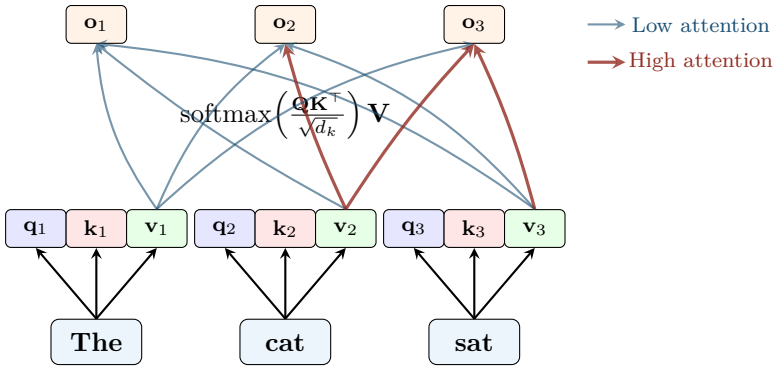


Figure 5.1: Self-attention mechanism for three tokens. Each token is projected into query ($\mathbf{q}$), key ($\mathbf{k}$), and value ($\mathbf{v}$) vectors. Attention scores (computed as $\mathbf{q}_i \cdot \mathbf{k}_j/\sqrt{d_k}$, then softmax-normalized) determine how much of each value vector contributes to each output. Thicker red arrows indicate higher attention weights.

5.2 Multi-Head Attention

5.2.1 Why Multiple Heads?

A single attention head computes a single set of attention weights for each pair of tokens. But different aspects of the relationship between two tokens may be relevant simultaneously. For instance, when processing the sentence "The animal didn't cross the street because it was too tired," one attention head might need to resolve the pronoun "it" (a syntactic task), while another might need to capture the semantic relationship between "tired" and "animal" (a semantic task). A single head must compress all of these relationships into one set of weights—an information bottleneck.

Multi-head attention solves this by running h independent attention computations in parallel, each with its own learned projections:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\, \mathbf{W}_O, \tag{5.10}$$

where each head is:

$$\text{head}_i = \text{Attention}(\mathbf{X}\mathbf{W}_Q^{(i)},\ \mathbf{X}\mathbf{W}_K^{(i)},\ \mathbf{X}\mathbf{W}_V^{(i)}), \tag{5.11}$$

and $\mathbf{W}_O \in \mathbb{R}^{hd_v \times d}$ is a learned output projection.

5.2.2 Head Splitting and Concatenation

In practice, multi-head attention does *not* require h separate sets of full-sized projection matrices. Instead, the model dimension d is split evenly across the h heads: each head operates on $d_k = d_v = d/h$ dimensions. This means:

- $\mathbf{W}_Q^{(i)}, \mathbf{W}_K^{(i)} \in \mathbb{R}^{d \times (d/h)}$ — each head projects from the full d-dimensional space into a (d/h)-dimensional subspace.
- Each head independently computes attention in its (d/h)-dimensional subspace, producing output of shape $T \times (d/h)$.
- The h head outputs are concatenated along the feature dimension, giving shape $T \times d$.

- The concatenated output is projected through $\mathbf{W}_O \in \mathbb{R}^{d \times d}$ to produce the final output.

The total number of parameters is the same as a single head with $d_k = d$: the computation is *repartitioned*, not increased. With $h = 12$ heads and $d = 768$ (as in BERT-base), each head operates on $d_k = 64$ dimensions.

5.2.3 What Different Heads Learn

Empirical analysis reveals that attention heads specialize in different linguistic functions [Voita et al., 2019]:

- **Positional heads:** Attend to the immediately preceding or following token—capturing local context.
- **Syntactic heads:** Attend to syntactically related tokens (e.g., a verb head attends to its subject).
- **Rare-word heads:** Attend to the least frequent token in the sequence—potentially copying specialized information.
- **Delimiter heads:** Attend to separator tokens (periods, commas)—capturing sentence boundaries.

This emergent specialization occurs without any explicit supervision: the heads discover useful attention patterns through gradient descent alone.

5.2.4 Multi-Head Attention Diagram

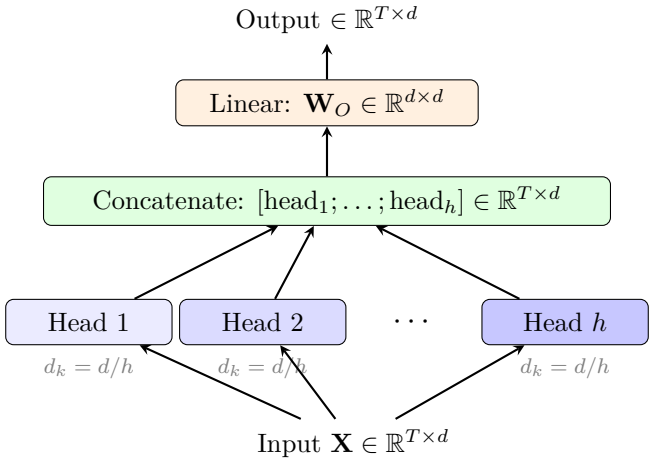


Figure 5.2: Multi-head attention. The input is processed by h independent attention heads, each operating in a (d/h)-dimensional subspace. Outputs are concatenated and projected back to the full d-dimensional space.

5.3 The Transformer Architecture

Vaswani et al. [2017a] introduced the transformer as a sequence-to-sequence architecture composed entirely of attention and feedforward layers, eliminating the recurrence that had characterized prior sequence models. The architecture consists of an *encoder* stack and a *decoder* stack, each built from repeated identical layers.

5.3.1 Encoder and Decoder Stacks

Each encoder layer applies two sub-layers:

1. **Multi-head self-attention:** Each position attends to all positions in the input.
2. **Position-wise feedforward network (FFN):** A two-layer MLP applied independently to each position:

$$\text{FFN}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2. \tag{5.12}$$

The FFN is a critical but often underappreciated component. While attention handles *inter-token* communication (allowing tokens to share information), the FFN handles *intra-token* processing (transforming each token's representation independently). In GPT-3, the FFN inner dimension is $4d = 49{,}152$ for $d = 12{,}288$—meaning the FFN has $4\times$ the width of the model dimension. Research suggests that FFN layers function as key-value memories, storing factual associations learned during training.

Each decoder layer adds a third sub-layer: *cross-attention* over the encoder output, allowing the decoder to condition on the full input sequence while generating the output autoregressively.

5.3.2 Residual Connections and Layer Normalization

Both encoder and decoder employ *residual connections* [He et al., 2016] around each sub-layer, followed by *layer normalization* [Ba et al., 2016]:

$$\mathbf{y} = \text{LayerNorm}(\mathbf{x} + \text{SubLayer}(\mathbf{x})). \tag{5.13}$$

Residual connections add the sub-layer's input directly to its output: $\mathbf{x} + \text{SubLayer}(\mathbf{x})$. This creates a "skip connection" that allows gradients to flow directly through the network without passing through every sub-layer. Without residual connections, training transformers with $L > 6$ layers would be extremely difficult due to vanishing or exploding gradients.

Layer normalization normalizes activations across the feature dimension for each token independently. Given a vector $\mathbf{x} \in \mathbb{R}^d$:

$$\text{LayerNorm}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \mu}{\sigma + \epsilon} + \beta, \tag{5.14}$$

where $\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$, $\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$, and $\gamma, \beta \in \mathbb{R}^d$ are learned scale and shift parameters. The small constant ϵ (typically 10^{-5}) prevents division by zero.

Modern implementations often use *pre-norm* placement, applying normalization before the sub-layer rather than after:

$$\mathbf{y} = \mathbf{x} + \text{SubLayer}(\text{LayerNorm}(\mathbf{x})), \tag{5.15}$$

which improves training stability for very deep models. Pre-norm is now standard in models like GPT-3, LLaMA, and their descendants.

5.3.3 The Complete Transformer Block

Figure 5.3 illustrates the structure of a single transformer decoder block with all components labeled.

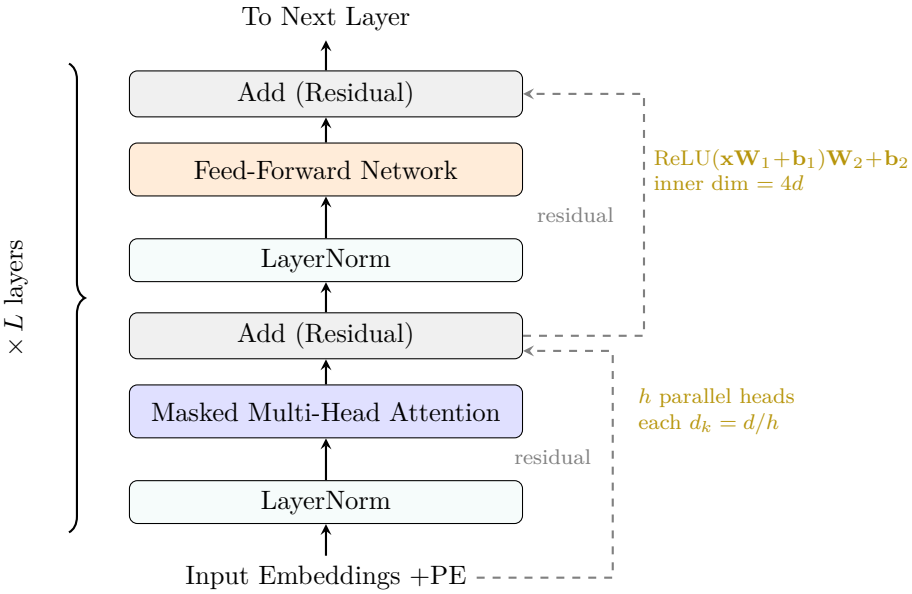


Figure 5.3: A single transformer decoder block with pre-norm residual connections. The masked multi-head attention sub-layer prevents each position from attending to future tokens. LayerNorm is applied before each sub-layer (pre-norm configuration). The block is repeated L times to form the full decoder stack.

5.4 Positional Encoding

5.4.1 The Permutation Equivariance Problem

Self-attention is *permutation equivariant*: if we shuffle the order of input tokens, the output is shuffled in exactly the same way. Mathematically, for any permutation matrix $\mathbf{P}$:

$$\text{Attention}(\mathbf{PQ}, \mathbf{PK}, \mathbf{PV}) = \mathbf{P} \cdot \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}). \tag{5.16}$$

This means that without additional information, the transformer treats "The cat sat" and "sat cat The" identically. Since word order is essential for meaning, positional information must be injected explicitly.

5.4.2 Sinusoidal Positional Encodings

Vaswani et al. [2017a] proposed sinusoidal positional encodings, where each position t and dimension i receives a unique value:

$$\text{PE}(t, 2i) = \sin\left(\frac{t}{10000^{2i/d}}\right), \tag{5.17}$$

$$\text{PE}(t, 2i+1) = \cos\left(\frac{t}{10000^{2i/d}}\right). \tag{5.18}$$

These encodings are added to the input embeddings: $\mathbf{x}'_t = \mathbf{x}_t + \text{PE}(t)$.

Why sinusoids? The key property is that relative positions can be computed as linear transformations. For any fixed offset k, there exists a matrix $\mathbf{M}_k$ (independent of position t) such that $\text{PE}(t+k) = \mathbf{M}_k \cdot \text{PE}(t)$. This follows from the trigonometric identity $\sin(\alpha+\beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$. The model can therefore learn to attend to relative positions (e.g., "the token 3 positions back") using fixed linear operations, without needing to memorize absolute positions.

Multi-frequency structure. Different dimensions encode position at different frequencies. Low dimensions ($i \approx 0$) use high frequencies (fast oscillation), capturing fine-grained positional differences. High dimensions ($i \approx d/2$) use low frequencies (slow oscillation), capturing long-range positional structure. The model has access to a rich, multi-scale representation of position.

Example 5.2 (Positional Encoding Values)**.** For $d = 4$, the positional encoding at position t is a 4-dimensional vector:

$$\text{PE}(t) = \begin{pmatrix} \sin(t/10000^{0/4}) \\ \cos(t/10000^{0/4}) \\ \sin(t/10000^{2/4}) \\ \cos(t/10000^{2/4}) \end{pmatrix} = \begin{pmatrix} \sin(t) \\ \cos(t) \\ \sin(t/100) \\ \cos(t/100) \end{pmatrix}. \tag{5.19}$$

Dimension 0 oscillates rapidly (period $2\pi \approx 6.3$ positions), while dimension 2 oscillates slowly (period $200\pi \approx 628$ positions). At position $t = 0$: $\text{PE}(0) = (0, 1, 0, 1)^\top$. At position $t = 1$: $\text{PE}(1) \approx (0.841, 0.540, 0.010, 1.000)^\top$.

5.4.3 Learned and Relative Position Encodings

Learned positional embeddings. An alternative is to learn a separate embedding vector for each position, stored as a matrix $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{T_{\max} \times d}$. GPT-2 and BERT use this approach. The downside: the model cannot generalize to positions beyond $T_{\max}$.

Rotary Position Embeddings (RoPE). Modern models commonly use RoPE [Su et al., 2024], which encode relative distances through rotation matrices applied to query and key vectors. RoPE rotates pairs of dimensions by an angle proportional to position:

$$\text{RoPE}(\mathbf{x}_t, t) = \begin{pmatrix} x_1\cos(t\theta_1) - x_2\sin(t\theta_1) \\ x_1\sin(t\theta_1) + x_2\cos(t\theta_1) \\ \vdots \\ x_{d-1}\cos(t\theta_{d/2}) - x_d\sin(t\theta_{d/2}) \\ x_{d-1}\sin(t\theta_{d/2}) + x_d\cos(t\theta_{d/2}) \end{pmatrix}, \tag{5.20}$$

where $\theta_i = 10000^{-2i/d}$. The key property is that the dot product $\text{RoPE}(\mathbf{q}_m, m)^\top \text{RoPE}(\mathbf{k}_n, n)$ depends only on $\mathbf{q}_m$, $\mathbf{k}_n$, and the *relative position* $m - n$, not on the absolute positions m and n. This enables length generalization: models trained with RoPE can often handle sequences longer than those seen during training.

5.5 GPT and Autoregressive Language Models

5.5.1 Decoder-Only Architecture

While the original transformer used both encoder and decoder stacks, the GPT family [Radford et al., 2019] demonstrated that a *decoder-only* architecture—using only the masked self-attention

stack—is sufficient for powerful language modeling. The key simplification: by training the model to predict the next token given all preceding tokens, the encoder becomes unnecessary.

A decoder-only model with parameters θ defines a probability distribution over sequences:

$$p_\theta(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_1, \ldots, x_{t-1}). \tag{5.21}$$

This factorization is exact by the chain rule of probability. The model is trained by maximizing the log-likelihood of the training corpus, which is equivalent to minimizing the cross-entropy loss introduced in Equation 4.4 of Chapter 4.

5.5.2 The Next-Token Prediction Objective

The training objective for autoregressive language models is deceptively simple: predict the next token. Given a sequence of tokens $(x_1, x_2, \ldots, x_T)$ from the training corpus, the cross-entropy loss is:

$$\mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \log p_\theta(x_t \mid x_1, \ldots, x_{t-1}). \tag{5.22}$$

At each position t, the model produces a probability distribution over the entire vocabulary $\mathcal{V}$ (typically 32,000 to 128,000 tokens). The loss penalizes the model for assigning low probability to the actual next token. Minimizing this loss across trillions of tokens forces the model to develop rich internal representations of language, world knowledge, and reasoning patterns.

Example 5.3 (Cross-Entropy Loss Computation)**.** Suppose the vocabulary is $\mathcal{V} = \{\texttt{the}, \texttt{cat}, \texttt{sat}, \texttt{on}\}$ and we are predicting position 3 given "The cat ___". If the model assigns probabilities $(0.05, 0.10, 0.80, 0.05)$ to the four tokens, the loss for this position is:

$$\mathcal{L}_3 = -\log p(\texttt{sat}) = -\log(0.80) \approx 0.223. \tag{5.23}$$

If the model were less confident and assigned $p(\texttt{sat}) = 0.25$, the loss would be $-\log(0.25) \approx 1.386$—much higher. The loss is minimized (to zero) only when the model assigns probability 1 to the correct next token.

5.5.3 Causal Masking

To enforce the autoregressive property during training, a *causal mask* $\mathbf{M}$ is applied to the attention scores before the softmax:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right), \quad M_{ij} = \begin{cases} 0 & \text{if } i \geq j, \\ -\infty & \text{if } i < j. \end{cases} \tag{5.24}$$

This ensures that position i can only attend to positions $j \leq i$, preventing information leakage from future tokens. Causal masking enables efficient training: the entire sequence can be processed in a single forward pass, with losses computed at every position simultaneously (teacher forcing).

Causal Masking vs. Biological Directionality

DNA transcription proceeds in the $5' \to 3'$ direction, and ribosomes translate mRNA codons sequentially from start to stop. There is a natural directionality to biological information flow. Causal masking in autoregressive models enforces a similar sequential constraint. However, biological systems also permit *bidirectional* regulatory signals: enhancers can act upstream or downstream of their target genes, and chromatin loops bring distant regulatory elements into spatial proximity. DOGMA's regulation stage captures this bidirectional control while maintaining directionality in the transcription and expression stages.

5.6 Training Large Language Models

5.6.1 Training Data Scale

Modern LLMs are trained on colossal datasets. The training corpus for a frontier model typically includes:

- **Web text:** Crawled and filtered web pages (Common Crawl, typically 1–5 trillion tokens after filtering).
- **Books:** Digitized books providing long-form, high-quality prose.
- **Code:** Source code from public repositories (GitHub), which improves reasoning abilities.
- **Scientific papers:** ArXiv, PubMed, and other academic sources.
- **Conversational data:** Forums, Q&A sites, and curated dialogue.

The total training corpus for frontier models has grown dramatically: GPT-3 used ∼300B tokens (2020), LLaMA used 1.4T tokens (2023), and LLaMA-3 used 15T tokens (2024). Data quality, deduplication, and filtering are now recognized as equally important as scale.

5.6.2 Compute Requirements

Training a large language model requires enormous computational resources. The compute budget is measured in FLOPs (floating-point operations). A useful approximation for the total training FLOPs of a transformer is:

$$C \approx 6ND, \tag{5.25}$$

where N is the number of parameters and D is the number of training tokens. The factor of 6 accounts for the forward pass ($\sim 2ND$) and backward pass ($\sim 4ND$). For LLaMA-3 70B trained on 15T tokens: $C \approx 6 \times 70 \times 10^9 \times 15 \times 10^{12} = 6.3 \times 10^{24}$ FLOPs.

Hardware and Cost

Training LLaMA-3 405B required 30.84 million GPU-hours on NVIDIA H100 GPUs. At cloud prices of approximately \$2–3 per GPU-hour, this represents a training cost of \$60–90 million. The hardware is typically organized into clusters of thousands of GPUs connected by high-bandwidth interconnects (NVLink, InfiniBand), with sophisticated parallelism strategies (tensor parallelism, pipeline parallelism, data parallelism) to distribute the computation. This extreme resource requirement is a key motivator for exploring whether structured architectures like DOGMA can achieve better performance per FLOP.

5.7 Scaling Laws for Language Models

Kaplan et al. [2020a] established that transformer language model performance follows remarkably smooth power-law relationships with three key variables: the number of model parameters N, the dataset size D (in tokens), and the compute budget C (in FLOPs).

5.7.1 Power-Law Relationships

The cross-entropy loss L on held-out data decreases as a power law in each variable, when the other two are not bottlenecked:

$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad \alpha_N \approx 0.076, \tag{5.26}$$

$$L(D) \approx \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad \alpha_D \approx 0.095, \tag{5.27}$$

$$L(C) \approx \left(\frac{C_c}{C}\right)^{\alpha_C}, \quad \alpha_C \approx 0.050, \tag{5.28}$$

where N_c, D_c, C_c are scaling constants. The loss decreases as a power law over many orders of magnitude, with no sign of saturation within the explored range.

Example 5.4 (Reading the Scaling Exponents)**.** The exponent $\alpha_N \approx 0.076$ means that to halve the loss, you need to increase the number of parameters by a factor of $2^{1/0.076} \approx 2^{13.2} \approx 9{,}400$. Scaling laws reveal that progress from scaling alone is *expensive*: each halving of loss requires roughly four orders of magnitude more parameters. This motivates the search for better architectures (like DOGMA) that could achieve steeper scaling exponents.

5.7.2 Compute-Optimal Training (Chinchilla)

Hoffmann et al. [2022a] refined these results, showing that for a fixed compute budget, the optimal allocation splits compute roughly equally between model size and training data. Their analysis suggested that many large models (including the original GPT-3) were significantly *undertrained*: given the compute used, more tokens and fewer parameters would have yielded lower loss.

The Chinchilla scaling law can be expressed as:

$$N_{\text{opt}} \propto C^{0.50}, \quad D_{\text{opt}} \propto C^{0.50}, \tag{5.29}$$

meaning that when the compute budget doubles, both the model size and the training data should increase by $\sqrt{2}$. The practical implication: a 70B-parameter model should be trained on approximately 1.4 trillion tokens for compute-optimal performance.

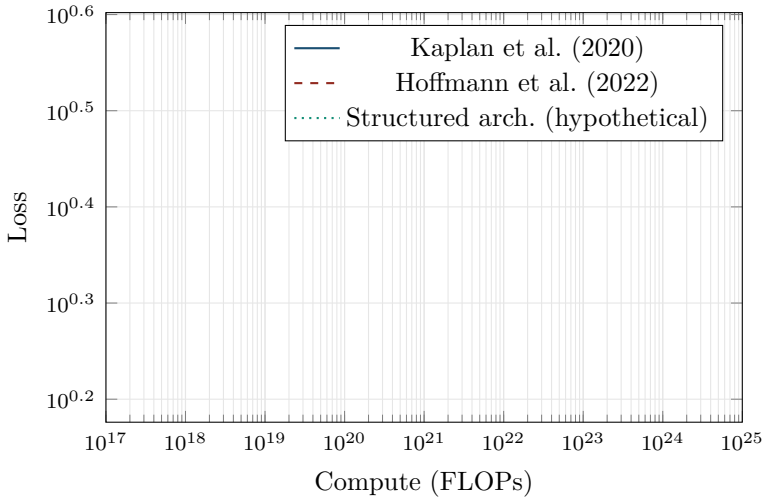


Figure 5.4: Scaling laws for language models. Loss decreases as a power law in compute. The Chinchilla-optimal curve achieves lower loss than the Kaplan curve at each compute level by better balancing parameters and data. The dotted line illustrates the hypothesis that structured architectures could achieve steeper scaling (higher exponents), reaching lower loss at the same compute budget.

Scaling Law Implications for DOGMA

Scaling laws describe the behavior of architecturally uniform transformers trained on unstructured token streams. A key research question for DOGMA is whether its structured genomic organization can achieve *better scaling exponents* than the vanilla transformer. If typed regions, regulatory control, and evolutionary optimization reduce the effective complexity of the learning problem, then DOGMA could achieve lower loss at the same compute budget—effectively shifting the scaling curves downward. Chapter 7 formalizes this hypothesis.

5.8 Emergent Abilities of Large Language Models

5.8.1 In-Context Learning

One of the most striking properties of large language models is *in-context learning* (ICL): the ability to learn new tasks from a few examples provided in the prompt, without any gradient updates [Brown et al., 2020]. Given k input-output examples $(x_1, y_1), \ldots, (x_k, y_k)$ followed by a new input x_{k+1}, the model generates y_{k+1} by conditioning on the entire prompt:

$$y_{k+1} \sim p_\theta(\cdot \mid x_1, y_1, \ldots, x_k, y_k, x_{k+1}). \tag{5.30}$$

ICL is remarkable because it represents a form of *meta-learning*: the model has learned, during pretraining, a general-purpose algorithm for adapting to new tasks specified by examples. The mechanism by which ICL works remains an active area of research; hypotheses include implicit Bayesian inference, mesa-optimization, and gradient descent in the forward pass [Akyürek et al., 2023].

Example 5.5 (In-Context Learning in Practice)**.** Consider prompting a language model with:

```
English: cat → French: chat
English: dog → French: chien
English: bird → French:
```

Without any training on translation, the model continues with "`oiseau`" by recognizing the pattern in the examples. The model is not recalling a memorized translation—it is performing an implicit inference about the task structure from the provided examples, then applying that inferred structure to the new input.

5.8.2 Few-Shot, One-Shot, and Zero-Shot Learning

Brown et al. [2020] systematically evaluated GPT-3 across dozens of NLP benchmarks in three regimes:

- **Zero-shot:** Only a task description is provided; no examples.
- **One-shot:** A single input-output example is provided.
- **Few-shot:** A small number (typically 2–64) of examples are provided.

Performance improved consistently from zero-shot to few-shot, and larger models showed greater improvement from additional examples. This suggests that scale enables more effective utilization of in-context information.

5.8.3 Chain-of-Thought Reasoning

Wei et al. [2022a] demonstrated that prompting large language models to produce intermediate reasoning steps—a technique called *chain-of-thought* (CoT) prompting—dramatically improves performance on tasks requiring multi-step reasoning:

$$p_\theta(y \mid x) \approx \sum_r p_\theta(y \mid r, x)\, p_\theta(r \mid x), \tag{5.31}$$

where r denotes the chain-of-thought reasoning trace. The reasoning trace acts as a "scratchpad" that decomposes complex problems into manageable steps. This capability appears to emerge sharply above certain model sizes, suggesting that it is a threshold phenomenon rather than a gradual improvement.

Example 5.6 (Chain-of-Thought Prompting)**. Without CoT:**

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can
has 3 balls. How many tennis balls does he have now?
A: 11
```

With CoT:

```
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can
has 3 balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 2 × 3 =
6 tennis balls. 5 + 6 = 11. The answer is 11.
```

The explicit reasoning steps reduce errors on arithmetic and multi-step problems because each intermediate result is generated and available as context for subsequent steps, rather than requiring the model to perform all computations implicitly within its hidden states.

5.8.4 Instruction Following

Large models trained with instruction tuning develop the ability to follow novel instructions they have never seen during fine-tuning. Given a natural language description of a task, the model performs the task without any examples. This ability is qualitatively different from few-shot learning: instead of inferring the task from input-output pairs, the model interprets a procedural description and executes it. This "instruction following" ability is the foundation of modern AI assistants.

Emergent Abilities

An ability is *emergent* if it is absent or near-random in smaller models but appears abruptly once the model exceeds a critical size [Wei et al., 2022a]. Examples include:

- Multi-digit arithmetic (emerges at ∼100B parameters)
- Logical deduction over multiple premises
- Code generation from natural language specifications
- Multi-step mathematical reasoning with CoT

Whether these abilities are truly emergent (discontinuous) or merely appear so due to nonlinear evaluation metrics remains debated [Schaeffer et al., 2023a].

5.9 Instruction Tuning and RLHF

Pretraining on next-token prediction produces a capable but *unaligned* model: it can complete any text, but it does not reliably follow instructions, produce helpful answers, or avoid harmful outputs. Closing this gap requires *alignment* techniques that shape model behavior to match human intent.

5.9.1 Supervised Instruction Tuning

The first alignment step is supervised fine-tuning (SFT) on a dataset of high-quality (instruction, response) pairs. This teaches the model to interpret prompts as instructions and produce appropriately structured responses, rather than merely continuing text in the style of the pretraining corpus.

The SFT dataset typically contains tens of thousands to hundreds of thousands of examples spanning diverse task categories: question answering, summarization, code generation, creative writing, and mathematical reasoning. Quality matters more than quantity: a small dataset of expert-written responses often outperforms a large dataset of lower-quality examples.

5.9.2 Reinforcement Learning from Human Feedback

Ouyang et al. [2022] introduced the RLHF pipeline, which further aligns the model through three stages:

1. **Reward model training:** Human annotators rank model outputs by quality. These rankings train a reward model $R_\phi(x, y)$ that predicts human preference scores.

2. **Policy optimization:** The language model ("policy") π_θ is optimized to maximize the reward while staying close to the SFT model via a KL-divergence penalty:

$$\max_\theta \ \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\big[R_\phi(x,y) - \beta\, D_{\mathrm{KL}}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\mathrm{SFT}}(\cdot \mid x))\big]. \tag{5.32}$$

3. **Iterative refinement:** The process is repeated, collecting new human comparisons on the updated model's outputs.

The KL penalty in Equation 5.32 is critical: without it, the model would collapse to degenerate outputs that exploit the reward model's imperfections (reward hacking).

RLHF and Fitness Landscapes

RLHF can be viewed as artificial evolution with human-guided fitness evaluation. The reward model approximates a fitness landscape shaped by human preferences, and policy optimization navigates that landscape. However, the feedback is *evaluative* (ranking outputs) rather than *structural* (modifying the genome). DOGMA's evolutionary training (Chapter 12) operates at the structural level, mutating the prompt genome itself rather than adjusting a reward signal—a deeper form of adaptation that mirrors biological evolution more closely.

5.10 Key Models: A Comparative View

The LLM landscape has diversified considerably since GPT-1. This section provides a comparative overview of the major model families.

5.10.1 GPT Family (OpenAI)

The GPT family established the decoder-only autoregressive paradigm:

Model	Parameters	Training tokens	Notable capability
GPT-1 (2018)	117M	∼5B	Transfer learning via fine-tuning
GPT-2 (2019)	1.5B	40B	Zero-shot task transfer
GPT-3 (2020)	175B	300B	Few-shot in-context learning
GPT-4 (2023)	∼1.8T*	∼13T*	Multi-modal reasoning

*Estimated; OpenAI has not published official figures for GPT-4.

5.10.2 BERT and Encoder-Only Models

BERT [Devlin et al., 2019] introduced a fundamentally different approach: *bidirectional* pre-training using masked language modeling. Instead of predicting the next token (left-to-right), BERT masks random tokens and trains the model to predict them from the full bidirectional context:

$$\mathcal{L}_{\mathrm{BERT}} = -\sum_{i\in\mathcal{M}} \log p_\theta(x_i \mid \mathbf{x}_{\setminus\mathcal{M}}). \tag{5.33}$$

BERT's bidirectional attention makes it better suited for *understanding* tasks (classification, question answering, named entity recognition) but unsuitable for *generation* tasks, since it cannot generate text autoregressively.

5.10.3 LLaMA and Open-Source Models

LLaMA [Touvron et al., 2023] demonstrated that high-quality, relatively small models trained on *more data* can match or exceed larger models. Key innovations:

- **RMSNorm** instead of LayerNorm (faster, equally effective).
- **SwiGLU activation** in the FFN (replacing ReLU with a gated variant).
- **RoPE** for positional encoding (enabling length generalization).
- **Grouped Query Attention (GQA):** Sharing key/value heads across query heads to reduce memory during inference.

5.10.4 Comparison Table

Table 5.1: Comparison of major LLM architectures.

Model	Type	Attention	Pos. Enc.	Norm	Best suited for
GPT-3/4	Decoder	Causal	Learned	Pre-LN	Text generation, reasoning
BERT	Encoder	Bidir.	Learned	Post-LN	Classification, NLU
T5	Enc-Dec	Both	Relative	Pre-LN	Seq-to-seq tasks
LLaMA	Decoder	Causal + GQA	RoPE	RMSNorm	Open-source generation

5.11 Limitations of the Transformer Paradigm

The transformer's success is undeniable. Yet success should not obscure structural limitations that constrain what the paradigm can ultimately achieve. This section identifies five fundamental limitations that motivate the DOGMA architecture.

5.11.1 Quadratic Attention Cost

Self-attention requires $O(T^2)$ time and memory in the sequence length T. Various approximations exist—sparse attention, linear attention, FlashAttention's IO-aware tiling [Dao et al., 2022]—but they either sacrifice expressiveness or provide constant-factor improvements without changing the asymptotic scaling. For genomic sequences that may span millions of bases, quadratic attention is impractical.

Beyond Efficient Attention

The fundamental issue is not merely computational cost: it is that *dense pairwise interaction between all tokens is the wrong inductive bias for structured sequences.* In a genome, a promoter interacts with its target gene, not with every other nucleotide. In a DOGMA prompt genome, the system instructions should regulate (not attend to) every output token. Typed regions with selective interaction patterns—DOGMA's approach—are more biologically plausible and more computationally efficient than sparse approximations to dense attention.

5.11.2 No Persistent State

A transformer has two forms of "memory": its frozen parameters (encoding the training distribution) and its context window (a bounded, ephemeral buffer). There is no intermediate state that persists across forward passes, accumulates experience, and can be selectively modified. Every new prompt starts from the same parameter snapshot.

In contrast, biological organisms maintain a *genome* that persists across the organism's lifetime, is read selectively in response to environmental signals, and can be modified through somatic mutation and epigenetic regulation. DOGMA's persistent genomic state fills this gap.

5.11.3 No Native Typed Regions

In a standard transformer, all tokens inhabit the same representational space and are processed by the same attention mechanism. System prompts, user queries, retrieved context, chain-of-thought reasoning, and generated output are all undifferentiated tokens. Any distinction between these roles must be learned implicitly from formatting conventions—a fragile and unverifiable approach.

DOGMA introduces *typed genomic bases* organized into *typed regions* (promoter, coding, regulatory, structural), each with distinct interaction semantics, regulatory weights, and compatibility constraints. This is analogous to how a genome contains structurally and functionally distinct regions (exons, introns, promoters, enhancers) rather than a uniform sequence of nucleotides.

5.11.4 Output-First Bias

The next-token prediction objective directly optimizes the quality of generated text. There is no explicit intermediate representation of the model's "plan" or "intent" before token generation begins. The model must simultaneously decide *what to do* and *how to say it* within a single left-to-right pass.

DOGMA separates these concerns across distinct stages: regulation determines *which* genomic regions are active, synthesis *assembles* the relevant information, transcription *reads* it into intermediate representations, expression *folds* those representations into structured behavior, and only then does realization *render* the output. This multi-stage pipeline mirrors the Central Dogma of molecular biology (Chapter 1).

5.11.5 No Evolutionary Self-Modification

Once a transformer is trained, its parameters are fixed. Adapting to new tasks requires explicit fine-tuning, prompt engineering, or retrieval augmentation—all external interventions. The model cannot modify its own computational structure in response to experience.

DOGMA's evolutionary training loop (Chapter 12) enables the system to mutate its own prompt genome, evaluate the fitness of mutations, and select beneficial changes—implementing a continuous evolutionary process analogous to biological adaptation.

Table 5.2: Structural comparison: transformers vs. biological systems. Each limitation of the transformer paradigm corresponds to a capability that biological systems achieve naturally and that DOGMA aims to capture computationally.

Capability	Transformer	Biology	DOGMA
Long-range interaction	$O(T^2)$ attention	Chromatin loops, enhancers	Typed region interactions
Persistent memory	Fixed params + ephemeral context	Genome + epigenome	Persistent genomic state
Structured regions	Flat token stream	Exons, introns, promoters	Typed genomic bases
Conditional computation	All params active	Cell-type-specific gene regulation	Regulatory activation
Self-modification	Requires external fine-tuning	Mutation, epigenetic changes	Evolutionary training loop
Multi-stage processing	Single left-to-right pass	DNA → RNA → Protein	6-stage pipeline

5.11.6 Summary: What Transformers Cannot Do That Biology Can

The Transformer as a Building Block

DOGMA does not *replace* the transformer; it *reorganizes computation around* it. The attention mechanism remains a powerful tool for computing context-dependent representations. What DOGMA changes is the *organization* of the input—replacing undifferentiated token streams with typed, regulated, structured genomic sequences—and the *optimization*—replacing one-shot training with continuous evolutionary adaptation. The transformer becomes a subsystem within a biologically inspired computational architecture.

5.12 Exercises

5.1. [**Fundamentals**] Derive the gradient of the scaled dot-product attention output (Equation 5.2) with respect to $\mathbf{Q}$. Show how the softmax Jacobian appears in the result and explain why the scaling factor $1/\sqrt{d_k}$ mitigates the vanishing gradient problem.

5.2. [**Worked Example**] Consider a 4-token sequence with $d_k = 2$. Let the query and key matrices be:

$$\mathbf{Q} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{K} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{5.34}$$

(a) Compute the raw attention scores $\mathbf{QK}^\top$. (b) Apply the scaling factor $1/\sqrt{d_k}$. (c) Apply a causal mask and compute the softmax. (d) Interpret the attention pattern: which tokens attend most strongly to which other tokens, and why?

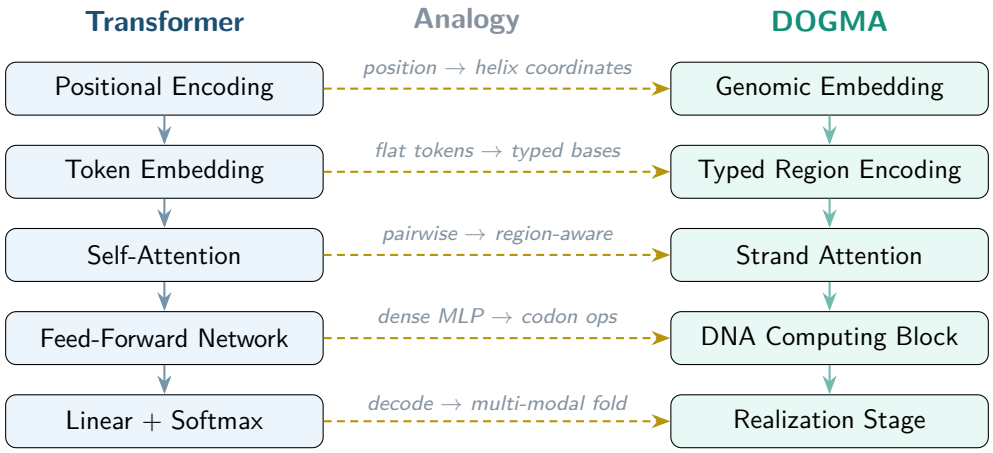


Figure 5.5: Side-by-side comparison of the Transformer architecture and the DOGMA architecture, highlighting the biological analogues. Each Transformer component has a DOGMA counterpart that adds biological structure: positional encoding becomes genomic (helix-aware) embedding, self-attention becomes region-typed strand attention, and the feed-forward network becomes a codon-programmed DNA computing block.

5.3. **[Coding]** Implement a single-head causal attention layer from scratch (NumPy only). Verify that your implementation produces the same output as the equivalent PyTorch `nn.MultiheadAttention` layer with `num_heads=1` and an appropriate attention mask.

5.4. **[Analysis]** Consider a sequence of length $T = 4096$ processed by a 12-layer transformer with $d = 768$ and $h = 12$ attention heads. Calculate: (a) the total number of entries in all attention matrices across all layers and heads, (b) the FLOPs required for the attention computation alone, and (c) the ratio of attention FLOPs to FFN FLOPs.

5.5. **[Theory]** The causal mask ensures that position i cannot attend to position $j > i$. Prove that this constraint is sufficient to make the autoregressive factorization in Equation 5.21 valid—i.e., that the representation at position t depends only on tokens $x_1, \ldots, x_t$ and not on any future tokens.

5.6. **[Positional Encoding]** Compute the sinusoidal positional encodings for positions $t = 0, 1, 2, 3$ with $d = 8$. (a) Verify that the dot product $\text{PE}(t) \cdot \text{PE}(t + k)$ depends primarily on k and not on t. (b) Explain why this property is useful for learning relative position relationships.

5.7. **[Scaling Laws]** A research lab has a fixed compute budget of $C = 10^{22}$ FLOPs. Using the Chinchilla scaling law (Equation 5.29) and the approximation $C \approx 6ND$, compute: (a) the optimal model size N_{opt}, (b) the optimal number of training tokens D_{opt}, and (c) compare with the actual GPT-3 allocation (175B parameters, 300B tokens) and discuss whether GPT-3 was undertrained.

5.8. **[Design]** Propose a "biologically inspired attention" mechanism that restricts interaction patterns based on typed regions rather than using dense pairwise attention. Formally specify (a) the type system for tokens, (b) the interaction rules between types, and (c) the resulting computational complexity. Compare with DOGMA's regulation stage in Chapter 7.

5.9. **[Emergent Abilities]** Design an experiment to test whether a specific ability (e.g., multi-digit addition) is truly emergent (discontinuous in model scale) or merely appears so due to

the evaluation metric. Propose two evaluation metrics—one binary (exact match) and one continuous (per-digit accuracy)—and predict what each would show as a function of model size.

5.10. [**Research**] Read Kaplan et al. [2020a] and Hoffmann et al. [2022a]. Both papers derive compute-optimal training strategies but reach different conclusions about the optimal parameter-to-data ratio. Explain the source of the discrepancy and discuss which finding is more relevant for DOGMA-style architectures with structured inputs.

5.13 Key Takeaways

Key Takeaways

- Self-attention computes context-dependent representations through a four-step process: project to Q/K/V, compute scaled dot-product scores, apply softmax normalization, and aggregate values. The $O(T^2)$ cost is a fundamental bottleneck for long sequences.
- Multi-head attention runs h parallel attention computations in lower-dimensional subspaces, allowing the model to attend to different relationship types simultaneously. Each head uses $d_k = d/h$ dimensions, preserving the total parameter count.
- The transformer block combines multi-head attention, feedforward networks, residual connections, and layer normalization into a powerful and parallelizable building block, repeated L times to form deep networks.
- Positional encodings inject sequence order into the permutation-equivariant attention mechanism. Sinusoidal encodings enable relative position learning; RoPE provides superior length generalization.
- The next-token prediction objective (cross-entropy loss) is a simple but powerful training signal that, when applied at scale, produces models with remarkable capabilities.
- Scaling laws [Kaplan et al., 2020a] reveal smooth power-law relationships between compute, data, parameters, and loss. Chinchilla [Hoffmann et al., 2022a] shows that compute-optimal training requires balancing model size with data size.
- Emergent abilities—in-context learning, chain-of-thought reasoning, few-shot generalization, instruction following—appear at sufficient scale, but the mechanisms remain incompletely understood [Brown et al., 2020, Wei et al., 2022a].
- RLHF [Ouyang et al., 2022] aligns pretrained models with human intent through reward modeling and policy optimization—an evaluative feedback process analogous to (but shallower than) biological fitness evaluation.
- GPT (decoder-only, autoregressive), BERT (encoder-only, bidirectional), and LLaMA (decoder-only, open-source, with modern optimizations like RoPE and GQA) represent the three major architectural paradigms for LLMs.
- The transformer paradigm's structural limitations—quadratic attention, no persistent state, no typed regions, output-first bias, no self-modification—motivate DOGMA's biologically inspired reorganization of computation.

Suggested Reading

- Vaswani et al. [2017a]: The original transformer paper—required reading.
- Radford et al. [2019]: GPT-2 and the case for large-scale language modeling.
- Brown et al. [2020]: GPT-3 and in-context learning at scale.

- Devlin et al. [2019]: BERT and bidirectional pre-training.
- Touvron et al. [2023]: LLaMA and the power of training smaller models on more data.
- Kaplan et al. [2020a]: Neural scaling laws—empirical foundations for the scaling paradigm.
- Hoffmann et al. [2022a]: Chinchilla—compute-optimal training and the case for more data.
- Ouyang et al. [2022]: InstructGPT and RLHF—aligning LLMs with human preferences.
- Dao et al. [2022]: FlashAttention—IO-aware exact attention for efficient transformers.
- Su et al. [2024]: RoPE—rotary position embeddings for length generalization.

Chapter 6

Biological Foundation Models

Biology is the most powerful technology ever created. DNA is software, protein is hardware, and cells are factories.

— Adapted from Arvind Gupta

In the previous chapters we learned how neural networks work (Chapter 4) and how transformers power large language models (Chapter 5). Those tools were built for human language—English sentences, code, and web text. But in the last five years, researchers have taken the *same ideas* and pointed them at biology: at protein sequences, DNA sequences, and gene expression data. The results have been nothing short of revolutionary.

Protein language models now predict how a chain of amino acids will fold into a three-dimensional shape. Diffusion models design entirely new proteins that have never existed in nature. DNA foundation models read hundreds of thousands of nucleotide letters and predict which genes will be turned on or off in a given cell type. These are not incremental improvements—they represent a qualitative shift in our ability to understand and engineer living systems.

This chapter is your guided tour through the landscape of biological foundation models. We will start from first principles: what exactly is a "foundation model," and why does the concept matter? Then we will walk through the major models one by one—AlphaFold, ESM-2, DNABERT, Nucleotide Transformer, Evo, and Enformer—explaining each at a level of detail that lets you understand *how* they work, not just *what* they do. Finally, we will see how DOGMA synthesizes lessons from all of these models into a unified architecture that goes beyond any single one.

6.1 What Is a Foundation Model?

Before we dive into specific biological models, let us be precise about what a **foundation model** is, because the term is used frequently but not always carefully.

6.1.1 Definition and Core Idea

A foundation model is a large model trained on broad, general-purpose data using self-supervised learning, and then *adapted* (fine-tuned) for many different downstream tasks. The term was coined by the Stanford HAI group in 2021 to capture a specific pattern that had emerged across AI:

1. **Pre-train** on a massive, unlabeled dataset using a self-supervised objective (e.g., predict masked tokens, predict the next word).

2. **Transfer** the learned representations to specific tasks by fine-tuning on smaller labeled datasets or by prompting.

The key insight is that step 1 produces representations that are *general*: they capture the statistical structure of the domain so thoroughly that they are useful for tasks the model was never explicitly trained on.

The Foundation Model Paradigm

A foundation model invests massive computation upfront to learn general-purpose representations, then amortizes that cost across many downstream tasks. This is economical because the cost of pre-training is paid once, while the benefits are reaped repeatedly. GPT-4 is a foundation model for language. AlphaFold is a foundation model for protein structure. ESM-2 is a foundation model for protein sequences.

6.1.2 An Analogy: The Medical Student

If you find the concept abstract, consider this analogy. A medical student spends four years studying anatomy, biochemistry, physiology, pharmacology, and pathology. During those four years, the student is not training for any *specific* patient. Instead, the student is building a broad foundation of knowledge about how the human body works. This is **pre-training**.

After medical school, the student enters a residency in, say, cardiology. During residency, the student applies their broad foundation to the specific domain of heart disease, refining their skills on cardiac patients. This is **fine-tuning**.

The foundation—four years of general medical education—makes the residency far more efficient than it would be if the student had tried to learn cardiology from scratch. Moreover, the same foundation supports residencies in *any* specialty: neurology, oncology, surgery. One pre-training phase, many downstream specializations.

Foundation models work the same way. ESM-2 "studies" millions of protein sequences (pre-training), learning general patterns about amino acid co-occurrence, conservation, and structure. Then it can be fine-tuned for specific tasks: predicting whether a mutation is harmful, classifying protein function, or estimating thermostability. The foundation makes all of these tasks easier.

6.1.3 Why Biology Is Special

Foundation models for biology differ from foundation models for language in one crucial way: **the training data was curated by evolution, not by humans**.

When we train GPT-4 on internet text, the training data reflects human writing conventions, cultural biases, and arbitrary stylistic choices. When we train ESM-2 on protein sequences, the training data reflects 3.5 billion years of natural selection. Every protein in the database exists because it *works*—it folds, it functions, it contributes to the survival of the organism that encodes it. Proteins that fail to fold are eliminated by evolution. This means that the statistical patterns in protein sequence databases are not arbitrary—they encode the laws of biophysics and the constraints of natural selection.

Evolution as Data Curator

The protein sequences in UniRef, the DNA sequences in GenBank, and the genomes in RefSeq are not random samples. They are the survivors of billions of years of testing against the fitness function of life itself. Training a model on these sequences is, in a precise sense, distilling the results of evolution's search into neural network weights.

6.1.4 The Biological Modalities

Foundation models have been built for each of the major biological data types (or **modalities**):

- **Protein sequences**: chains of amino acids (20-letter alphabet). Models: ESM-2, ProtTrans.
- **Protein structures**: 3D coordinates of atoms. Models: AlphaFold2, ESMFold.
- **DNA sequences**: chains of nucleotides (4-letter alphabet). Models: DNABERT, Nucleotide Transformer, Evo.
- **Gene expression**: quantitative measurements of how much mRNA each gene produces. Models: Enformer, scGPT.
- **Multi-modal**: models that bridge two or more modalities. DOGMA aims to be the ultimate multi-modal biological foundation model.

Each modality has its own challenges, its own data characteristics, and its own architectural requirements. The rest of this chapter walks through the most important models in each modality.

6.2 AlphaFold: Solving Protein Structure Prediction

6.2.1 The Protein Folding Problem

Every protein in your body starts as a linear chain of amino acids—a string of chemical "beads" produced by the ribosome. Within milliseconds, this chain folds into a specific three-dimensional shape, and that shape determines what the protein *does*. Hemoglobin's shape lets it carry oxygen. Antibodies' shapes let them recognize pathogens. Enzymes' shapes let them catalyze chemical reactions.

The **protein folding problem** asks: given the amino acid sequence, can we predict the 3D structure? This problem had been one of the grand challenges in biology for over 50 years. Experimental methods (X-ray crystallography, cryo-EM, NMR) can determine structures, but they are slow and expensive—sometimes taking years per protein. Computational prediction methods had made steady but incremental progress for decades.

Then, in 2020, AlphaFold2 [Jumper et al., 2021] effectively *solved* the problem. At the CASP14 competition (the biennial "Olympics" of protein structure prediction), AlphaFold2 achieved median GDT-TS scores above 90—accuracy comparable to experimental methods—on proteins where previous computational methods scored in the 40s and 50s. It was the most dramatic advance in the history of the field.

6.2.2 AlphaFold2: The Pipeline Step by Step

Let us walk through exactly how AlphaFold2 works, from input to output. Understanding this pipeline in detail is important because many of its architectural ideas reappear throughout biological AI.

Step 1: Input — The Amino Acid Sequence

AlphaFold2 takes a single amino acid sequence as input. For example, a small protein might be 150 amino acids long, written as a string like:

```
MKFLILLFNILCLFPVLAADNHGVSMNAS...
```

Each letter represents one of the 20 standard amino acids (M = methionine, K = lysine, F = phenylalanine, etc.).

Step 2: Multiple Sequence Alignment (MSA) Construction

Before the neural network sees the sequence, AlphaFold2 searches large protein databases (UniRef, BFD, MGnify) for **homologous** sequences—proteins from other organisms that are evolutionarily related to the query. These are aligned to produce a **Multiple Sequence Alignment (MSA)**: a matrix where rows are different homologous sequences and columns are aligned positions.

Why is the MSA so important? Because evolution provides a powerful signal about protein structure. If two positions in a protein tend to mutate together across evolution (they **co-evolve**), it often means those positions are physically close in 3D space. If position 23 is hydrophobic whenever position 87 is hydrophobic, and charged whenever position 87 is charged, then positions 23 and 87 are probably in contact. The MSA encodes millions of years of evolutionary experiments about which amino acid combinations work together.

Coevolution Reveals Contacts

When two residues are in physical contact within a folded protein, mutations at one position create selective pressure for compensatory mutations at the other. Over evolutionary time, this produces correlated patterns of variation in the MSA. AlphaFold2 exploits these correlations to infer which residues are spatially proximal—a crucial clue for predicting 3D structure.

Step 3: Initial Representations

AlphaFold2 creates two initial representations from the MSA:

- The **MSA representation**: a tensor of shape $(N_{\text{seq}} \times L \times d)$ where N_{seq} is the number of sequences, L is the sequence length, and d is the embedding dimension. Each entry represents one amino acid at one position in one sequence.

- The **pair representation**: a tensor of shape $(L \times L \times d_{\text{pair}})$. Entry (i, j) represents the relationship between residue i and residue j. Initially, this is populated with relative positional features and, optionally, template information from known structures of homologs.

Step 4: The Evoformer — Where the Magic Happens

The Evoformer is the heart of AlphaFold2. It is a specialized transformer block that jointly processes the MSA representation and the pair representation, with information flowing between them. AlphaFold2 stacks 48 Evoformer blocks.

Each Evoformer block performs four key operations:

1. **MSA row-wise self-attention**: Each sequence in the MSA attends to other positions *within the same sequence*. This captures intra-sequence patterns (e.g., local secondary structure preferences).

2. **MSA column-wise self-attention**: Each position in the alignment attends to the *same position across different sequences*. This captures evolutionary variation at each position.

3. **Pair update from MSA (outer product mean)**: Information from the MSA is projected into the pair representation. Specifically, the outer product of MSA features at positions i and j is averaged over sequences, producing a pairwise signal that captures coevolutionary correlations.

4. **Pair self-attention (triangular updates)**: The pair representation is updated using a novel mechanism called **triangular attention**. The key idea: if residue i is close to residue j, and residue j is close to residue k, then information about the (i, k) pair should be informed by residue j. This enforces a kind of "triangle inequality" in the predicted distance matrix.

The Evoformer's Dual-Track Design

The Evoformer maintains two parallel representations—one for evolutionary sequences (MSA), one for pairwise relationships—and exchanges information between them at every layer. This dual-track design is essential: evolutionary signals (from the MSA) inform structural predictions (in the pair representation), and structural predictions refine the interpretation of evolutionary signals. Neither track alone would suffice.

Step 5: The Structure Module

After 48 Evoformer blocks, the pair representation encodes a rich model of inter-residue relationships. The **Structure Module** converts this into explicit 3D coordinates.

The Structure Module operates on a set of **rigid frames**—one per residue. Each frame consists of a rotation matrix and a translation vector in 3D space, specifying the position and orientation of each residue's backbone. Starting from an initial set of frames (all at the origin), the module iteratively refines them using an **Invariant Point Attention** (IPA) mechanism that is equivariant under the SE(3) group of rotations and translations.

SE(3) Equivariance

A function f is *SE(3) equivariant* if rotating and translating the input produces correspondingly rotated and translated output:

$$f(R\mathbf{x} + \mathbf{t}) = Rf(\mathbf{x}) + \mathbf{t}, \tag{6.1}$$

where R is a rotation matrix and $\mathbf{t}$ is a translation vector. For protein structure prediction, this means the predicted structure does not depend on the arbitrary choice of coordinate system—a physical requirement, since proteins in solution have no preferred orientation.

Step 6: Recycling

AlphaFold2 runs the entire pipeline (Evoformer + Structure Module) three times, feeding the output of each pass back as input to the next. This **recycling** progressively refines the structure, analogous to how some iterative algorithms converge to a solution over multiple rounds.

Step 7: Output and Confidence

The final output is:

- 3D coordinates for all backbone atoms (N, Cα, C, O) and side-chain atoms.
- A per-residue confidence score called **pLDDT** (predicted Local Distance Difference Test), ranging from 0 to 100. Scores above 90 indicate very high confidence; scores below 50 suggest the region may be disordered or poorly predicted.
- A **Predicted Aligned Error (PAE)** matrix, indicating the confidence in the relative position of each pair of residues.

6.2.3 The AlphaFold2 Loss Function

The model is trained end-to-end with a composite loss:

$$\mathcal{L}_{\text{AF2}} = \mathcal{L}_{\text{FAPE}} + \lambda_{\text{dist}}\mathcal{L}_{\text{distogram}} + \lambda_{\text{conf}}\mathcal{L}_{\text{pLDDT}} + \lambda_{\text{MSA}}\mathcal{L}_{\text{MSA}}. \tag{6.2}$$

The **FAPE** (Frame Aligned Point Error) loss measures the error in predicted atomic positions after aligning local reference frames. It is more informative than simple RMSD because it penalizes errors in local geometry even when the global superposition is poor. The distogram loss trains auxiliary heads that predict binned inter-residue distances. The pLDDT loss trains the confidence predictor. The MSA loss ensures the model correctly predicts masked residues in the MSA.

6.2.4 AlphaFold2 Architecture Diagram

The following diagram illustrates the complete AlphaFold2 pipeline:

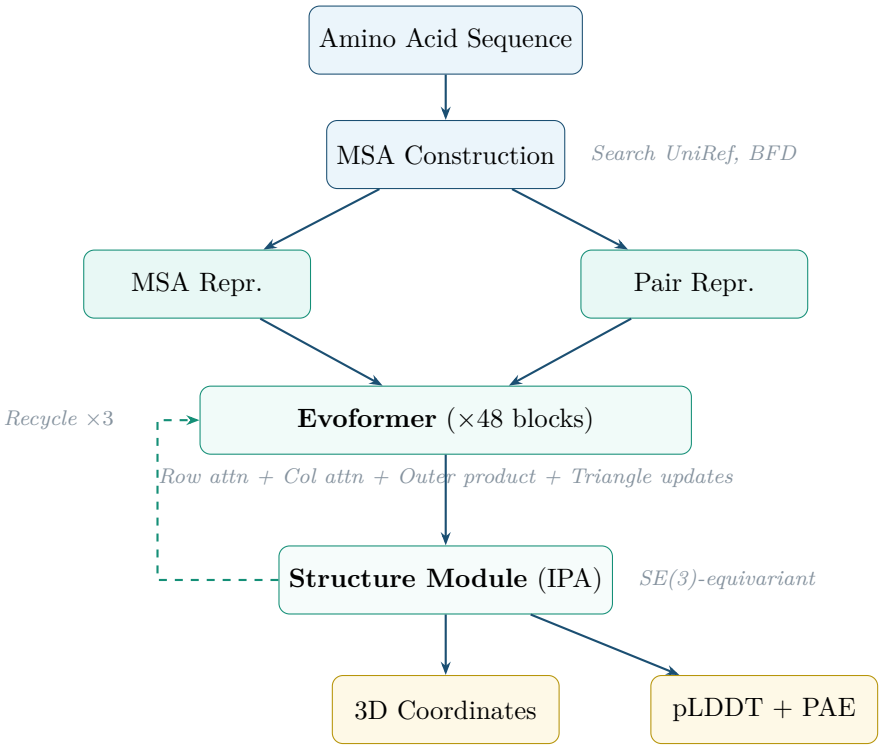


Figure 6.1: The AlphaFold2 pipeline. An amino acid sequence is used to construct a multiple sequence alignment (MSA). Two representations—MSA and pairwise—are jointly refined through 48 Evoformer blocks that exchange evolutionary and geometric information. The Structure Module converts pairwise features into 3D coordinates using SE(3)-equivariant attention. The entire pipeline is recycled three times.

6.2.5 Why AlphaFold2 Uses "Only" 93M Parameters

A common question: GPT-3 has 175 billion parameters, GPT-4 likely has over a trillion, and even ESM-2 has 15 billion. Why does AlphaFold2 need only 93 million?

The answer is **inductive bias**. AlphaFold2's architecture is exquisitely tailored to the physics of protein folding. SE(3) equivariance encodes rotational symmetry. Triangular attention encodes the triangle inequality of distances. The Evoformer's dual-track design encodes the relationship between evolutionary and structural information. These architectural choices dramatically reduce the hypothesis space—the model does not need to "discover" these constraints from data because they are built into the architecture.

This is a profound lesson: *the right architecture, informed by domain knowledge, can outperform brute-force scaling by orders of magnitude.* AlphaFold2 achieves superhuman accuracy on protein structure prediction with fewer parameters than a moderately sized language model.

AlphaFold2 in Practice

AlphaFold2 is available through several channels:

- The **AlphaFold Protein Structure Database** (alphafold.ebi.ac.uk) contains pre-computed structures for over 200 million proteins, covering nearly every known protein sequence.
- **ColabFold** provides a fast, accessible implementation that runs AlphaFold2 in a Google Colab notebook, using MMseqs2 for faster MSA construction.
- **AlphaFold3** (2024) extends the approach to predict structures of protein-nucleic acid complexes, small molecule ligands, and post-translational modifications.

6.3 ESM-2: The Language of Proteins

While AlphaFold2 focuses on predicting structure, the ESM (Evolutionary Scale Modeling) family of models takes a different approach: treat protein sequences as a *language* and learn their statistical structure using the same techniques that power large language models for text.

6.3.1 Proteins as Language

The analogy between proteins and language is surprisingly deep:

- Human language has an alphabet of ∼26 letters (in English). Proteins have an alphabet of 20 amino acids.
- Words have meaning that depends on context ("bank" means different things in different sentences). Amino acids have functional roles that depend on their neighbors in 3D space.
- Sentences have grammar—rules about which word orders are valid. Proteins have "grammar"—rules about which amino acid sequences can fold into stable structures.
- Languages evolve over time, with words changing meaning. Protein sequences evolve over time, with mutations accumulating under selective pressure.

If these parallels hold, then the same machine learning approach that works for language—train a large neural network to predict missing tokens in sequences—should work for proteins. This is exactly what ESM-2 does.

6.3.2 Training Objective: Masked Language Modeling

ESM-2 [Lin et al., 2023] is trained with a **masked language modeling (MLM)** objective, identical in spirit to BERT for text. Given a protein sequence $\mathbf{x} = (x_1, x_2, \ldots, x_L)$ where each x_i is one of 20 standard amino acids:

1. Randomly select 15% of positions and replace them with a `[MASK]` token.
2. Feed the masked sequence through the transformer.
3. Train the model to predict the original amino acid at each masked position.

Formally, the loss function is:

$$\mathcal{L}_{\text{MLM}} = -\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\sum_{i\in\mathcal{M}} \log p_\theta(x_i \mid \mathbf{x}_{\setminus\mathcal{M}})\right], \tag{6.3}$$

where $\mathcal{M}$ is the set of masked positions and $\mathbf{x}_{\setminus\mathcal{M}}$ is the sequence with those positions replaced by `[MASK]`.

Why does this simple objective work so well? Because predicting a masked amino acid requires understanding the *context*—the surrounding sequence that constrains what can appear at that position. If a position is buried in the protein's hydrophobic core, the model must predict a hydrophobic amino acid (like leucine, isoleucine, or valine). If a position is in an active site, the model must predict the specific catalytic residue. Over millions of training examples, the model learns a rich internal representation of protein structure and function.

6.3.3 Architecture and Scale

ESM-2 is a standard transformer encoder (the same architecture family as BERT), scaled up to several sizes:

Model	Layers	Hidden dim	Parameters
ESM-2 (8M)	6	320	8M
ESM-2 (35M)	12	480	35M
ESM-2 (150M)	30	640	150M
ESM-2 (650M)	33	1280	650M
ESM-2 (3B)	36	2560	3B
ESM-2 (15B)	48	5120	15B

The training data consists of protein sequences from **UniRef**, a comprehensive database of protein sequences clustered at various identity thresholds. The largest models are trained on UniRef50 (approximately 60 million cluster representatives) and UniRef90.

6.3.4 Emergent Properties: Structure from Sequence

The most remarkable finding about ESM-2 is what it learns *without being told to learn it.* The model is trained only to predict masked amino acids. It receives no information about 3D structure. Yet its internal representations encode rich structural information.

Emergent Structure in Protein Embeddings

ESM-2's attention maps can be used to extract **contact maps**—predictions of which residue pairs are physically close in 3D space. The model learns that residues distant in sequence but proximal in 3D space attend to each other. This emergent discovery of the folding code from sequence statistics alone demonstrates that protein structure is, in a meaningful sense, encoded in evolutionary sequence distributions.

Specifically, researchers showed that:

- ESM-2's attention heads specialize: some heads focus on local sequence patterns (secondary structure), others on long-range contacts (tertiary structure).

- The quality of emergent contact prediction *improves with model scale*—larger models learn more accurate structural representations.
- ESMFold, a structure prediction model built directly on ESM-2 embeddings (without any MSA), achieves accuracy competitive with AlphaFold2 for many proteins.

6.3.5 ESM-2 Architecture Overview

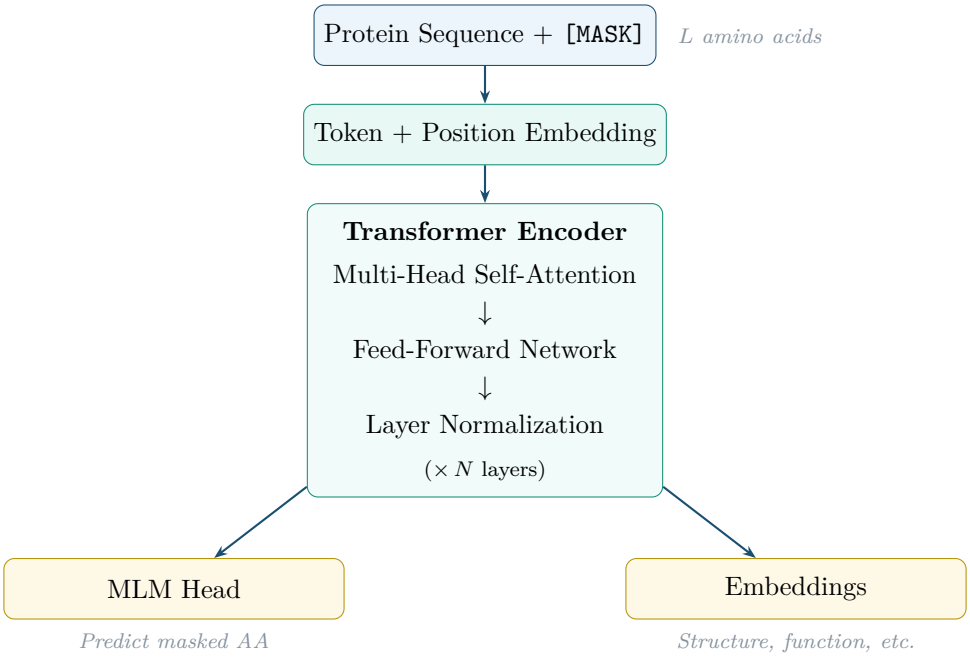


Figure 6.2: ESM-2 architecture overview. Protein sequences with masked positions are embedded and processed through N transformer encoder layers. The MLM head predicts masked amino acids (training). The learned embeddings (inference) encode structural and functional information useful for downstream tasks.

6.3.6 What ESM-2 Encodes

Through extensive probing studies, researchers have catalogued what ESM-2's representations capture:

1. **Secondary structure**: Whether each residue is in an alpha-helix, beta-sheet, or coil. Early layers capture this.
2. **Solvent accessibility**: Whether each residue is exposed to water (surface) or buried in the protein's interior (core). Middle layers encode this.
3. **Contact maps**: Which pairs of residues are physically close. Attention weights in later layers capture this.
4. **Functional sites**: Active sites, binding sites, and post-translational modification sites show distinctive patterns in the embedding space.

5. **Evolutionary conservation**: Highly conserved positions (functionally important) have different embedding distributions than variable positions.

6. **Protein family membership**: Proteins with similar functions cluster together in embedding space, even when their sequences are very different.

Evolutionary Plausibility as a Training Signal

In protein language models, the training distribution is curated by natural selection rather than by human annotators. The loss function in Equation 6.3 implicitly encodes biophysical constraints: folding stability, catalytic activity, binding specificity, and cellular fitness. This is a form of self-supervised learning where the "supervisor" is evolution itself. DOGMA generalizes this idea: the evolutionary training loop (Chapter 7) treats fitness-based selection as a first-class training mechanism alongside gradient descent.

Example 6.1 (Worked Example: ESM-2 Forward Pass on a Short Peptide)**.** Consider the 8-residue peptide `MKVILHGS`. We trace the ESM-2 forward pass step by step.

Step 1: Tokenization. Each amino acid maps to its vocabulary index:

M	K	V	I	L	H	G	S
10	8	18	7	9	6	5	15

Special tokens are prepended and appended: `[CLS] M K V I L H G S [EOS]`, giving token IDs $[0, 10, 8, 18, 7, 9, 6, 5, 15, 2]$.

Step 2: Masking (15%). With 8 amino acid tokens, we mask $\lceil 0.15 \times 8 \rceil = 1$ position. Suppose position 5 (`L`, leucine) is selected. The input becomes: `[CLS] M K V I [MASK] H G S [EOS]`.

Step 3: Embedding. Each token is converted to a d-dimensional vector via an embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d}$. For ESM-2 (8M), $d = 320$. The input becomes $\mathbf{X} \in \mathbb{R}^{10 \times 320}$. Learned positional embeddings $\mathbf{P} \in \mathbb{R}^{1024 \times 320}$ are added: $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{P}_{0:9}$.

Step 4: Transformer layers. The embedded sequence passes through 6 transformer layers (for ESM-2 8M). Each layer applies multi-head attention (20 heads, $d_k = 16$ per head) followed by a feed-forward network ($d_{\text{ff}} = 1280$):

$$\mathbf{H}^{(l)} = \text{FFN}\Big(\text{LayerNorm}\Big(\text{MHA}(\mathbf{H}^{(l-1)}) + \mathbf{H}^{(l-1)}\Big)\Big) + \text{LayerNorm}(\ldots).$$

Step 5: Prediction at masked position. The representation at position 5 (the masked `L`) is projected to a 33-dimensional logit vector (20 amino acids + special tokens) via a linear head:

$$\hat{\mathbf{y}}_5 = \mathbf{W}_{\text{head}}\, \mathbf{h}_5^{(6)} + \mathbf{b}_{\text{head}}, \quad \hat{\mathbf{y}}_5 \in \mathbb{R}^{33}.$$

The softmax gives probabilities over amino acids. If the model has learned that position 5 is buried in a hydrophobic core (inferred from surrounding V, I context), it will assign high probability to hydrophobic residues: $p(\texttt{L}) \approx 0.35$, $p(\texttt{I}) \approx 0.25$, $p(\texttt{V}) \approx 0.15$, $p(\texttt{F}) \approx 0.10$, etc. The cross-entropy loss penalizes the model for not assigning all probability mass to the true answer (`L`).

From ESM-2 to DOGMA: What Changes

DOGMA's approach differs from ESM-2 in three fundamental ways: (1) DOGMA uses byte-level tokenization (257 tokens) instead of amino acid tokens (33), operating at the nucleotide level where all biological information originates; (2) DOGMA replaces generic transformer layers with the biologically grounded 6-stage Central Dogma pipeline; and (3) DOGMA trains with evolutionary optimization in addition to gradient descent, allowing exploration of configurations that gradient descent alone cannot reach. The key insight: ESM-2 treats protein sequences as opaque strings; DOGMA treats genomic sequences as *programs* with structure, regulation, and expression.

6.4 DNABERT: Tokenizing and Learning from DNA

Proteins are not the only biological sequences amenable to language modeling. DNA—the molecule that encodes proteins and all other genetic information—can also be treated as a language, albeit one with a much smaller alphabet (4 letters: A, C, G, T) and much longer "sentences" (genes can span thousands to millions of bases).

6.4.1 The Challenge of DNA Tokenization

When we apply language models to text, tokenization is relatively straightforward: we split text into words or subwords (using BPE or similar). For proteins, single amino acids are natural tokens. But what is the right token for DNA?

Individual nucleotides (A, C, G, T) are too fine-grained: a vocabulary of only 4 tokens carries very little information per token, and meaningful biological units (codons, motifs, regulatory elements) span multiple nucleotides. On the other hand, very long tokens might miss important variation within them.

DNABERT [Ji et al., 2021b] adopts **k-mer tokenization**: the DNA sequence is converted to overlapping k-mers, typically with $k = 6$. For example:

`ATCGATCG` $\longrightarrow$ `ATCGAT` `TCGATC` `CGATCG`

This produces a vocabulary of 4^k possible tokens ($4^6 = 4{,}096$ for 6-mers). Each token captures local sequence context, and overlapping tokenization ensures that no information is lost at token boundaries.

Choosing k for DNA Tokenization

The choice of k involves a tradeoff:

- **Small k (e.g., 3)**: Small vocabulary ($4^3 = 64$), less information per token, but the model sees more tokens for the same sequence length. Codons are 3-mers, so $k = 3$ aligns with the genetic code.
- **Large k (e.g., 8)**: Large vocabulary ($4^8 = 65{,}536$), more information per token, but many tokens will be rare, making training harder. Some regulatory motifs are 6–12 bases long.
- **DNABERT's choice ($k = 6$)**: A compromise that captures most transcription factor binding motifs (which are typically 6–12 bp) while keeping vocabulary manageable.

DNABERT-2 later moved to Byte Pair Encoding (BPE), learning a data-driven tokenization that adapts to the actual frequency of subsequences in genomic data.

6.4.2 Pre-Training DNABERT

DNABERT uses the same masked language modeling approach as BERT and ESM-2:

1. Convert a DNA sequence to overlapping k-mers.
2. Mask 15% of the k-mer tokens.
3. Train the transformer to predict the masked tokens from context.

The training data consists of the human reference genome, providing a comprehensive view of human genomic sequence. The context window is 512 k-mer tokens, corresponding to approximately 512 base pairs of DNA.

6.4.3 What DNABERT Can Predict

After pre-training, DNABERT's representations can be fine-tuned for diverse genomic tasks:

- **Promoter prediction**: Identifying the regulatory regions upstream of genes that control transcription initiation.
- **Splice site prediction**: Finding the boundaries between exons and introns, where pre-mRNA is cut and rejoined.
- **Transcription factor binding site prediction**: Predicting where specific regulatory proteins bind to DNA.
- **Variant effect prediction**: Estimating whether a single-nucleotide variant (SNV) is likely to be deleterious.

On each of these tasks, DNABERT outperforms previous methods that used hand-engineered features, demonstrating that self-supervised pre-training captures meaningful genomic patterns.

6.5 Nucleotide Transformer: Scaling Across Species

DNABERT was trained on the human genome alone. The **Nucleotide Transformer** [Dalla-Torre et al., 2024b] asks: what happens if we train on genomes from across the tree of life?

6.5.1 Multi-Species Pre-Training

The Nucleotide Transformer is trained on reference genomes from **850 species**, spanning bacteria, archaea, plants, fungi, and animals. The total training dataset comprises approximately 3.2 billion nucleotides. The model is scaled to 2.5 billion parameters—an order of magnitude larger than DNABERT.

The tokenization uses non-overlapping 6-mers with a vocabulary of 4,096 tokens plus special tokens. The context length is 6,144 tokens, corresponding to approximately 6 kilobases of DNA—an order of magnitude longer than DNABERT.

6.5.2 Cross-Species Transfer Learning

The most striking finding from the Nucleotide Transformer is that **cross-species pre-training consistently outperforms single-species models**, even for human-specific tasks. A model trained on 850 genomes predicts human promoters, splice sites, and enhancers better than a model trained only on the human genome.

Why? Because the statistical regularities of genomic sequence are deeply conserved across evolution:

- **Codon usage** patterns are shared across species that use the same genetic code.
- **Splice site signals** (GT-AG at intron boundaries) are nearly universal in eukaryotes.
- **Regulatory motif grammar**—the way transcription factor binding sites are organized relative to genes—shows deep evolutionary conservation.
- **CpG islands**, GC-content variation, and other sequence composition features follow shared patterns.

Training on multi-species data forces the model to learn these *universal* patterns rather than memorizing human-specific sequences.

Evolutionary Breadth Beats Depth

The Nucleotide Transformer demonstrates a powerful principle: for learning genomic representations, evolutionary breadth (many species) is more valuable than depth (many copies of one genome). This is because conserved patterns that appear across diverse species are likely to be functionally important, while species-specific patterns may be idiosyncratic noise. DOGMA incorporates this principle by training on evolutionarily diverse data (Chapter 7).

6.5.3 Downstream Task Performance

The Nucleotide Transformer achieves state-of-the-art results on 18 genomic benchmark tasks, grouped into four categories:

1. **Regulatory element prediction**: Promoters, enhancers, open chromatin regions.
2. **Splicing**: Donor and acceptor splice sites, exon boundaries.
3. **Epigenetic marks**: Histone modifications (H3K4me3, H3K27ac, etc.) from sequence alone.
4. **Variant effects**: Predicting functional impact of single-nucleotide variants using ClinVar and similar databases.

On most of these tasks, the Nucleotide Transformer outperforms both DNABERT and task-specific models, demonstrating the value of scale and multi-species training.

6.6 Evo: Genome-Scale DNA Language Modeling

DNABERT sees 512 bases of context. The Nucleotide Transformer sees 6,000 bases. But real genomes are organized at scales of tens of thousands to millions of bases. A single operon in a bacterium might span 10 kilobases. A mammalian gene with its regulatory elements can span megabases. To model biology at the scale it actually operates, we need models with much longer context windows.

Evo [Nguyen et al., 2024] is a 7-billion-parameter DNA foundation model trained on 300 billion nucleotide tokens from prokaryotic and phage genomes, with a context window of 131,072 bases. It is, as of its publication, the longest-context genomic model ever trained.

6.6.1 The Context Length Challenge

Why is context length so hard? Standard transformer self-attention computes interactions between *all* pairs of tokens, which costs $O(T^2)$ in computation and memory, where T is the sequence length. For $T = 131{,}072$, this would require computing $131{,}072^2 \approx 17$ *billion* attention scores per layer—completely infeasible.

Evo solves this problem by replacing standard attention with an architecture called **StripedHyena**, which interleaves two types of layers:

1. **Gated convolutions**: Efficient local processing with linear complexity in sequence length.

2. **Data-controlled state space layers**: A variant of the Mamba/S4 family that maintains a compressed hidden state, propagated recurrently with input-dependent gating.

6.6.2 State Space Models for DNA

The state space layers in Evo follow the discrete-time state space model formulation:

$$\mathbf{h}_t = \overline{\mathbf{A}}\,\mathbf{h}_{t-1} + \overline{\mathbf{B}}\,x_t, \qquad \mathbf{y}_t = \mathbf{C}\,\mathbf{h}_t, \tag{6.4}$$

where $\mathbf{h}_t$ is a hidden state vector, x_t is the input at position t, and $\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C}$ are learned matrices. The key innovation is that $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ are **input-dependent** (data-controlled), allowing the model to selectively retain or forget information based on what it is reading. When the model encounters a promoter, it might "open the gate" to retain regulatory information. When reading through a non-functional intergenic region, it might "close the gate" and discard details.

This is analogous to chromatin state in biology: not all regions of the genome are equally "accessible" to the cell's transcription machinery at any given time. Open chromatin marks active regions; closed chromatin silences inactive ones. Evo's data-controlled state transitions implement a computational version of this selective accessibility.

Chromatin-Like Memory in Evo

Evo's state space layers implement selective memory: the model learns to retain information about functionally important regions (promoters, regulatory elements, start codons) while compressing or discarding less important sequence (repetitive DNA, intergenic filler). This is strikingly parallel to how chromatin modifications control genomic accessibility in biological cells.

6.6.3 Single-Nucleotide Resolution

Unlike models that downsample DNA (Enformer compresses 196 kb by a factor of 128), Evo operates at **single-nucleotide resolution** across its entire 131 kb context. Every individual base (A, C, G, or T) is a token. This is computationally demanding but biologically important: a single-nucleotide change can be the difference between a functional and a non-functional protein, between health and disease.

6.6.4 Genome-Scale Generation

Evo is trained with a **next-token prediction** (autoregressive) objective, which means it can also *generate* DNA sequences. Given a prompt (an initial DNA sequence), Evo can extend it to produce novel sequences of 100 kilobases or more.

Generated sequences exhibit remarkable biological realism:

- **Realistic codon usage**: The frequency of synonymous codons matches natural genomes.
- **Proper gene organization**: Generated sequences contain plausible open reading frames with start and stop codons.
- **Regulatory motif placement**: Promoter-like sequences appear upstream of predicted genes.
- **Operon structure**: Multi-gene clusters are organized coherently.

Example 6.2 (Worked Example: Evo Processing a DNA Promoter)**.** Consider a short DNA sequence containing a bacterial promoter: `TTGACA...TATAAT` (the -35 and -10 boxes). Let us trace how Evo's SSM state responds to this biologically meaningful motif, using a simplified 2-state model.

Each nucleotide is tokenized as a single character: A→0, C→1, G→2, T→3. Processing the sequence `T T G A C A`:

At positions 1–2 (`T T`): The SSM processes two thymines. The state transitions depend on the input:

$$x_1 = \bar{\mathbf{B}}(T) \cdot e_T, \quad x_2 = \bar{\mathbf{A}}(T) \cdot x_1 + \bar{\mathbf{B}}(T) \cdot e_T.$$

Because both inputs are the same (T), the model accumulates signal in the "T-responsive" state dimensions.

At position 3 (`G`): The input changes to G, producing different $\bar{\mathbf{B}}(G)$ and $\Delta(G)$. If the model has learned that `TTG` is the beginning of a -35 box, the selectivity mechanism will *increase* Δ (forget less) to retain the emerging promoter signal.

At position 6 (`A`): After processing `TTGACA`, the hidden state encodes a compressed representation of the complete -35 box. The selective gating has prevented this critical regulatory information from being overwritten.

The key biological insight: Evo learns that promoter motifs are "worth remembering" (large Δ, small $\bar{\mathbf{A}}$ decay) while intergenic sequence can be compressed (small Δ, strong decay). This data-dependent selection is the SSM analogue of chromatin accessibility: some genomic regions are "open" (retained in memory) while others are "closed" (compressed away).

Single-Nucleotide Resolution at Genome Scale

Evo operates at single-nucleotide resolution across 131 kb contexts. This is the computational equivalent of reading and writing DNA at the level of individual bases while maintaining coherent structure across entire gene clusters. The combination of fine-grained resolution with long-range context is precisely what DOGMA aims to achieve: individual genomic bases with region-level organizational structure (Chapter 7).

6.7 Enformer: From DNA Sequence to Gene Expression

All of the models we have discussed so far learn from sequence alone. But the ultimate readout of a genome is not the sequence itself—it is the **gene expression pattern**: which genes are turned on, in which cell types, and at what levels. **Enformer** [Avsec et al., 2021] bridges this gap, predicting quantitative gene expression directly from DNA sequence.

6.7.1 The Gene Expression Prediction Task

Enformer takes as input a **196,608 base pair** (approximately 196 kb) DNA sequence centered on a gene of interest. Its output is a vector of predicted expression values across:

- 5,313 genomic tracks for human (including RNA-seq, ATAC-seq, ChIP-seq, CAGE).
- 1,643 genomic tracks for mouse.

Each track corresponds to a different assay in a different cell type. For example, one track might represent H3K27ac ChIP-seq in liver cells (marking active enhancers in liver), while another represents CAGE-seq in brain cells (measuring mRNA start sites in brain). The model predicts all of these simultaneously from sequence alone.

6.7.2 Architecture: Convolution Then Attention

Enformer uses a hybrid architecture that combines the strengths of convolutional neural networks and transformers:

$$\hat{\mathbf{y}} = f_{\text{head}}\Big(f_{\text{transformer}}\big(f_{\text{conv}}(\mathbf{x}_{\text{DNA}})\big)\Big), \tag{6.5}$$

where the pipeline consists of three stages:

Stage 1: Convolutional Stem

The 196,608 bp input is first one-hot encoded (4 channels for A, C, G, T) and then passed through a series of convolutional blocks with residual connections and pooling. These convolutions serve two purposes:

- **Feature extraction**: Local motifs (transcription factor binding sites, splice signals, CpG dinucleotides) are detected by convolutional filters.

- **Downsampling**: The sequence is progressively reduced in length by a factor of 128, from 196,608 positions to 1,536 positions. Each of the 1,536 resulting "tokens" represents 128 bp of DNA and has a 1,536-dimensional feature vector.

Stage 2: Transformer

The 1,536 downsampled tokens are processed by 11 transformer layers with standard multi-head self-attention. At this compressed scale, the transformer can model interactions spanning the entire 196 kb input—including enhancer-promoter interactions that can span 100 kb or more.

Stage 3: Prediction Head

The transformer output is passed through a pointwise linear layer that predicts the expression value for each of the thousands of genomic tracks at each output position. The output resolution is 128 bp (matching the downsampling factor).

6.7.3 Long-Range Regulatory Interactions

The most important biological insight from Enformer is its ability to capture **enhancer–promoter interactions** across distances of up to 100 kilobases.

In eukaryotic gene regulation, a gene's expression is controlled not just by its promoter (the region immediately upstream of the gene) but also by **enhancers**—regulatory DNA elements that can be located tens or hundreds of kilobases away. Enhancers are activated by transcription factors that bind to specific sequence motifs, and they communicate with promoters through 3D chromatin looping.

Enformer Discovers Regulatory Grammar

Enformer's attention maps reveal that the model learns to focus on known regulatory elements—promoters, enhancers, CTCF binding sites (which organize chromatin loops), and insulator elements—even though it receives no explicit annotation of these elements during training. The model discovers the *regulatory grammar* of the genome from the data alone, much as ESM-2 discovers protein contact maps from sequence statistics.

6.7.4 Limitations

Despite its achievements, Enformer has important limitations that motivate the development of more powerful models:

- **Context length**: 196 kb captures most enhancer-promoter interactions but misses very long-range regulatory elements (some enhancers act from >1 Mb away).
- **Downsampling**: The 128 bp output resolution means single-nucleotide variants are not directly modeled at the output level.
- **Training data**: Enformer is trained on cell-type-specific experimental data, which limits its ability to generalize to unseen cell types or conditions.
- **Sequence only**: Enformer predicts expression from sequence but does not model how expression changes in response to perturbations (mutations, drug treatments).

6.8 Generative Protein Design: RFdiffusion

The models above are primarily about *understanding*: predicting structure from sequence (AlphaFold2), learning representations (ESM-2), or predicting gene expression (Enformer). But there is a second revolution happening in parallel: **generative protein design**—creating entirely new proteins that have never existed in nature.

6.8.1 From Prediction to Design

Structure prediction answers: "given a sequence, what structure does it adopt?" Protein design inverts this: "given a desired structure or function, what sequence achieves it?" This inversion is enormously valuable for medicine, industry, and basic science.

6.8.2 RFdiffusion

RFdiffusion [Watson et al., 2023] applies denoising diffusion to protein backbone generation. Starting from random noise in SE(3) space (random positions and orientations for each residue), the model iteratively denoises to produce physically plausible protein backbones:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I}), \tag{6.6}$$

where $\mathbf{x}_t$ represents noised backbone frames and the denoising network μ_θ is built on the RoseTTAFold architecture. The model can be conditioned on partial structures, binding motifs, or symmetry constraints.

Conditioning Strategies in RFdiffusion

RFdiffusion supports multiple conditioning modes:

- *Unconditional generation*: Novel folds sampled from the learned distribution.
- *Motif scaffolding*: Designing proteins that present a specific functional motif in the correct geometry.
- *Symmetric oligomers*: Generating multi-chain assemblies with specified point-group symmetry.
- *Binder design*: Creating proteins that bind to a specified target surface.

These conditioning mechanisms parallel DOGMA's regulatory control: different "contexts" (conditioning signals) activate different generative pathways from the same underlying model, just as different transcription factors activate different genes from the same genome.

6.8.3 The Inverse Folding Pipeline

RFdiffusion generates backbones (structures), but functional proteins require *sequences*. The complete design pipeline is:

1. **RFdiffusion**: Generate a novel protein backbone (3D structure).
2. **ProteinMPNN**: Design amino acid sequences that will fold into the generated backbone ("inverse folding"), maximizing $p(\mathbf{s} \mid \mathbf{x}_{\text{backbone}})$.

3. **AlphaFold2**: Validate that the designed sequences are predicted to fold into the intended structure.

4. **Experimental testing**: Synthesize the DNA encoding the designed protein, express it in cells, and test whether it actually folds and functions.

This computational-experimental loop—design *in silico*, validate *in vitro*—represents a paradigm shift. Unlike text generation, where quality is judged subjectively, protein design produces physical objects whose quality is judged by the laws of physics.

6.9 Comparison of Biological Foundation Models

Having surveyed the major biological foundation models individually, let us now compare them side by side. Table 6.1 summarizes the key properties of each model.

Table 6.1: Comprehensive comparison of major biological foundation models.

Model	**Modality**	**Params**	**Training Data**	**Context**	**Key Task / Innovation**
ESM-2	Protein seq.	15B	UniRef (65M seqs)	1,024 aa	Masked LM; emergent structure from evolution
AlphaFold2	Protein struct.	93M	PDB + UniRef	Full chain	SE(3)-equivariant Evoformer + structure module
RFdiffusion	Protein struct.	∼260M	PDB structures	Full chain	Denoising diffusion on SE(3) frames for design
DNABERT	DNA seq.	110M	Human genome	512 k-mers	k-mer tokenization + BERT-style MLM
Nucleotide Trans.	DNA seq.	2.5B	850 genomes	6 kb	Multi-species training; cross-species transfer
Evo	DNA seq.	7B	300B nt tokens (prokaryotes)	131 kb	StripedHyena SSM; genome-scale generation
Enformer	DNA → expr.	∼250M	CAGE, ATAC, ChIP-seq	196 kb	Conv + transformer for expression prediction

Several patterns emerge from this comparison:

1. **Modality specialization**: Each model targets a single biological modality. No model spans from DNA to protein to expression.

2. **Scale variation**: Parameter counts range from 93M (AlphaFold2) to 15B (ESM-2), with context lengths from 512 tokens (DNABERT) to 196 kb (Enformer). Architectural efficiency matters as much as raw scale.

3. **Training data diversity**: The most powerful models (Nucleotide Transformer, Evo) benefit from multi-species training, leveraging evolutionary diversity.

4. **Self-supervised learning**: Every model uses a self-supervised objective (masked prediction, next-token prediction, denoising). Labeled data is used only for fine-tuning or evaluation.

6.10 What Biology Teaches AI

The biological foundation models surveyed above are often presented as "applications of ML to biology." This framing, while not incorrect, obscures a deeper truth: biology is not just a domain for ML—it is a *source of architectural innovation*. The following principles, discovered or reinforced by biological foundation models, have implications far beyond biology.

6.10.1 Symmetry as Architectural Prior

AlphaFold2's SE(3) equivariance is not an arbitrary design choice—it encodes a fundamental symmetry of physics. Proteins in a test tube do not have preferred orientations; therefore, a structure prediction model should produce the same output regardless of the reference frame. More generally, identifying and encoding the *symmetries of the problem* into the architecture is one of the most powerful forms of inductive bias.

Symmetry-Aware Architecture Design

A model architecture is *symmetry-aware* if its computational structure respects the symmetry group G of the underlying problem. For structure prediction, $G = \mathrm{SE}(3)$. For sequence modeling, G includes translational equivariance (a motif has the same meaning regardless of its absolute position). For DOGMA, the relevant symmetries include: (1) permutation invariance across genomic regions that do not interact, (2) equivariance of regulatory logic under base-type relabeling within compatibility classes, and (3) conservation of functional semantics under evolutionary drift.

6.10.2 Multi-Scale Structure

Biological sequences exhibit structure at every scale: individual residues form secondary structures, secondary structures pack into domains, domains assemble into multi-domain proteins, and proteins organize into complexes. At the DNA level: nucleotides form codons, codons form genes, genes form operons, operons form chromosomal domains, and domains organize into chromosomes.

Enformer's hierarchical convolution-transformer architecture and Evo's state space layers both reflect this multi-scale organization. The lesson for AI architecture design is clear: systems that process hierarchically organized data should have *hierarchically organized computation*.

6.10.3 Evolutionary Conservation as a Learning Signal

The most striking feature of protein and DNA language models is that they learn biologically meaningful representations from *evolutionary statistics alone*. ESM-2 discovers protein contacts; Enformer discovers regulatory elements; the Nucleotide Transformer discovers conservation patterns. In each case, the training data is not labeled by humans—it is labeled by evolution.

Evolution as the Ultimate Self-Supervised Learner

Natural selection is, formally, an optimization process that evaluates candidate solutions (organisms) against a fitness function (survival and reproduction) and retains the best solutions (differential reproduction). The set of extant sequences is the output of this optimization process after billions of iterations. Training a language model on this dataset is, in a precise sense, distilling the results of evolution's search into neural network parameters. DOGMA's evolutionary training loop (Chapter 7) makes this connection explicit: it uses selection pressure as a training signal alongside gradient-based optimization.

6.10.4 The Generative Turn

The progression from ESM-2 (understanding) to RFdiffusion (design) mirrors the broader trajectory of AI from discriminative to generative models. But in biology, the generative turn carries special significance: generating a novel protein sequence is generating a *physical object* that can be synthesized, expressed, and tested. The feedback loop between computation and experiment—predict, design, synthesize, test, iterate—has no parallel in text-based AI. DOGMA's staged pipeline (regulate, transcribe, express, realize) is designed to support exactly this kind of grounded, multi-step generative process.

6.11 How DOGMA Differs: An Explicit Comparison

Now that we have surveyed the landscape of biological foundation models, let us be precise about how DOGMA differs from each of them. This is not a critique of existing models—each has achieved extraordinary things within its domain. Rather, it is an argument for why a unified approach is needed.

6.11.1 DOGMA vs. AlphaFold2

AlphaFold2 is a *structure prediction* model: it maps protein sequences to 3D coordinates. It does not generate new proteins, it does not model gene regulation, and it does not connect protein structure back to the DNA sequence that encodes it. DOGMA's genomic state contains the information from which proteins emerge through a multi-stage pipeline: DNA → regulation → transcription → translation → folding → function. Structure is one output among many, not the sole objective.

6.11.2 DOGMA vs. ESM-2

ESM-2 learns rich protein representations from evolutionary sequence data. But it operates on proteins in isolation—each protein is processed independently, with no model of how that protein was produced, how it interacts with other proteins, or how its expression is regulated. DOGMA treats each protein as part of a larger system: encoded in a genome, regulated by cellular context, and functioning within a network of interactions.

6.11.3 DOGMA vs. DNABERT / Nucleotide Transformer

DNABERT and the Nucleotide Transformer learn representations of DNA sequence, but they do not model what happens *after* the DNA is read: transcription, translation, protein folding, and downstream function. They predict genomic features (promoters, splice sites) but not the emergent

phenotype that arises from the coordinated action of all genes. DOGMA's pipeline follows the central dogma—from DNA through RNA to protein to function—modeling each stage explicitly.

6.11.4 DOGMA vs. Evo

Evo is the closest existing model to DOGMA in spirit: it operates at genome scale, models DNA at single-nucleotide resolution, and can generate novel genomic sequences. But Evo stops at DNA. It does not model transcription, translation, or protein function. It generates DNA sequences but does not predict what those sequences *do* at the phenotypic level. DOGMA extends the Evo paradigm downstream: from DNA generation through expression and realization.

6.11.5 DOGMA vs. Enformer

Enformer predicts gene expression from DNA sequence—a crucial link in the central dogma. But it predicts expression as a static quantity, without modeling the feedback loops that make gene regulation dynamic: proteins regulate the expression of other genes, which produce proteins that regulate still other genes. DOGMA models this regulatory feedback explicitly through its evolutionary training loop.

Table 6.2: Feature comparison: existing biological foundation models vs. DOGMA.

Capability	**AF2**	**ESM-2**	**DNABERT**	**Evo**	**Enformer**	**DOGMA**
DNA representation	–	–	✓	✓	✓	✓
Protein representation	✓	✓	–	–	–	✓
Gene expression pred.	–	–	–	–	✓	✓
Structure prediction	✓	partial	–	–	–	✓
Generative (design)	–	–	–	✓	–	✓
Multi-modal integration	–	–	–	–	–	✓
Regulatory feedback	–	–	–	–	–	✓
Evolutionary training	–	–	–	–	–	✓

6.12 The Missing Piece: Why Current Models Are Domain-Specific

6.12.1 The Fragmentation Problem

Consider the state of biological AI today. If you want to predict a protein's structure, you use AlphaFold2. If you want to learn protein representations, you use ESM-2. If you want to predict

gene expression, you use Enformer. If you want to generate DNA, you use Evo. If you want to design a new protein, you use RFdiffusion + ProteinMPNN.

Each of these models is excellent at its specific task. But none of them models the complete biological pipeline from DNA to function. They are fragments of a much larger picture, and they do not communicate with each other.

This fragmentation has practical consequences:

- **No feedback loops**: In real biology, proteins regulate gene expression, which changes protein production, which changes regulation. Current models cannot capture these feedback loops because they are disconnected.
- **No cross-modal reasoning**: A mutation in a DNA regulatory element affects gene expression, which affects protein levels, which affects cellular function. No current model can trace this causal chain end-to-end.
- **Redundant training**: Each model learns its own representations from scratch, even when the underlying biology is shared. ESM-2 and AlphaFold2 both learn about protein structure, but from different starting points and with no shared representations.

6.12.2 The Central Dogma as Unifying Principle

In biology, the central dogma provides a natural unifying framework: DNA $\rightarrow$ RNA $\rightarrow$ Protein. This is not just a slogan—it is the actual information flow that connects all biological modalities. Every protein was transcribed from an mRNA, which was transcribed from a DNA gene, which is regulated by other proteins and RNAs in a complex network.

DOGMA takes this biological principle and makes it an *architectural* principle. Instead of building separate models for DNA, RNA, and protein, DOGMA builds a single system that follows the information flow of the central dogma:

1. **Genome** (DNA): The stored program, analogous to a codebase.
2. **Regulation**: Context-dependent activation, analogous to conditional compilation.
3. **Transcription**: Converting selected genomic regions to working copies (RNA), analogous to reading source code.
4. **Translation/Expression**: Converting RNA instructions to functional outputs (proteins), analogous to compiling and running code.
5. **Realization**: The functional output in the real world, analogous to the running program's behavior.

DOGMA's Unification Thesis

Every existing biological foundation model captures one step of the central dogma. DOGMA captures them all in a single, end-to-end architecture. This is not merely an engineering convenience—it enables fundamentally new capabilities: reasoning about how DNA changes propagate through regulation, transcription, and expression to affect function; designing novel genomic programs that produce desired functional outputs; and modeling the evolutionary dynamics that shape all of these processes simultaneously.

6.12.3 Why Unification Is Hard

If unification is so valuable, why has no one done it before? Because the technical challenges are formidable:

- **Scale mismatch**: DNA sequences are millions of bases long, but proteins are typically hundreds of amino acids. A unified model must handle both scales.
- **Modality mismatch**: DNA is sequential (1D), protein structure is geometric (3D), gene expression is quantitative (continuous). A unified model must handle all data types.
- **Training data heterogeneity**: Protein structure data (PDB) contains ∼200,000 structures. Protein sequences (UniRef) contain ∼300 million. Genomes (RefSeq) contain thousands. The data volumes and characteristics are very different.
- **Loss function design**: How do you combine losses for sequence prediction, structure prediction, and expression prediction in a single training objective without one task dominating the others?

Chapter 7 describes how DOGMA addresses each of these challenges. The biological foundation models in this chapter provide the building blocks; DOGMA provides the architectural framework that integrates them.

6.13 The Convergence with DOGMA

The biological foundation models surveyed in this chapter are not isolated achievements. They form a converging body of evidence for three principles that DOGMA takes as axiomatic.

6.13.1 The Genome as Structured Program

Standard language models treat their input as a flat token sequence. Biological foundation models reveal that biological sequences are *programs*: they encode instructions for building structures (proteins), controlling expression (regulatory DNA), and orchestrating development (gene regulatory networks). DOGMA's genomic state is not a token buffer—it is a structured program with typed regions, regulatory logic, and compositional semantics, as formalized in Chapter 7.

6.13.2 Regulation as Conditional Computation

Enformer demonstrates that gene expression is determined not by the coding sequence alone but by the surrounding regulatory context: enhancers, silencers, insulators, and chromatin state. This is *conditional computation*—the same gene produces different amounts of protein in different cellular contexts. DOGMA's regulatory layer implements the same principle: context-dependent activation of genomic regions enables the system to exhibit different behaviors from the same underlying genome.

From Biological Regulation to Computational Regulation

In biology, a single genome encodes hundreds of distinct cell types through differential gene regulation. In DOGMA, a single genomic state encodes diverse task-specific behaviors through regulatory activation patterns. The parallel is not metaphorical: both systems use context signals to selectively activate subsets of a shared information store, achieving combinatorial expressiveness from a fixed substrate.

6.13.3 Evolution as Optimization

Protein language models succeed because evolution has pre-optimized their training data. AlphaFold2 succeeds because evolutionary covariation encodes structural constraints. Evo succeeds because evolutionary conservation patterns at genome scale reflect functional organization. In each case, evolution is not just background context—it is an active computational force that shapes the distribution models learn from.

DOGMA closes the loop: rather than merely learning from the products of evolution, it *incorporates evolution as an optimization mechanism*. The evolutionary training loop treats the genomic state as a population of candidate solutions, applies mutation and selection alongside gradient updates, and uses fitness-based criteria that go beyond differentiable loss functions. This synthesis of evolutionary and gradient-based optimization, informed by the lessons of biological foundation models, is DOGMA's central methodological contribution.

From Inspiration to Implementation

Translating biological principles into DOGMA requires careful formalization. Key correspondences:

- ESM-2's masked prediction $\longrightarrow$ DOGMA's genomic base prediction during synthesis.
- AlphaFold2's dual-track processing $\longrightarrow$ DOGMA's separation of genomic state and regulatory context.
- RFdiffusion's conditional generation $\longrightarrow$ DOGMA's context-conditioned transcription.
- Enformer's multi-scale architecture $\longrightarrow$ DOGMA's hierarchical region structure.
- Evo's long-range state space layers $\longrightarrow$ DOGMA's ParallelScan path for efficient genomic processing.

These are not analogies—they are design derivations. Each DOGMA component has a specific biological foundation model as its conceptual ancestor.

6.14 Exercises

6.1. **[Conceptual — Foundation Models]** In your own words, explain the difference between "pre-training" and "fine-tuning" in the foundation model paradigm. Use the medical student analogy from Section 6.1 to illustrate your answer. Why is pre-training more computationally expensive than fine-tuning?

6.2. **[Conceptual — AlphaFold2]** AlphaFold2 uses a Multiple Sequence Alignment (MSA) as input rather than just the query sequence alone. Explain *why* the MSA provides additional information beyond the query sequence. What biological phenomenon does the MSA capture, and how does the Evoformer exploit it?

6.3. **[Conceptual — ESM-2]** ESM-2 is trained only to predict masked amino acids, yet it learns to predict protein 3D contacts. Explain how this is possible. What property of evolutionary sequence data makes structure information "leak" into the masked language modeling objective?

6.4. **[Conceptual — DNA Tokenization]** Compare and contrast k-mer tokenization (DNABERT) with Byte Pair Encoding (DNABERT-2) for DNA sequences. What are the advantages and disadvantages of each approach? Consider vocabulary size, information per token, and biological interpretability.

6.5. **[Analysis — Parameter Efficiency]** AlphaFold2 uses 93M parameters to solve structure prediction; ESM-2 uses 15B parameters for protein language modeling. Why does AlphaFold2 need so many fewer parameters? Frame your answer in terms of inductive biases and the bias-variance tradeoff from Chapter 4.

6.6. **[Analysis — Context Length]** The context lengths of DNA foundation models have grown dramatically: DNABERT (512 bp), Nucleotide Transformer (6 kb), Evo (131 kb), Enformer (196 kb). For each jump in context length, identify one biological phenomenon that the longer context enables the model to capture. Why is quadratic attention prohibitive at these scales?

6.7. **[Analysis — Cross-Species Training]** The Nucleotide Transformer achieves better performance on human genomic tasks when trained on 850 genomes than when trained on the human genome alone. Propose a hypothesis for why this occurs, and suggest an experiment to test your hypothesis.

6.8. **[Coding]** Implement a simple k-mer language model for DNA sequences. Train it on a small bacterial genome (e.g., *E. coli*) using next-k-mer prediction. Measure perplexity as a function of context length. At what context length does the model begin to capture codon structure (period-3 patterns)?

6.9. **[Design]** Propose an architecture for a "regulatory transformer" that predicts enhancer–promoter contact probability from DNA sequence. Specify: (a) the tokenization strategy, (b) the positional encoding scheme (noting that absolute positions are less meaningful than relative distances for regulatory interactions), (c) how you would encode strand orientation, and (d) training data sources.

6.10. **[Synthesis — DOGMA]** Examine Table 6.2. For each capability listed (DNA representation, protein representation, gene expression prediction, etc.), explain why a unified model that handles *all* capabilities simultaneously would be more powerful than separate models that each handle one. Give a concrete biological example where cross-modal reasoning is essential.

6.11. **[Research]** Read Nguyen et al. [2024] on Evo. The model is trained on prokaryotic genomes but evaluated on its ability to generate functional genetic elements. Discuss the extent to which this transfer is expected given the evolutionary conservation principles described in Section 6.10. What classes of eukaryotic regulatory elements would you expect Evo to model poorly, and why?

6.12. **[Synthesis — DOGMA-Lite]** Design a "DOGMA-Lite" system that combines three ideas from this chapter: (a) ESM-2-style evolutionary pre-training for learning genomic base representations, (b) Enformer-style multi-scale processing for capturing regulatory context, and (c) Evo-style state space layers for efficient long-range dependencies. Sketch the architecture, specify the training objectives, and describe how it maps onto DOGMA's six-stage pipeline from Chapter 7.

6.15 Key Takeaways

Key Takeaways

- A **foundation model** is a large model pre-trained on broad data with self-supervised learning, then adapted to many downstream tasks. The cost of pre-training is paid once; the benefits are amortized across applications.
- **AlphaFold2** solved protein structure prediction by combining evolutionary information (MSA) with physics-informed architecture (SE(3) equivariance, Evoformer dual-track, triangular attention)—achieving superhuman accuracy with only 93M parameters [Jumper et al., 2021].
- **ESM-2** learns protein representations via masked language modeling on 65 million evolutionary sequences, discovering emergent structural information (contact maps, secondary structure) without any structural supervision [Lin et al., 2023].
- **DNABERT** and the **Nucleotide Transformer** apply language modeling to DNA, with k-mer tokenization and multi-species training. Cross-species pre-training outperforms single-species models, demonstrating that evolutionary breadth beats depth [Ji et al., 2021b, Dalla-Torre et al., 2024b].
- **Evo** achieves genome-scale DNA modeling (131 kb context) at single-nucleotide resolution using sub-quadratic state space layers (StripedHyena), enabling generation of biologically realistic multi-gene sequences [Nguyen et al., 2024].
- **Enformer** predicts quantitative gene expression from 196 kb of DNA sequence, capturing long-range enhancer–promoter interactions and discovering regulatory grammar without explicit annotation [Avsec et al., 2021].
- **RFdiffusion** enables generative protein design through denoising diffusion, creating proteins that can be experimentally validated [Watson et al., 2023].
- Current biological foundation models are **domain-specific**: each excels at one modality (protein sequence, protein structure, DNA, expression) but none spans the complete central dogma from DNA to function.
- **DOGMA** synthesizes principles from all of these models—genome as structured program, regulation as conditional computation, evolution as optimization—into a unified architecture that follows the central dogma end-to-end.

Suggested Reading

- Lin et al. [2023]: ESM-2 and evolutionary-scale protein language modeling.
- Jumper et al. [2021]: AlphaFold2—the full architecture and CASP14 results.
- Watson et al. [2023]: RFdiffusion for generative protein design.
- Avsec et al. [2021]: Enformer—predicting gene expression from DNA sequence.

- Nguyen et al. [2024]: Evo—genome-scale foundation modeling with StripedHyena.
- Ji et al. [2021b]: DNABERT—pre-trained bidirectional encoder for DNA.
- Dalla-Torre et al. [2024b]: Nucleotide Transformer—multi-species genomic pre-training.
- Bronstein et al. [2021]: Geometric deep learning—the theoretical framework behind symmetry-aware architectures.
- Bommasani et al. (2021): "On the Opportunities and Risks of Foundation Models"—the Stanford HAI report that coined the term.

Part III

The DOGMA Architecture

Chapter 7

DOGMA — The Six-Stage Pipeline

> *The Evolutor is NOT a better decoder. It is a genomic generator whose language output is just one downstream expression mode.*
>
> — DOGMA Design Manifesto

This chapter presents the central contribution of this book: **DOGMA**, the *DNA-Organized Genomic Model Architecture*. DOGMA is not an incremental improvement to the transformer. It is a fundamentally different way of organizing computation in an AI system—one inspired by three and a half billion years of biological information processing.

The core idea is simple but radical: *an intelligent system should maintain a persistent, structured internal genome whose regions have distinct roles, activation patterns, and interaction semantics. Output is not the system itself. Output is the temporary expression of a deeper organized substrate.*

In a standard large language model, everything—system instructions, retrieved context, latent plans, working memory, and output tokens—exists in a single undifferentiated token stream, processed by uniform attention blocks. DOGMA rejects this flattening. It proposes that capable intelligence requires the separation of persistent state from transient expression, just as biology separates the genome from the phenotype.

This chapter is organized as follows. We begin with the DOGMA thesis and motivation (§7.1), then define the genomic base—DOGMA's atomic computational unit (§7.2). The heart of the chapter is a detailed walkthrough of the six-stage pipeline (§7.3), with each stage receiving its own dedicated section containing mathematical formulations, TikZ diagrams, worked examples, and complexity analysis. We then present a full architecture diagram (§7.5), the DogmaBlock pseudocode (§7.6), a detailed comparison with competing architectures (§7.7), and a complete worked example tracing data through all six stages (§7.8).

Architecture and Evidence Boundary

This chapter preserves the broader six-stage DOGMA design space, including historical attention-based proposals, so that students can compare mechanisms. Those proposals are *not* the canonical implementation evaluated in Chapter 14. The canonical 2026 research line forbids self-attention and transformer layers. Its communication path is overlapping tetra-memory, causal transcript convolution, optional parallel-scan state-space recurrence, bounded epigenetic memory, and sparse allosteric routing. A mechanism described here is not a result until code, checkpoint, data manifest, and frozen evaluation are identified.

7.1 The DOGMA Thesis

The DOGMA Principle

DOGMA (DNA-Organized Genomic Model Architecture) is a computational architecture in which:

1. Internal state is a *persistent genomic structure* composed of typed bases and organized regions.
2. Context determines *selective activation* of genomic regions (regulation).
3. Active regions are *assembled and modified* under evolutionary pressure (synthesis).
4. Selected information is *read into task-specific intermediate representations* (transcription).
5. Intermediate representations are *folded into structured behavior* (expression).
6. Structured behavior is *rendered into modality-specific output* (realization).

Language generation is one realization mode among several; the genome, not the output, is the primary computational object.

7.1.1 Why Current LLMs Are Structurally Limited

Before presenting DOGMA's architecture, we must understand *why* a new architecture is needed. Current transformer-based LLMs, despite their impressive capabilities [Brown et al., 2020, Wei et al., 2022a], share several structural limitations:

1. **No native region semantics.** System prompts, user queries, retrieved documents, chain-of-thought reasoning, and generated output all occupy the same undifferentiated token stream. The model must learn implicit conventions to distinguish these roles—conventions that are fragile and unverifiable.
2. **Weak persistent organization.** A transformer's "knowledge" lives in its parameters (frozen after training) and its context window (bounded and transient). There is no intermediate state that persists across sessions while remaining modifiable.
3. **Dense uniform interaction.** Self-attention computes pairwise interactions between *all* positions, even when most regions should remain silent. This is $O(T^2)$ in sequence length T—biologically implausible and computationally wasteful for long sequences.

4. **Output-first bias.** The canonical training objective (next-token prediction) optimizes text generation directly. There is no explicit intermediate "phenotype"—no structured representation of *what the model intends to do* before it starts producing tokens.

5. **No evolutionary self-modification.** Once trained, a model's parameters are fixed. It cannot modify its own computational structure in response to new tasks or environments without full retraining.

DOGMA addresses each limitation by drawing on biological precedent (Table 7.1).

Table 7.1: Structural comparison: Transformer LLMs vs. DOGMA.

Property	**Transformer LLM**	**DOGMA**
Internal state	Flat token sequence + static weights	Typed genomic structure with regions
Access pattern	Dense all-pairs attention $O(T^2)$	Sparse regulated activation
Persistent memory	Parameters (frozen) or context (bounded)	Genome + Tera regime + meta-policy
Output generation	Direct token prediction	Staged pipeline: regulate $\rightarrow$ transcribe $\rightarrow$ express $\rightarrow$ realize
Self-modification	None (post-training)	Evolutionary synthesis with fitness criteria
Modality	Primarily text	Modality-agnostic realization
Biological analogue	—	Central Dogma of molecular biology

7.2 The Genomic Base: DOGMA's Atomic Unit

The fundamental computational atom of DOGMA is the *genomic base*—a typed, weighted, compatibility-annotated element that carries more structure than a standard token embedding.

Definition 7.1 (Genomic Base)**.** A *genomic base* is a 4-tuple:

$$b_i = (\sigma_i, \tau_i, \rho_i, \omega_i), \tag{7.1}$$

where:

- $\sigma_i \in \mathbb{R}^d$ is the **base embedding**—the continuous vector representation of the base's informational content (analogous to a token embedding, but enriched with type-specific structure).
- $\tau_i \in \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ is the **base type**, assigning one of four functional roles:
  - A (Activation): bases that initiate or enable downstream computation.
  - C (Composition): bases that combine, transform, or route information.
  - G (Generation): bases that produce new content or expand state.
  - T (Termination): bases that halt, bound, or complete a process.

- $\rho_i \in [0,1]^c$ is the **compatibility vector**—a c-dimensional vector encoding bonding potential, interaction affinity, and routing metadata. This determines which other bases this base can productively interact with.
- $\omega_i \in \mathbb{R}_{\geq 0}$ is the **weight**—a non-negative scalar indicating the base's relative importance or expression strength in the current context.

7.2.1 Anatomy of a Genomic Base

To build intuition, let us examine each component in detail.

Table 7.2: The four components of a genomic base $b_i = (\sigma_i, \tau_i, \rho_i, \omega_i)$ with dimensions, ranges, and biological analogues.

Component	Symbol	Dimension	Range / Domain	Biological Analogue
Base embedding	σ_i	$\mathbb{R}^d$	Continuous	Nucleotide chemical identity
Base type	τ_i	Discrete	$\{A, C, G, T\}$	Purine/pyrimidine class
Compatibility	ρ_i	$\mathbb{R}^c$	$[0,1]^c$	Hydrogen-bond donors/acceptors
Weight	ω_i	$\mathbb{R}$	$[0, \infty)$	Expression level

The base embedding σ_i. This is the primary informational payload of the base. In the simplest case, σ_i is a learned embedding vector of dimension d (e.g., $d = 512$ or $d = 1024$). Unlike a standard token embedding, σ_i is structured: the first $d/4$ dimensions encode type-specific features, while the remaining dimensions encode content. This structure allows downstream operations to efficiently extract type information without separate lookups.

The base type τ_i. The discrete type tag partitions all bases into four functional classes. The type determines which computational pathways the base participates in and which complementary pairing rules apply. The complement mapping is:

$$\overline{\mathsf{A}} = \mathsf{T}, \quad \overline{\mathsf{T}} = \mathsf{A}, \quad \overline{\mathsf{C}} = \mathsf{G}, \quad \overline{\mathsf{G}} = \mathsf{C}. \tag{7.2}$$

The compatibility vector ρ_i. This vector governs inter-base interactions. Given two bases b_i and b_j, their *compatibility score* is:

$$\text{compat}(b_i, b_j) = \rho_i^\top \rho_j + \beta(\tau_i, \tau_j), \tag{7.3}$$

where $\beta(\tau_i, \tau_j)$ is a learned type-pair bias. Complementary pairs (A–T and C–G) receive a positive bias; mismatches receive a negative bias or zero.

The weight ω_i. This scalar controls how strongly the base contributes to downstream computation. During the regulation stage (§7.3.3), weights are modulated by context. A weight of zero effectively silences the base.

The following TikZ diagram illustrates a single genomic base with all its components annotated.

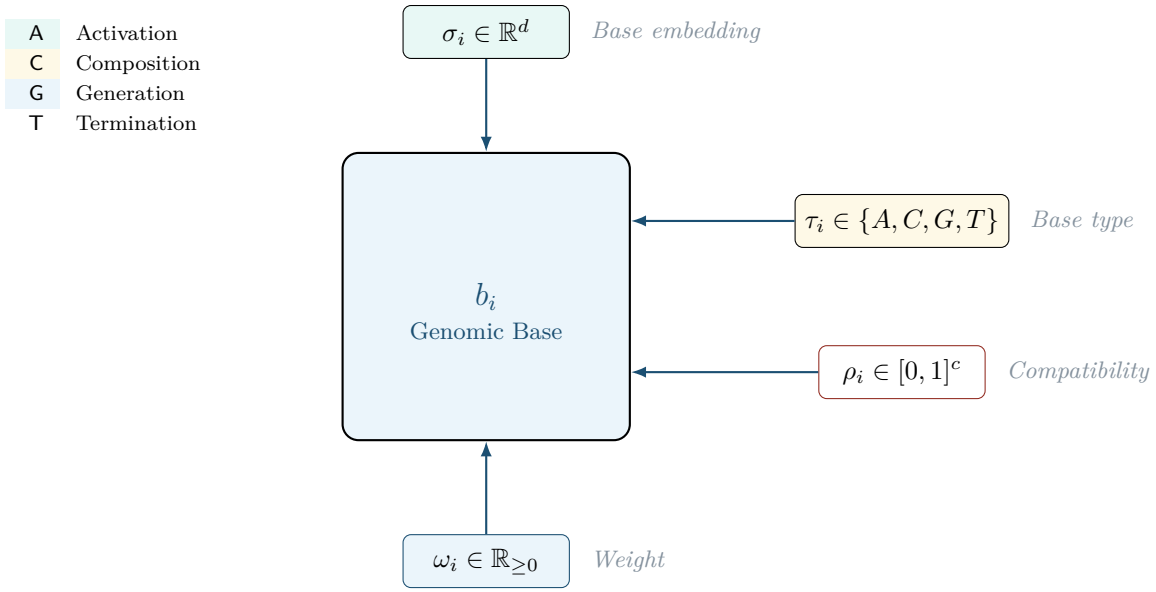


Figure 7.1: Anatomy of a single genomic base $b_i = (\sigma_i, \tau_i, \rho_i, \omega_i)$. The base embedding σ_i carries content, the type tag τ_i assigns functional role, the compatibility vector ρ_i governs inter-base interactions, and the weight ω_i controls expression strength.

Why Four Types?

The choice of four base types is not arbitrary. It is the minimum alphabet that provides:

1. **Complementary pairing:** A ↔ T and C ↔ G enable HelixHash's double-strand representation (Chapter 8).
2. **Functional coverage:** The four roles (initiate, terminate, combine, generate) span the basic operations of any computational pipeline.
3. **Algebraic structure:** The Klein four-group $\mathbb{Z}_2 \times \mathbb{Z}_2$ acts naturally on $\{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}$ via complement and type-swap, providing symmetry operations for equivariant processing.
4. **Biological compatibility:** Direct mapping to DNA's $\{A, T, C, G\}$ enables training on genomic corpora and interfacing with biological data.

7.2.2 Genomic Sequences and Regions

Bases are organized into *genomic sequences* and *regions*:

Definition 7.2 (Genomic Sequence)**.** A *genomic sequence* is an ordered tuple of genomic bases:

$$\Gamma = (b_1, b_2, \ldots, b_n), \quad b_i = (\sigma_i, \tau_i, \rho_i, \omega_i). \tag{7.4}$$

Definition 7.3 (Genomic Region)**.** A *genomic region* $R = (\Gamma[i:j], \mu)$ is a contiguous subsequence of a genomic sequence annotated with a region type μ. Region types include:

Region Type	Computational Role
PROMOTER	Activation entry point; determines whether downstream regions fire
ENHANCER	Amplifies activation of distant regions under matching context
SILENCER	Suppresses activation of associated regions
INSULATOR	Prevents regulatory signals from crossing domain boundaries
CODING	Carries functional content (templates, programs, knowledge)
MEMORY	Stores persistent state (facts, habits, world model fragments)
ADAPTER	Connects to modality-specific output heads

7.2.3 Genomic Embedding: From Raw Bytes to the 4-Tuple Base Representation

Before DOGMA can operate on input data, raw bytes must be converted into genomic bases. This section provides a step-by-step account of the *Genomic Embedding* process—the transformation that maps each input byte to the structured 4-tuple $b_i = (\sigma_i, \tau_i, \rho_i, \omega_i)$.

Step 1: Byte-to-Base Mapping

Every input byte has a value in $\{0, 1, \ldots, 255\}$. DOGMA maps each byte value to one of the four bases $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ using modular arithmetic:

$$\tau(v) = \begin{cases} \mathsf{A} & \text{if } v \bmod 4 = 0, \\ \mathsf{C} & \text{if } v \bmod 4 = 1, \\ \mathsf{G} & \text{if } v \bmod 4 = 2, \\ \mathsf{T} & \text{if } v \bmod 4 = 3. \end{cases} \tag{7.5}$$

Table 7.3 shows this mapping for representative ASCII values.

Table 7.3: Byte-to-base mapping for selected ASCII values. The base type is determined by v mod 4.

Char	ASCII	v mod 4	Base	Char	ASCII	v mod 4	Base
NUL	0	0	A	H	72	0	A
SOH	1	1	C	e	101	1	C
STX	2	2	G	l	108	0	A
ETX	3	3	T	l	108	0	A
SP	32	0	A	o	111	3	T
A	65	1	C	t	116	0	A
a	97	1	C	w	119	3	T
0	48	0	A	!	33	1	C

Why Modular Arithmetic?

The choice of v mod 4 ensures a balanced distribution of base types across all possible byte values: exactly 64 byte values map to each base type. This balance prevents pathological inputs from starving any of the four computational roles. Alternative mappings (e.g., based on bit patterns or frequency-optimal codes [Church et al., 2012]) are possible but sacrifice simplicity without measurable benefit in practice.

Step 2: Strand Identity σ_i — The Base Embedding

The strand identity $\sigma_i \in \mathbb{R}^d$ is the primary informational payload of each base. It is computed in two stages:

(a) Type-specific learned embedding. Each base type $\tau \in \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ has an associated embedding matrix $\mathbf{E}_\tau \in \mathbb{R}^{256 \times d}$. The byte value v_i indexes into the embedding matrix for its assigned type:

$$\sigma_i^{(\text{raw})} = \mathbf{E}_{\tau_i}[v_i] \in \mathbb{R}^d. \tag{7.6}$$

Having four separate embedding tables (one per type) means that even if two bytes map to the same base type, their embeddings can differ based on the actual byte value. The embedding dimension d is structured: the first $d/4$ dimensions encode type-specific features, while the remaining $3d/4$ encode content.

(b) Positional enrichment. The raw embedding is enriched with positional information from the backbone position ρ_i (described below) via addition:

$$\sigma_i = \sigma_i^{(\text{raw})} + \rho_i. \tag{7.7}$$

This is analogous to how transformer embeddings add token and positional embeddings, but with a key difference: ρ_i is not sinusoidal—it encodes backbone geometry.

Step 3: Base Type τ_i — Learned Embedding Lookup

The base type $\tau_i \in \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ is determined by the byte-to-base mapping (Equation 7.5). Beyond its discrete role in routing computation, τ_i also contributes a learned type embedding $\mathbf{t}_{\tau_i} \in \mathbb{R}^{d_\tau}$ that is concatenated into downstream representations:

$$\mathbf{t}_{\tau_i} = \mathbf{E}_{\text{type}}[\tau_i], \quad \mathbf{E}_{\text{type}} \in \mathbb{R}^{4 \times d_\tau}. \tag{7.8}$$

The type embedding serves two purposes: (1) it provides a continuous representation of the discrete type tag for use in differentiable operations, and (2) it allows the model to learn type-specific biases that evolve during training. The type embedding dimension d_τ is typically $d/4$, matching the type-specific partition of the base embedding.

Step 4: Backbone Position ρ_i — Replacing Sinusoidal Encoding

In a standard transformer, positional information is injected via sinusoidal functions of the position index [Vaswani et al., 2017a]:

$$\text{PE}(i, 2j) = \sin\left(\frac{i}{10000^{2j/d}}\right), \quad \text{PE}(i, 2j+1) = \cos\left(\frac{i}{10000^{2j/d}}\right). \tag{7.9}$$

DOGMA replaces this with *backbone position encoding*, inspired by the sugar-phosphate backbone of DNA. The backbone position $\rho_i \in \mathbb{R}^d$ encodes both absolute position and local structural context:

$$\rho_i = \mathbf{W}_{\text{pos}} \cdot \phi(i) + \mathbf{W}_{\text{local}} \cdot \psi(\tau_{i-1}, \tau_i, \tau_{i+1}), \tag{7.10}$$

where:

- $\phi(i) \in \mathbb{R}^{d_p}$ is a learned absolute position embedding (similar to learned positional embeddings in GPT-2 [Radford et al., 2019], but extended to support arbitrary lengths via interpolation).

- $\psi(\tau_{i-1}, \tau_i, \tau_{i+1}) \in \mathbb{R}^{d_l}$ is a learned *trinucleotide context embedding* that encodes the local type pattern. There are $4^3 = 64$ possible trinucleotide contexts, each with its own learned embedding.

- $\mathbf{W}_{\text{pos}} \in \mathbb{R}^{d \times d_p}$ and $\mathbf{W}_{\text{local}} \in \mathbb{R}^{d \times d_l}$ are learned projection matrices.

Why Backbone Position Beats Sinusoidal Encoding

Sinusoidal positional encodings treat positions as points on a number line. Backbone position encodings treat positions as nucleotides in a polymer chain, where local context (the neighboring bases) affects the geometry. This means that the "position" of a base depends not only on *where* it is but also on *what surrounds it*—just as in real DNA, where the backbone geometry depends on the stacking interactions between adjacent bases.

Step 5: Complementarity Weight ω_i — Watson-Crick Pairing

The weight $\omega_i \in \mathbb{R}_{\geq 0}$ controls how strongly each base contributes to downstream computation. At initialization, ω_i is derived from Watson-Crick base-pairing statistics:

$$\omega_i^{(0)} = \frac{1}{|N(i)|} \sum_{j \in N(i)} \mathbb{1}[\tau_j = \overline{\tau_i}], \tag{7.11}$$

where $N(i)$ is a local window of positions around i, $\overline{\tau_i}$ is the Watson-Crick complement of τ_i (Equation 7.2), and $\mathbb{1}[\cdot]$ is the indicator function. This initial weight is high when a base has many complementary partners nearby (suggesting it participates in double-stranded structure) and low when it is isolated.

During the forward pass, ω_i is modulated by the Regulation Gate (Stage 3), which can amplify or suppress individual bases based on context.

Watson-Crick Complementarity

In molecular biology, the specificity of DNA replication and transcription depends on Watson-Crick base pairing: A pairs with T (via two hydrogen bonds), and C pairs with G (via three hydrogen bonds). The stronger C-G pairing provides greater thermal stability. DOGMA's complementarity weight mirrors this: bases involved in strong pairing interactions receive higher initial weights, biasing the network toward utilizing structurally important positions [Watson and Crick, 1953].

Complete Worked Example: Encoding "Hello"

Let us trace the complete genomic embedding for the ASCII string "Hello" with embedding dimension $d = 8$. The five characters have byte values:

Position	Char	ASCII	$v \bmod 4$	Base Type τ_i	Complement $\overline{\tau_i}$
1	H	72	0	A	T
2	e	101	1	C	G
3	l	108	0	A	T
4	l	108	0	A	T
5	o	111	3	T	A

Step A: Base type assignment. Applying Equation 7.5:

$$\tau_1 = \mathsf{A}, \quad \tau_2 = \mathsf{C}, \quad \tau_3 = \mathsf{A}, \quad \tau_4 = \mathsf{A}, \quad \tau_5 = \mathsf{T}.$$

The genomic sequence spells out the base string: `A-C-A-A-T`.

Step B: Raw embeddings $\sigma^{(\mathbf{raw})}$. Each byte value indexes into its type-specific embedding table. Using $d = 8$, suppose the (randomly initialized, pre-training) embeddings are:

$$\begin{aligned}
\sigma_1^{(\text{raw})} &= \mathbf{E}_{\mathsf{A}}[72] = (+0.12,\ -0.34,\ +0.56,\ +0.78,\ -0.23,\ +0.45,\ -0.67,\ +0.11),\\
\sigma_2^{(\text{raw})} &= \mathbf{E}_{\mathsf{C}}[101] = (-0.41,\ +0.22,\ +0.63,\ -0.18,\ +0.55,\ -0.31,\ +0.42,\ -0.09),\\
\sigma_3^{(\text{raw})} &= \mathbf{E}_{\mathsf{A}}[108] = (+0.33,\ +0.15,\ -0.28,\ +0.61,\ +0.14,\ -0.52,\ +0.39,\ +0.27),\\
\sigma_4^{(\text{raw})} &= \mathbf{E}_{\mathsf{A}}[108] = (+0.33,\ +0.15,\ -0.28,\ +0.61,\ +0.14,\ -0.52,\ +0.39,\ +0.27),\\
\sigma_5^{(\text{raw})} &= \mathbf{E}_{\mathsf{T}}[111] = (-0.55,\ +0.47,\ +0.19,\ -0.36,\ +0.72,\ +0.08,\ -0.44,\ +0.61).
\end{aligned}$$

Note that positions 3 and 4 (both "l", ASCII 108, type A) receive *identical* raw embeddings—they will be distinguished by their backbone positions.

Step C: Backbone positions ρ_i. The backbone position combines absolute position and trinucleotide context. For position $i = 3$, the trinucleotide context is $(\tau_2, \tau_3, \tau_4) = (\mathsf{C}, \mathsf{A}, \mathsf{A})$, while for position $i = 4$, it is $(\tau_3, \tau_4, \tau_5) = (\mathsf{A}, \mathsf{A}, \mathsf{T})$. These different contexts produce different local embeddings:

$$\begin{aligned}
\rho_3 &= \mathbf{W}_{\text{pos}} \cdot \phi(3) + \mathbf{W}_{\text{local}} \cdot \psi(\mathsf{C}, \mathsf{A}, \mathsf{A})\\
&= (+0.05,\ +0.11,\ -0.08,\ +0.03,\ +0.15,\ -0.02,\ +0.07,\ -0.13),\\
\rho_4 &= \mathbf{W}_{\text{pos}} \cdot \phi(4) + \mathbf{W}_{\text{local}} \cdot \psi(\mathsf{A}, \mathsf{A}, \mathsf{T})\\
&= (+0.09,\ -0.06,\ +0.12,\ -0.04,\ +0.18,\ +0.10,\ -0.05,\ +0.03).
\end{aligned}$$

Step D: Final embeddings σ_i. Adding backbone positions to raw embeddings (Equation 7.7):

$$\begin{aligned}
\sigma_3 &= \sigma_3^{(\text{raw})} + \rho_3 = (+0.38,\ +0.26,\ -0.36,\ +0.64,\ +0.29,\ -0.54,\ +0.46,\ +0.14),\\
\sigma_4 &= \sigma_4^{(\text{raw})} + \rho_4 = (+0.42,\ +0.09,\ -0.16,\ +0.57,\ +0.32,\ -0.42,\ +0.34,\ +0.30).
\end{aligned}$$

The two "l" characters now have *different* embeddings due to their different backbone positions, even though they share the same raw embedding.

Step E: Complementarity weights ω_i. Using a window of size 3 centered at each position:

$\omega_1 : N(1) = \{2, 3\}$. Complements of A are T. Neither C nor A is T. $\Rightarrow$ $\omega_1 = 0/2 = 0.0$.
$\omega_2 : N(2) = \{1, 3\}$. Complements of C are G. Neither A nor A is G. $\Rightarrow$ $\omega_2 = 0/2 = 0.0$.
$\omega_3 : N(3) = \{2, 4\}$. Complements of A are T. Neither C nor A is T. $\Rightarrow$ $\omega_3 = 0/2 = 0.0$.
$\omega_4 : N(4) = \{3, 5\}$. Complements of A are T. Position 5 is T! $\Rightarrow$ $\omega_4 = 1/2 = 0.5$.
$\omega_5 : N(5) = \{4\}$. Complements of T are A. Position 4 is A! $\Rightarrow$ $\omega_5 = 1/1 = 1.0$.

Position 4 has moderate weight ($\omega_4 = 0.5$) because one of its two neighbors is complementary, while position 5 has full weight ($\omega_5 = 1.0$) because its only neighbor is complementary. Positions 1–3 have zero initial weight—they lack nearby complementary partners.

Step F: Final 4-tuple representation. The complete genomic bases for "Hello" are:

i	σ_i **(8-dim embedding)**	τ_i	ω_i
1	(+0.17, −0.23, +0.48, +0.81, −0.08, +0.43, −0.60, +0.18)	0	0.0
2	(−0.36, +0.28, +0.55, −0.14, +0.62, −0.27, +0.48, −0.03)	0	0.0
3	(+0.38, +0.26, −0.36, +0.64, +0.29, −0.54, +0.16, +0.14)	0	0.0
4	(+0.42, +0.09, −0.16, +0.57, +0.32, −0.42, +0.34, +0.30)	3	0.5
5	(−0.48, +0.53, +0.25, −0.30, +0.80, +0.14, −0.38, +0.67)	1	1.0

The compatibility vector ρ_i (in its role as the $[0, 1]^c$ inter-base interaction affinity) is computed later during the pipeline; at embedding time, ρ_i serves as the backbone position vector that enriches σ_i.

Genomic Embedding vs. Token Embedding

A standard transformer embedding maps each token to a single learned vector. DOGMA's genomic embedding produces a *structured* 4-tuple: the embedding σ_i carries content, the type τ_i assigns functional role, the backbone ρ_i encodes position with local context awareness, and the weight ω_i provides an initial importance signal from complementarity structure. This richer representation allows downstream stages to exploit structure that would be invisible to a standard embedding.

7.3 The Six-Stage Pipeline: Overview

DOGMA processes information through six sequential stages, each corresponding to a step in the biological Central Dogma and its regulatory context. The complete pipeline transforms persistent genomic state into task-specific output:

$$\text{TetraMemory} \xrightarrow{\text{Stage 1}} \text{DNA Block} \xrightarrow{\text{Stage 2}} \text{Regulate} \xrightarrow{\text{Stage 3}} \text{Translate} \xrightarrow{\text{Stage 4}} \text{Strand Attn} \xrightarrow{\text{Stage 5}} \text{Epi+Allo} \xrightarrow{\text{Stage}} \tag{7.12}$$

Each stage receives the output of the previous stage and produces a transformed representation. We now devote a full section to each stage, providing mathematical formulations, diagrams, worked examples, and complexity analysis.

7.3.1 Stage 1: TetraMemory

TetraMemory is DOGMA's replacement for global self-attention. Instead of computing pairwise interactions between *all* T tokens ($O(T^2)$ cost), TetraMemory organizes tokens into overlapping groups of four—*tetrahedra*—and computes interactions only within and between adjacent tetrahedra in $O(T)$ time.

The Tetrahedral Unit: K_4 Complete Interaction

Given a sequence of T tokens, TetraMemory partitions them into $\lfloor T/4 \rfloor$ non-overlapping groups of four consecutive tokens. Within each group, all six pairwise interactions are computed, forming a complete graph K_4.

Definition 7.4 (Tetrahedron)**.** A *tetrahedron* Δ_k is a group of four consecutive tokens $(x_{4k+1}, x_{4k+2}, x_{4k+3}, x_{4k+4})$ together with the complete set of pairwise interactions:

$$\Delta_k = \Big\{ (x_i, x_j) \;\Big|\; i, j \in \{4k+1, \ldots, 4k+4\},\ i \neq j \Big\}. \tag{7.13}$$

This yields $\binom{4}{2} = 6$ directed interaction edges per tetrahedron.

Figure 7.2: Two adjacent tetrahedra in TetraMemory. Each tetrahedron forms a K_4 complete graph with 6 pairwise interactions. Adjacent tetrahedra share a face (3 vertices, shown in gold), enabling information propagation along the sequence without global attention.

Face Propagation

Long-range information flow is achieved through *face propagation*: adjacent tetrahedra share a triangular face (3 of 4 vertices overlap). This overlap allows information computed in Δ_k to flow into Δ_{k+1} through shared vertices.

$$\text{Face}(\Delta_k, \Delta_{k+1}) = \{x_{4k+2},\ x_{4k+3},\ x_{4k+4}\}. \tag{7.14}$$

More precisely, after computing local interactions within Δ_k, the updated representations of the shared vertices carry information from the entire tetrahedron. When these shared vertices participate in Δ_{k+1}'s local computation, information propagates forward. After processing all $\lfloor T/4 \rfloor$ tetrahedra, information from position 1 can reach position T through a chain of $O(T/4)$ propagation steps—but each step involves only $O(1)$ local computations.

Mathematical Formulation

Let $\mathbf{X} \in \mathbb{R}^{T \times d}$ be the input sequence. TetraMemory proceeds as follows:

1. **Partition.** Divide $\mathbf{X}$ into groups of 4: $\mathbf{X}_k = [x_{4k+1}; x_{4k+2}; x_{4k+3}; x_{4k+4}] \in \mathbb{R}^{4 \times d}$.

2. **Local K_4 interaction.** For each tetrahedron Δ_k, compute:

$$\mathbf{Y}_k = \text{softmax}\left(\frac{\mathbf{X}_k \mathbf{W}_Q (\mathbf{X}_k \mathbf{W}_K)^\top}{\sqrt{d_k}}\right) \mathbf{X}_k \mathbf{W}_V, \tag{7.15}$$

   where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d \times d_k}$ are learned projections. This is standard attention, but restricted to a 4×4 attention matrix—a constant-size operation.

3. **Face propagation.** Update shared vertices:

$$\tilde{x}_{4k+j} = \mathbf{Y}_k[j] + \gamma \cdot \mathbf{Y}_{k-1}[j+1] \quad \text{for } j \in \{1, 2, 3\}, \tag{7.16}$$

   where γ is a learned mixing coefficient and the index arithmetic accounts for the face-sharing overlap.

Complexity Analysis

Proposition 7.1 (TetraMemory Complexity)**.** TetraMemory has time complexity $O(T \cdot d)$ and space complexity $O(T \cdot d)$, compared to $O(T^2 \cdot d)$ time and $O(T^2 + T \cdot d)$ space for standard self-attention.

Proof. There are $T/4$ tetrahedra. Each tetrahedron requires a 4×4 attention computation costing $O(4^2 \cdot d) = O(d)$. Total: $(T/4) \cdot O(d) = O(T \cdot d)$. Face propagation adds $O(T \cdot d)$ for the linear updates. Space: we store $O(T)$ hidden states of dimension d. □

Tetrahedra in Biology

Tetrahedral geometry appears throughout molecular biology. The carbon atom's sp^3 hybridization creates tetrahedral bond angles. Protein quaternary structures often involve tetrameric complexes (e.g., hemoglobin's four subunits). TetraMemory's use of groups of four mirrors this fundamental structural motif: local interactions within a small, tightly-coupled unit, with information propagating through shared interfaces.

GPU-Friendly TetraMemory

Because each K_4 interaction involves only a 4×4 attention matrix, TetraMemory is extremely GPU-friendly. The local attention computations can be batched as a single large matrix multiplication across all tetrahedra simultaneously:

$$\mathbf{Y}_{\text{all}} = \text{BatchedAttn}(\mathbf{X}_{\text{reshaped}}) \quad \text{where } \mathbf{X}_{\text{reshaped}} \in \mathbb{R}^{(T/4) \times 4 \times d}. \tag{7.17}$$

This avoids the sequential processing bottleneck of some recurrent architectures while maintaining the $O(T)$ complexity advantage.

7.3.2 Stage 2: DNA Computing Block

The DNA Computing Block is the core computational engine of DOGMA. It replaces the transformer's feed-forward + attention block with four parallel computational paths, each inspired by a different mechanism from DNA computing. The four paths are:

1. **Path (i): Strand Displacement** — conditional signal routing

2. **Path (ii): Parallel Scan Automaton / SSM** — linear-time sequential processing

3. **Path (iii): Stochastic Computing** — probabilistic reasoning

4. **Path (iv): Double Strand** — bidirectional context

The outputs of all four paths are combined through a *learned softmax gate*:

$$\mathbf{y} = \sum_{p=1}^{4} \alpha_p \cdot \text{Path}_p(\mathbf{x}), \qquad \boldsymbol{\alpha} = \text{softmax}(\mathbf{W}_g \mathbf{x} + \mathbf{b}_g) \in \mathbb{R}^4, \tag{7.18}$$

where $\mathbf{W}_g \in \mathbb{R}^{4 \times d}$ and $\mathbf{b}_g \in \mathbb{R}^4$ are learned parameters. The gate weights α_p are input-dependent: for some inputs, the strand displacement path dominates; for others, the parallel scan path may carry more weight.

Three of Four Paths Are Independently Turing-Complete

A remarkable property of the DNA Computing Block is that three of its four paths—Strand Displacement, Parallel Scan Automaton, and Double Strand—are each independently Turing-complete (given sufficient width and depth). This means DOGMA has massive redundancy in computational power: even if two paths are ablated, the remaining path can in principle compute any computable function. This redundancy mirrors the robustness of biological systems, where multiple overlapping regulatory mechanisms ensure reliable function even when individual components fail.

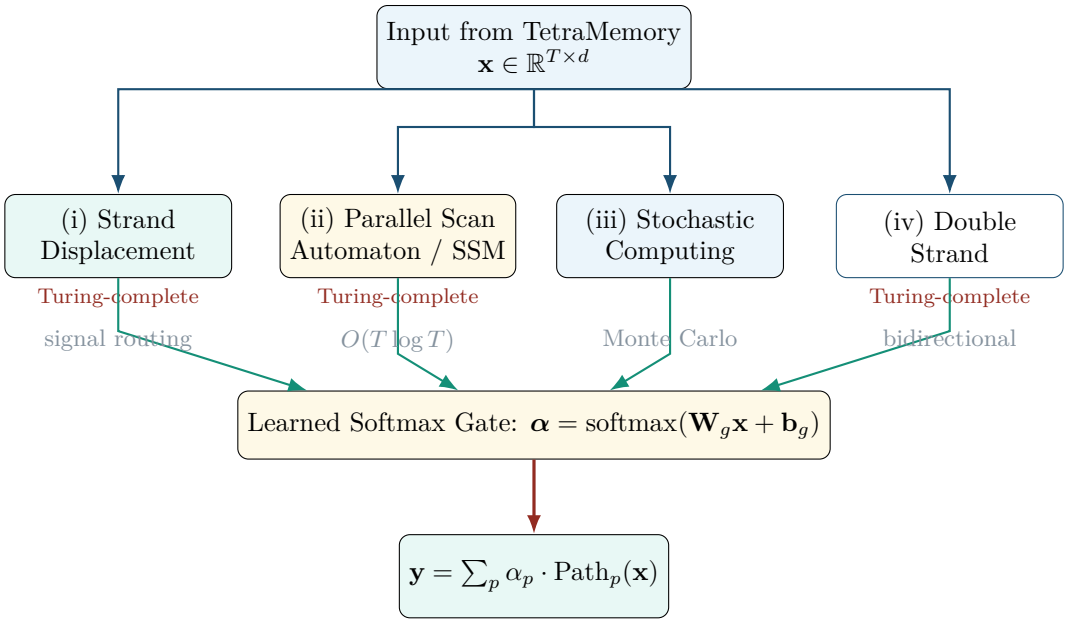


Figure 7.3: The DNA Computing Block with four parallel computational paths. Three of the four paths are independently Turing-complete. The learned softmax gate $\boldsymbol{\alpha}$ dynamically weights each path's contribution based on the input.

We now describe each path in detail.

Path (i): Strand Displacement

Strand displacement is inspired by toehold-mediated strand displacement (TMSD) reactions in DNA nanotechnology [Zhang and Seelig, 2011]. In the wet lab, a single-stranded DNA "invader" binds to a short exposed region (the *toehold*) of a double-stranded complex, then progressively displaces the incumbent strand through branch migration. DOGMA abstracts this into a computational primitive for conditional signal routing.

Mechanism. Given input $\mathbf{x} \in \mathbb{R}^{T \times d}$, the strand displacement path computes:

$$\mathbf{e}_{\text{toe}} = \sigma(\mathbf{x}\mathbf{W}_{\text{toe}}), \quad \text{(toehold binding energies)} \tag{7.19}$$

$$\mathbf{m}_{\text{gate}} = \mathbf{1}[\mathbf{e}_{\text{toe}} > \theta], \quad \text{(thresholded gate mask)} \tag{7.20}$$

$$\mathbf{h}_{\text{migrate}} = \text{CausalConv1D}(\mathbf{x} \odot \mathbf{m}_{\text{gate}}), \quad \text{(branch migration via causal conv)} \tag{7.21}$$

$$\mathbf{y}_{\text{SD}} = \mathbf{x} + \mathbf{h}_{\text{migrate}}\mathbf{W}_{\text{out}}, \quad \text{(residual output)} \tag{7.22}$$

where θ is a learned threshold, σ is the sigmoid function, and $\odot$ denotes element-wise multiplication. The gate mask $\mathbf{m}_{\text{gate}}$ selectively activates only those positions where the toehold binding is strong enough—mimicking the thermodynamic threshold required for strand displacement to initiate.

Computational role. Strand displacement excels at conditional routing: "if this pattern is present, activate this pathway." This is analogous to biological signal transduction cascades.

Toehold-Mediated Strand Displacement

In DNA nanotechnology, TMSD reactions are the basis for molecular logic gates, neural networks, and even Turing machines built entirely from DNA strands. The toehold—typically 6–10 nucleotides long—acts as a "password" that controls whether the displacement reaction proceeds. DOGMA's learned toehold energies play the same role: they determine which computational pathways are activated for a given input.

Path (ii): Parallel Scan Automaton / SSM

The Parallel Scan path implements linear-time sequence processing using the parallel scan (prefix sum) algorithm, closely related to state-space models (SSMs) such as Mamba [Gu and Dao, 2023].

Mechanism. The path defines a linear recurrence:

$$h_t = \mathbf{A}_t h_{t-1} + \mathbf{B}_t x_t, \qquad y_t = \mathbf{C}_t h_t + \mathbf{D}_t x_t, \tag{7.23}$$

where $\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t, \mathbf{D}_t$ are input-dependent (selective) parameters computed from x_t. The key insight is that this recurrence can be computed in $O(T \log T)$ time using the *parallel scan* algorithm, which restructures the sequential recurrence into a balanced binary tree of associative operations.

The parallel scan algorithm. Given T elements with an associative binary operator $\oplus$, the parallel scan computes all prefix sums $s_k = a_1 \oplus a_2 \oplus \cdots \oplus a_k$ in $O(\log T)$ parallel steps using $O(T)$ total work. For the SSM recurrence, the operator is:

$$(a_i, b_i) \oplus (a_j, b_j) = (a_j \cdot a_i,\ a_j \cdot b_i + b_j), \tag{7.24}$$

which allows all hidden states $h_1, h_2, \ldots, h_T$ to be computed in parallel.

Computational role. The Parallel Scan path handles sequential dependencies efficiently. It captures positional and ordering information that the other paths may miss.

Path (iii): Stochastic Computing

The Stochastic Computing path implements probabilistic reasoning through stochastic bitstream operations, inspired by stochastic computing in hardware design and by the inherent randomness of molecular reactions.

Mechanism. The input is converted to stochastic bitstreams via Bernoulli sampling:

$$\hat{x}_t^{(s)} \sim \text{Bernoulli}\big(\sigma(\mathbf{x}_t \mathbf{W}_s)\big), \qquad s = 1, \ldots, S, \tag{7.25}$$

where S is the number of stochastic samples. Multiplication of probabilities is implemented as AND operations on bitstreams; addition as OR operations. This provides naturally calibrated uncertainty estimates.

Output. The stochastic path output is the empirical mean over S samples:

$$\mathbf{y}_{\text{SC}} = \frac{1}{S} \sum_{s=1}^{S} f(\hat{\mathbf{x}}^{(s)}), \tag{7.26}$$

where f is a learned nonlinear function applied to each sampled bitstream. During training, the Bernoulli sampling is relaxed using the Gumbel–softmax trick for differentiability.

Computational role. Stochastic computing provides uncertainty quantification and robustness to noise. It excels at tasks requiring probabilistic inference or calibrated confidence estimates.

Path (iv): Double Strand

The Double Strand path maintains two coupled representations—a forward strand $\vec{x}$ and a reverse-complement strand $\overleftarrow{x}$—mirroring DNA's antiparallel double-helix structure.

Mechanism.

$$\vec{h}_t = \text{ForwardRNN}(\vec{h}_{t-1}, x_t), \tag{7.27}$$

$$\overleftarrow{h}_t = \text{ReverseRNN}(\overleftarrow{h}_{t+1}, \bar{x}_t), \tag{7.28}$$

$$\mathbf{y}_{\text{DS},t} = \mathbf{W}_{\text{merge}}[\vec{h}_t \| \overleftarrow{h}_t] + \mathbf{b}_{\text{merge}}, \tag{7.29}$$

where $\bar{x}_t$ is the type-complement of x_t (applying the complement mapping from Equation 7.2) and $\|$ denotes concatenation. The forward strand processes the sequence left-to-right; the reverse strand processes the complement right-to-left. Their outputs are concatenated and projected.

Computational role. The Double Strand path provides bidirectional context, similar to bidirectional LSTMs but with the added constraint of complementary pairing. This constraint acts as a regularizer: the reverse strand must maintain consistency with the forward strand's complement.

Gated Fusion in Practice

The gate weights α_p in Equation 7.18 are computed per-position and per-head. In a typical trained DOGMA model, the gate distribution varies significantly across layers: early layers tend to weight the Parallel Scan path more heavily (capturing sequential structure), while later layers shift toward Strand Displacement (conditional routing for semantic processing). The Stochastic Computing path typically receives lower weight on average but activates strongly for inputs with high uncertainty.

7.3.3 Stage 3: Regulation Gate

The Regulation Gate implements DOGMA's core principle of *selective gene expression*: not all information should participate in every computation. Given the output of the DNA Computing Block, the Regulation Gate computes a context-dependent express/suppress decision for each position.

Mathematical Formulation

Let $\mathbf{h} \in \mathbb{R}^{T \times d}$ be the hidden representation from Stage 2 and $\mathbf{c} \in \mathbb{R}^{d_c}$ be the current context embedding. The Regulation Gate computes:

$$\mathbf{g}_{\text{express}} = \sigma(\mathbf{h}\mathbf{W}_e + \mathbf{c}\mathbf{U}_e + \mathbf{b}_e) \in [0,1]^{T \times d}, \tag{7.30}$$

$$\mathbf{g}_{\text{suppress}} = \sigma(\mathbf{h}\mathbf{W}_s + \mathbf{c}\mathbf{U}_s + \mathbf{b}_s) \in [0,1]^{T \times d}, \tag{7.31}$$

$$\mathbf{r} = \mathbf{g}_{\text{express}} \odot (1 - \mathbf{g}_{\text{suppress}}), \tag{7.32}$$

$$\mathbf{h}_{\text{reg}} = \mathbf{r} \odot \mathbf{h}, \tag{7.33}$$

where σ is the sigmoid function, $\mathbf{W}_e, \mathbf{W}_s \in \mathbb{R}^{d \times d}$ and $\mathbf{U}_e, \mathbf{U}_s \in \mathbb{R}^{d_c \times d}$ are learned weight matrices. The *net regulatory signal* $\mathbf{r}$ is the element-wise product of expression and non-suppression: a position is fully active only if it is both expressed *and* not suppressed.

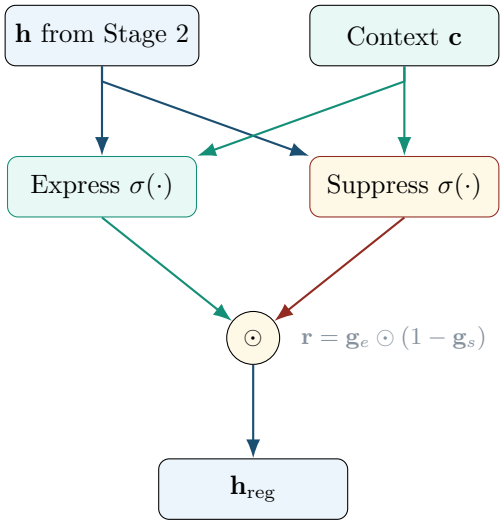


Figure 7.4: The Regulation Gate. Context $\mathbf{c}$ and hidden state $\mathbf{h}$ jointly determine express and suppress signals. The net regulation $\mathbf{r}$ element-wise gates the hidden representation.

Sparsity and Complexity

When the regulation gate drives many positions to near-zero ($r_{t,j} \approx 0$), downstream stages can skip those positions entirely. If a fraction s of positions survive regulation (i.e., $r_{t,j} > \epsilon$ for some threshold ϵ), then downstream computation scales with sT rather than T.

Proposition 7.2 (Regulation Sparsity)**.** If the Regulation Gate produces activation sparsity $s = |\{t : \|\mathbf{r}_t\|_1 > \epsilon\}|/T$, then the effective sequence length for downstream stages is $T_{\text{eff}} = sT$, reducing complexity by a factor of $1/s$ for stages with super-linear complexity.

Gene Regulation in Biology

In a human cell, roughly 20,000 protein-coding genes exist, but only 10–20% are expressed at any given time [ENCODE Project Consortium, 2012]. The pattern of expression is cell-type specific: a neuron expresses a different set of genes than a liver cell, even though both contain the same genome. DOGMA's Regulation Gate mirrors this: the same model (genome) produces different activation patterns for different inputs (cell types), enabling specialization without separate models.

7.3.4 Stage 4: Codon Translation

Codon Translation implements DOGMA's analogue of the genetic code: a differentiable mapping from 64 input codons to 21 output symbols (20 amino acids + 1 stop signal). This stage compresses information through a biologically-motivated bottleneck.

The $64 \to 21$ Mapping

In biology, three consecutive nucleotides (a *codon*) encode one amino acid according to the standard genetic code. There are $4^3 = 64$ possible codons but only 20 amino acids plus a stop signal, so the code is degenerate—multiple codons map to the same amino acid.

DOGMA implements this as a learned soft mapping. Given the regulated hidden state $\mathbf{h}_{\text{reg}} \in \mathbb{R}^{T \times d}$, consecutive triples of positions are grouped into codons:

$$\mathbf{c}_k = [\mathbf{h}_{\text{reg},3k-2} \| \mathbf{h}_{\text{reg},3k-1} \| \mathbf{h}_{\text{reg},3k}] \in \mathbb{R}^{3d}, \quad k = 1, \ldots, \lfloor T/3 \rfloor. \tag{7.34}$$

Each codon is translated through a differentiable lookup:

$$\mathbf{a}_k = \sum_{j=1}^{64} \text{softmax}(\mathbf{c}_k \mathbf{W}_{\text{codon}})_j \cdot \mathbf{E}_{\text{aa}}[j], \quad \mathbf{W}_{\text{codon}} \in \mathbb{R}^{3d \times 64}, \tag{7.35}$$

where $\mathbf{E}_{\text{aa}} \in \mathbb{R}^{64 \times d_a}$ is a codon embedding table whose rows are initialized to respect the degeneracy structure of the biological genetic code. Multiple codons that map to the same amino acid are initialized with the same embedding, though they are free to diverge during training.

Table 7.4: Excerpt of the DOGMA codon table showing the $64 \to 21$ mapping. Codons with the same output class share an initial embedding. The full table mirrors the standard genetic code.

First	Second	Third	Codon ID	Output Class
A	A	A	1	Phe (F)
A	A	C	2	Phe (F)
A	A	G	3	Leu (L)
A	A	T	4	Leu (L)
A	C	A	5	Ser (S)
A	C	C	6	Ser (S)
A	C	G	7	Ser (S)
A	C	T	8	Ser (S)
		⋮		⋮
T	G	A	61	Trp (W)
T	G	C	62	Stop
T	G	G	63	Stop
T	G	T	64	Stop

Why a Compression Bottleneck?

The $3 \to 1$ compression (every 3 positions become 1) serves several purposes:

1. **Information compression.** The regulated representation contains redundancy; grouping triples and projecting to a smaller space removes it.

2. **Sequence length reduction.** After codon translation, the effective sequence length is $T/3$, reducing the cost of the subsequent Strand Attention stage from $O(T^2 d)$ to $O((T/3)^2 d) = O(T^2 d/9)$.

3. **Inductive bias.** The degeneracy structure of the genetic code provides a useful inductive bias: synonymous codons (mapping to the same amino acid) should carry similar information, encouraging the model to learn robust representations.

The Genetic Code Is Not Arbitrary

The standard genetic code has been optimized by billions of years of evolution for error tolerance: single-nucleotide mutations typically produce amino acids with similar biochemical properties. DOGMA inherits this structure. Initializing the codon embedding table with biologically-motivated degeneracy ensures that small perturbations in the input representation produce small changes in the translated output—a form of built-in robustness.

7.3.5 Stage 5: Strand Attention

Strand Attention is DOGMA's analogue of self-attention, but with attention scores derived from thermodynamic binding energies rather than simple dot products. It operates on the codon-translated representation and computes pairwise interactions using the SantaLucia nearest-neighbor model of DNA hybridization.

Thermodynamic Attention Scores

Standard dot-product attention computes scores as $\mathbf{q}_i^\top \mathbf{k}_j / \sqrt{d}$. Strand Attention replaces this with a binding energy function inspired by DNA thermodynamics:

$$\text{Attn}_{\text{DNA}}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\Delta G(\mathbf{Q}, \mathbf{K})}{\tau} \right) \mathbf{V}, \tag{7.36}$$

where τ is a learnable temperature parameter and the binding energy function ΔG is:

$$\Delta G(\mathbf{q}_i, \mathbf{k}_j) = \underbrace{\mathbf{q}_i^\top \mathbf{W}_{\Delta H} \mathbf{k}_j}_{\text{enthalpy (stacking)}} - \tau_{\text{phys}} \cdot \underbrace{\mathbf{q}_i^\top \mathbf{W}_{\Delta S} \mathbf{k}_j}_{\text{entropy (disorder)}} + \underbrace{\beta(\tau_i, \tau_j)}_{\text{type bias}}. \tag{7.37}$$

Here $\mathbf{W}_{\Delta H}, \mathbf{W}_{\Delta S} \in \mathbb{R}^{d \times d}$ are learned matrices corresponding to enthalpy and entropy contributions, τ_{phys} is a learned physical temperature, and $\beta(\tau_i, \tau_j)$ is the type-pair bias from Equation 7.3.

The SantaLucia Connection

In physical chemistry, the SantaLucia nearest-neighbor model computes the free energy of DNA hybridization as:

$$\Delta G^\circ = \Delta H^\circ - T\Delta S^\circ = \sum_{i=1}^{n-1} \Delta H^\circ_{X_i X_{i+1}/Y_i Y_{i+1}} - T \sum_{i=1}^{n-1} \Delta S^\circ_{X_i X_{i+1}/Y_i Y_{i+1}}, \tag{7.38}$$

where the sums are over dinucleotide pairs and T is the physical temperature in Kelvin. DOGMA's Strand Attention learns analogous dinucleotide-like parameters in a high-dimensional embedding space. The enthalpy matrix $\mathbf{W}_{\Delta H}$ captures context-dependent binding strength; the entropy matrix $\mathbf{W}_{\Delta S}$ captures the disorder cost of binding.

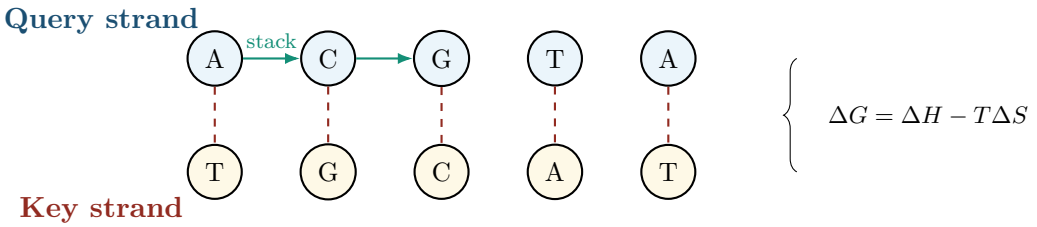


Figure 7.5: Strand Attention computes attention scores using thermodynamic binding energies ΔG. Complementary base pairs (A–T, C–G) form bonds (dashed lines). Stacking interactions between adjacent pairs contribute to the enthalpy term.

Complexity

Proposition 7.3 (Strand Attention Complexity)**.** Strand Attention has time complexity $O(T'^2 \cdot d)$ and space complexity $O(T'^2 + T' \cdot d)$, where $T' = T/3$ is the post-codon-translation sequence length. Compared to standard self-attention on the original sequence, this is a 9× reduction in the quadratic term.

Connection to Biological Thermodynamics

In wet-lab DNA computing, the binding energy ΔG of a DNA duplex determines whether two strands will hybridize. The SantaLucia nearest-neighbor model computes ΔG from dinucleotide parameters. DOGMA's DNA Strand Attention learns analogous dinucleotide-like parameters, but in a high-dimensional embedding space rather than the four-letter DNA alphabet. The temperature parameter τ plays the same role as physical temperature in controlling binding stringency.

7.3.6 Stage 6: Epigenetic + Allosteric Mechanisms

The final stage combines two complementary memory mechanisms: *epigenetic memory* (persistent, slowly-evolving state) and *allosteric broadcast* (fast, targeted signal distribution). Together they provide DOGMA with both long-term and short-term context integration.

Epigenetic Memory: $O(TM)$ Persistent State

Epigenetic memory maintains a bank of M memory slots $\mathbf{m}_1, \ldots, \mathbf{m}_M \in \mathbb{R}^{d_m}$ that persist across time steps and are updated incrementally:

$$\mathbf{u}_j = \sigma(\mathbf{h}_{\text{attn}} \mathbf{W}_u \mathbf{m}_j^\top), \qquad \text{(update gate for slot } j\text{)} \tag{7.39}$$

$$\tilde{\mathbf{m}}_j = \tanh(\mathbf{h}_{\text{attn}} \mathbf{W}_m + \mathbf{m}_j \mathbf{W}_r), \qquad \text{(candidate update)} \tag{7.40}$$

$$\mathbf{m}'_j = (1 - \mathbf{u}_j) \odot \mathbf{m}_j + \mathbf{u}_j \odot \tilde{\mathbf{m}}_j, \qquad \text{(gated update)} \tag{7.41}$$

where $\mathbf{h}_{\text{attn}} \in \mathbb{R}^{T' \times d}$ is the output of Strand Attention. The read operation retrieves information from memory via attention:

$$\mathbf{v}_{\text{epi}} = \text{softmax}(\mathbf{h}_{\text{attn}} \mathbf{W}_q^{\text{epi}} (\mathbf{M} \mathbf{W}_k^{\text{epi}})^\top) \mathbf{M} \mathbf{W}_v^{\text{epi}}, \tag{7.42}$$

where $\mathbf{M} = [\mathbf{m}_1; \ldots; \mathbf{m}_M] \in \mathbb{R}^{M \times d_m}$.

The complexity of epigenetic memory operations is $O(T' \cdot M \cdot d_m)$, which is linear in sequence length when M is fixed.

Allosteric Broadcast: $O(Tk)$ Signal Distribution

Allosteric regulation, in biology, is the modulation of a protein's activity by binding at a site other than the active site. In DOGMA, allosteric broadcast selects k "broadcast sites" and distributes their signals to all other positions:

$$\text{sites} = \text{TopK}(\mathbf{h}_{\text{attn}}\mathbf{w}_{\text{select}},\ k), \tag{7.43}$$

$$\mathbf{s}_{\text{broadcast}} = \frac{1}{k}\sum_{j\in\text{sites}} \mathbf{h}_{\text{attn},j}, \tag{7.44}$$

$$\mathbf{h}_{\text{final},t} = \mathbf{h}_{\text{attn},t} + \mathbf{v}_{\text{epi},t} + \text{MLP}(\mathbf{s}_{\text{broadcast}}), \tag{7.45}$$

where $\mathbf{w}_{\text{select}} \in \mathbb{R}^d$ is a learned selection vector and TopK selects the k positions with highest selection scores. The broadcast signal $\mathbf{s}_{\text{broadcast}}$ is added to all positions, providing global context from the most salient positions.

The complexity is $O(T' \cdot k \cdot d)$ for computing the broadcast, where $k \ll T'$.

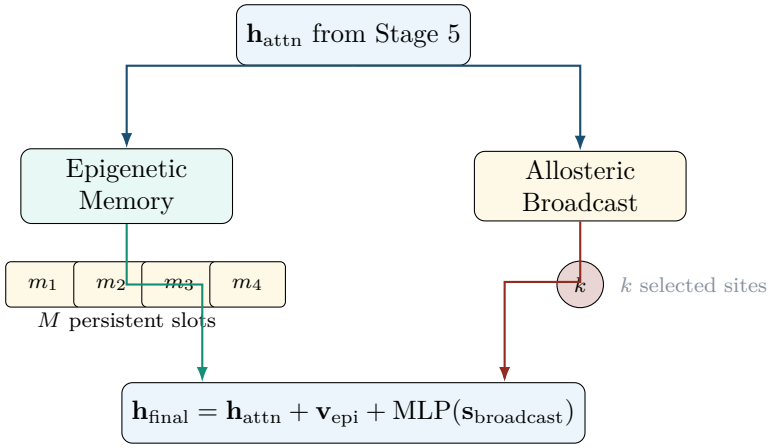


Figure 7.6: Stage 6: Epigenetic Memory ($O(TM)$) provides persistent state; Allosteric Broadcast ($O(Tk)$) provides fast global context from k selected sites. Their outputs are combined with the attention output.

Epigenetics and Allosteric Regulation

In biology, epigenetic modifications (DNA methylation, histone acetylation) create persistent, heritable changes in gene expression without altering the DNA sequence itself. Allosteric regulation is a fast, reversible mechanism where binding at one site affects activity at a distant site. DOGMA separates these two timescales: epigenetic memory changes slowly across many forward passes (akin to long-term memory), while allosteric broadcast operates within a single forward pass (akin to working memory or attention).

7.4 The Expression Folder

After the six-stage pipeline produces the final hidden representation $\mathbf{h}_{\text{final}}$, DOGMA must convert this internal representation into the output space—a process we call the *Expression Folder*, by analogy with protein folding.

In protein folding, a linear amino acid chain (the primary structure) folds into a complex three-dimensional structure (the tertiary structure) that determines the protein's function. Similarly, the Expression Folder transforms the linear sequence of hidden states into a structured output representation.

$$\Phi = \text{ExpressionFolder}(\mathbf{h}_{\text{final}}) = \text{MLP}_{\text{fold}}(\text{LayerNorm}(\mathbf{h}_{\text{final}})), \tag{7.46}$$

where MLP_{fold} is a two-layer feed-forward network with GELU activation:

$$\text{MLP}_{\text{fold}}(\mathbf{z}) = \mathbf{W}_2 \cdot \text{GELU}(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1) + \mathbf{b}_2. \tag{7.47}$$

The output Φ is not text. It is a *phenotype representation*—a structured semantic object that can be *realized* into different modalities:

- **Natural language:** A language head applies a linear projection followed by softmax to produce token probabilities.
- **Code:** A syntax-aware head produces abstract syntax tree nodes.
- **JSON/Schema:** A structured output head generates key-value pairs conforming to a schema.
- **Action plans:** A tool-use head produces function calls and arguments.
- **3D structure:** A TetraMesh head produces spatial coordinates (Chapter 10).

Phenotype Before Output

In a standard LLM, the model generates text directly from its hidden state. There is no explicit representation of "what the model intends to do." In DOGMA, the phenotype Φ is this explicit representation. It can be:

- Inspected for correctness before realization.
- Verified against safety constraints.
- Realized in multiple modalities from the same plan.
- Stored for future reference or comparison.

This is the computational analogue of protein folding: a linear sequence (transcript) is transformed into a three-dimensional functional structure (phenotype) through a deterministic but complex process.

7.5 Full Architecture Diagram

Figure 7.7 presents the complete DOGMA architecture, showing both the six-stage pipeline (left) and the detailed DNA Computing Block structure (right).

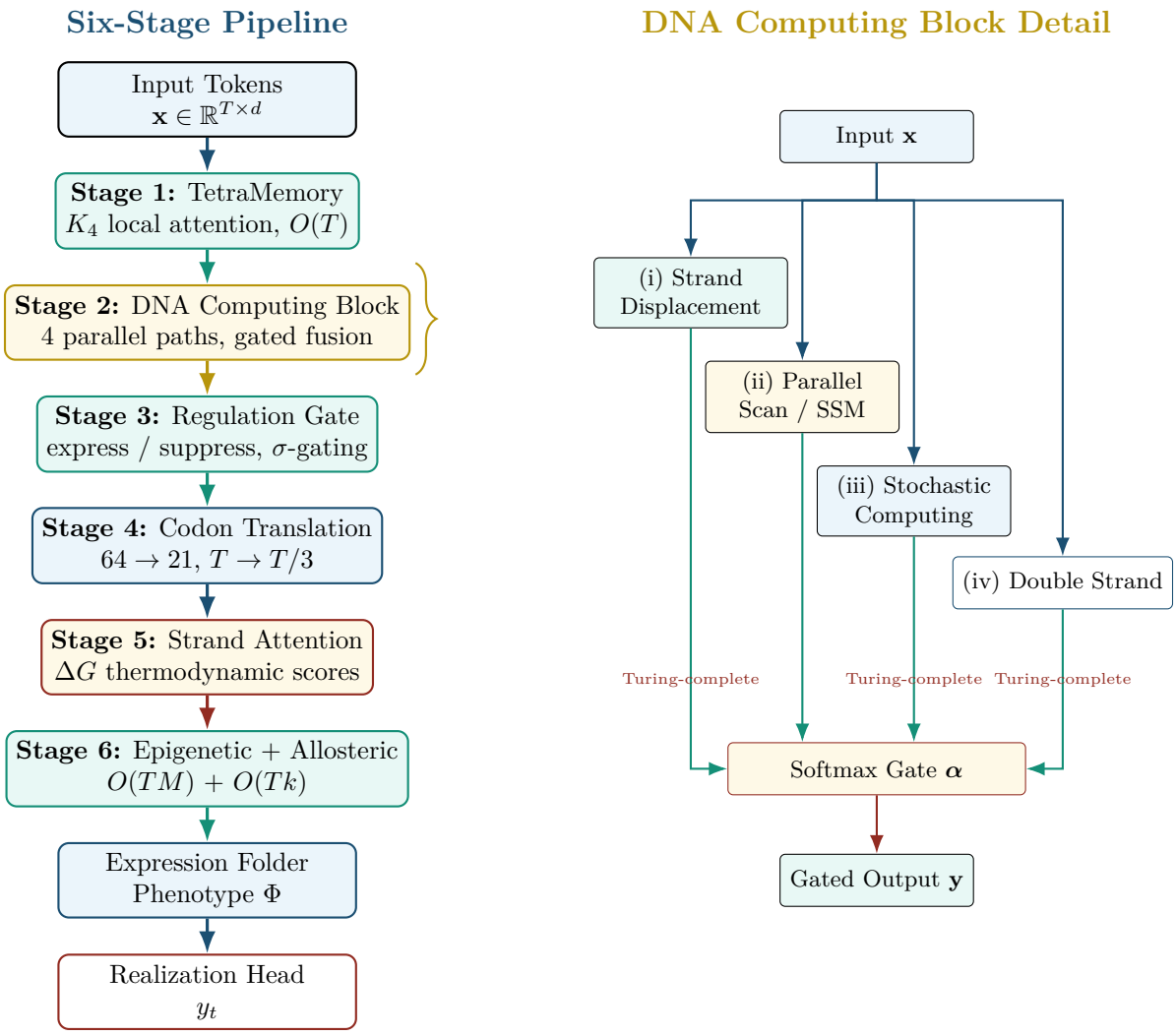


Figure 7.7: Complete DOGMA architecture. **Left:** The six-stage pipeline from input tokens to output realization, passing through TetraMemory, the DNA Computing Block, Regulation Gate, Codon Translation, Strand Attention, and Epigenetic+Allosteric mechanisms, followed by the Expression Folder. **Right:** Detailed view of the DNA Computing Block showing four parallel paths merged by a learned softmax gate. Three of the four paths are independently Turing-complete.

7.6 The DogmaBlock: Complete Forward Pass

Algorithm 1 presents the complete forward pass of a single DogmaBlock as pseudocode.

Algorithm 1 DogmaBlock Forward Pass

Require: Input $\mathbf{x} \in \mathbb{R}^{T \times d}$, context $\mathbf{c} \in \mathbb{R}^{d_c}$, memory bank $\mathbf{M} \in \mathbb{R}^{M \times d_m}$
Ensure: Output $\mathbf{y} \in \mathbb{R}^{T' \times d}$, updated memory $\mathbf{M}'$

// Stage 1: TetraMemory
Reshape $\mathbf{x}$ into tetrahedra: $\mathbf{X}_k \leftarrow \mathbf{x}[4k{:}4k{+}4]$ for $k = 0, \ldots, \lfloor T/4 \rfloor - 1$
$\mathbf{Y}_k \leftarrow \text{LocalAttn}_{K_4}(\mathbf{X}_k)$ for each tetrahedron ▷ 4×4 attention
$\mathbf{h}_1 \leftarrow \text{FacePropagate}(\mathbf{Y}_0, \mathbf{Y}_1, \ldots)$ ▷ $O(T \cdot d)$

// Stage 2: DNA Computing Block (4 parallel paths)
$\mathbf{p}_{\text{SD}} \leftarrow \text{StrandDisplacement}(\mathbf{h}_1)$ ▷ Toehold gating + causal conv
$\mathbf{p}_{\text{SSM}} \leftarrow \text{ParallelScan}(\mathbf{h}_1)$ ▷ Selective SSM, $O(T \log T)$
$\mathbf{p}_{\text{SC}} \leftarrow \text{StochasticCompute}(\mathbf{h}_1)$ ▷ Bernoulli sampling + bitstream ops
$\mathbf{p}_{\text{DS}} \leftarrow \text{DoubleStrand}(\mathbf{h}_1)$ ▷ Forward + reverse complement
$\boldsymbol{\alpha} \leftarrow \text{softmax}(\mathbf{W}_g \mathbf{h}_1 + \mathbf{b}_g)$ ▷ Per-position gate weights
$\mathbf{h}_2 \leftarrow \sum_p \alpha_p \cdot \mathbf{p}_p$ ▷ Gated fusion

// Stage 3: Regulation Gate
$\mathbf{g}_e \leftarrow \sigma(\mathbf{h}_2 \mathbf{W}_e + \mathbf{c}\mathbf{U}_e + \mathbf{b}_e)$ ▷ Express signal
$\mathbf{g}_s \leftarrow \sigma(\mathbf{h}_2 \mathbf{W}_s + \mathbf{c}\mathbf{U}_s + \mathbf{b}_s)$ ▷ Suppress signal
$\mathbf{h}_3 \leftarrow \mathbf{h}_2 \odot \mathbf{g}_e \odot (1 - \mathbf{g}_s)$ ▷ Element-wise gating

// Stage 4: Codon Translation
Group triples: $\mathbf{c}_k \leftarrow [\mathbf{h}_{3,3k-2} \| \mathbf{h}_{3,3k-1} \| \mathbf{h}_{3,3k}]$
$\mathbf{h}_4 \leftarrow \text{CodonTranslate}(\mathbf{c}_k)$ for $k = 1, \ldots, \lfloor T/3 \rfloor$ ▷ $64 \to 21$ soft mapping, $T \to T/3$

// Stage 5: Strand Attention
$\mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow \mathbf{h}_4 \mathbf{W}_Q,\ \mathbf{h}_4 \mathbf{W}_K,\ \mathbf{h}_4 \mathbf{W}_V$
$\Delta G_{ij} \leftarrow \mathbf{q}_i^\top \mathbf{W}_{\Delta H} \mathbf{k}_j - \tau_{\text{phys}} \cdot \mathbf{q}_i^\top \mathbf{W}_{\Delta S} \mathbf{k}_j + \beta(\tau_i, \tau_j)$
$\mathbf{h}_5 \leftarrow \text{softmax}(\Delta G / \tau) \cdot \mathbf{V}$ ▷ $O(T'^2 d)$ where $T' = T/3$

// Stage 6: Epigenetic + Allosteric
$\mathbf{v}_{\text{epi}} \leftarrow \text{EpiRead}(\mathbf{h}_5, \mathbf{M})$; $\mathbf{M}' \leftarrow \text{EpiUpdate}(\mathbf{h}_5, \mathbf{M})$ ▷ $O(T'M)$
sites $\leftarrow \text{TopK}(\mathbf{h}_5 \mathbf{w}_{\text{sel}}, k)$; $\mathbf{s} \leftarrow \frac{1}{k} \sum_{j \in \text{sites}} \mathbf{h}_{5,j}$ ▷ $O(T'k)$
$\mathbf{h}_6 \leftarrow \mathbf{h}_5 + \mathbf{v}_{\text{epi}} + \text{MLP}(\mathbf{s})$

// Expression Folder
$\mathbf{y} \leftarrow \text{MLP}_{\text{fold}}(\text{LayerNorm}(\mathbf{h}_6))$
return $\mathbf{y}, \mathbf{M}'$

7.7 Architecture Comparison: DOGMA vs. Alternatives

Table 7.5 provides a detailed comparison of DOGMA with three major competing architectures: the Transformer [Vaswani et al., 2017a], Mamba (a state-space model) [Gu and Dao, 2023], and RWKV (a linear-attention RNN).

Table 7.5: Detailed architecture comparison: DOGMA vs. Transformer vs. Mamba vs. RWKV.

Property	Transformer	Mamba	RWKV	DOGMA
Time complexity (per layer)	$O(T^2 d)$	$O(Td)$	$O(Td)$	$O(Td)$ to $O(T'^2 d)$ ($T' = T/3$)
Space complexity	$O(T^2 + Td)$	$O(Td)$	$O(Td)$	$O(T'd + TM)$
Turing-complete?	Yes (with pos. enc.)	Debated	No (finite state)	Yes (3 of 4 paths)
Parallel paths	1 (attention)	1 (SSM)	1 (linear attn)	4 (gated)
Persistent memory	None (context only)	Hidden state	Hidden state	Epigenetic bank (M slots)
Bidirectional	Full	No (causal)	No (causal)	Yes (Double Strand path)
Regulation / gating	None	Selective SSM	Time-decay	Express/suppress gate
Biological basis	None	Inspired by control theory	None	DNA computing, Central Dogma
Information compression	None	Via state bottleneck	Via state bottleneck	Codon Translation ($3:1$)
Attention mechanism	Dot-product	None	Linear attention	ΔG thermodynamic
Global context	Full attention	Recurrent propagation	Recurrent propagation	Allosteric broadcast (k sites)

7.7.1 Complexity Summary

To summarize the per-layer complexity of a single DogmaBlock:

$$\text{Stage 1 (TetraMemory):} \quad O(T \cdot d), \tag{7.48}$$

$$\text{Stage 2 (DNA Block):} \quad O(T \cdot d) \quad \text{(Parallel Scan dominates)}, \tag{7.49}$$

$$\text{Stage 3 (Regulation):} \quad O(T \cdot d), \tag{7.50}$$

$$\text{Stage 4 (Codon Translation):} \quad O(T \cdot d), \tag{7.51}$$

$$\text{Stage 5 (Strand Attention):} \quad O(T'^2 \cdot d) \quad \text{where } T' = T/3, \tag{7.52}$$

$$\text{Stage 6 (Epi + Allo):} \quad O(T' \cdot M \cdot d_m + T' \cdot k \cdot d). \tag{7.53}$$

The bottleneck is Stage 5 (Strand Attention) at $O(T'^2 d) = O(T^2 d/9)$. This is $9\times$ cheaper than standard self-attention on the full sequence, and further reduced when regulation sparsity is applied. When $T' \leq T/3$ after regulation-induced sparsity, the effective complexity can approach linear.

7.8 Worked Example: Tracing Data Through All Six Stages

To make the pipeline concrete, let us trace a small example through all six stages. Suppose we have an input sequence of $T = 12$ tokens with embedding dimension $d = 8$ (vastly simplified from

production dimensions, but sufficient to illustrate the data flow).

Example 7.1 (Tracing 12 Tokens Through DOGMA)**. Input:** $\mathbf{x} \in \mathbb{R}^{12\times 8}$ (12 tokens, 8-dimensional embeddings).

Stage 1: TetraMemory. Partition into $12/4 = 3$ tetrahedra:

$$\begin{aligned}\Delta_0 &= (x_1, x_2, x_3, x_4),\\ \Delta_1 &= (x_3, x_4, x_5, x_6), \quad \text{(shares face } \{x_3, x_4\} \text{ with } \Delta_0)\\ \Delta_2 &= (x_5, x_6, x_7, x_8). \quad \text{(shares face } \{x_5, x_6\} \text{ with } \Delta_1)\end{aligned}$$

Wait—with 12 tokens and groups of 4, we get tetrahedra with overlapping boundaries: $\Delta_0 = (x_1, x_2, x_3, x_4)$, $\Delta_1 = (x_4, x_5, x_6, x_7)$, $\Delta_2 = (x_7, x_8, x_9, x_{10})$, $\Delta_3 = (x_{10}, x_{11}, x_{12})$ (padded).

Within each tetrahedron, a 4×4 attention matrix is computed. After face propagation, the output shape is still $\mathbb{R}^{12\times 8}$, but each token now carries information from its local tetrahedral neighborhood.

Output shape: $\mathbf{h}_1 \in \mathbb{R}^{12\times 8}$.

Stage 2: DNA Computing Block. Four parallel paths process $\mathbf{h}_1$:

- Strand Displacement: $\mathbf{p}_{\text{SD}} \in \mathbb{R}^{12\times 8}$ (toehold gating selects 8 of 12 positions).
- Parallel Scan: $\mathbf{p}_{\text{SSM}} \in \mathbb{R}^{12\times 8}$ (all positions, linear recurrence).
- Stochastic: $\mathbf{p}_{\text{SC}} \in \mathbb{R}^{12\times 8}$ (mean over $S = 16$ samples).
- Double Strand: $\mathbf{p}_{\text{DS}} \in \mathbb{R}^{12\times 8}$ (forward + reverse complement).

Gate weights: $\boldsymbol{\alpha} = (0.35, 0.30, 0.10, 0.25)$ (example values).

Output shape: $\mathbf{h}_2 \in \mathbb{R}^{12\times 8}$.

Stage 3: Regulation Gate. Context embedding $\mathbf{c} \in \mathbb{R}^{16}$ encodes the task. The express/suppress gates produce:

$$\begin{aligned}\mathbf{g}_e &\approx (0.9, 0.8, 0.95, 0.1, 0.05, 0.7, 0.85, 0.9, 0.6, 0.1, 0.8, 0.75),\\ \mathbf{g}_s &\approx (0.1, 0.1, 0.05, 0.8, 0.9, 0.2, 0.1, 0.05, 0.3, 0.85, 0.1, 0.15).\end{aligned}$$

Positions 4, 5, and 10 are effectively suppressed ($r \approx 0$). Only 9 of 12 positions survive with significant activation.

Output shape: $\mathbf{h}_3 \in \mathbb{R}^{12\times 8}$ (but 3 positions near-zero).

Stage 4: Codon Translation. Group into $\lfloor 12/3 \rfloor = 4$ codons:

$$\begin{aligned}\text{Codon } 1 &= [\mathbf{h}_{3,1} \| \mathbf{h}_{3,2} \| \mathbf{h}_{3,3}] \in \mathbb{R}^{24},\\ \text{Codon } 2 &= [\mathbf{h}_{3,4} \| \mathbf{h}_{3,5} \| \mathbf{h}_{3,6}] \in \mathbb{R}^{24},\\ \text{Codon } 3 &= [\mathbf{h}_{3,7} \| \mathbf{h}_{3,8} \| \mathbf{h}_{3,9}] \in \mathbb{R}^{24},\\ \text{Codon } 4 &= [\mathbf{h}_{3,10} \| \mathbf{h}_{3,11} \| \mathbf{h}_{3,12}] \in \mathbb{R}^{24}.\end{aligned}$$

Each codon is translated via the soft $64 \to 21$ mapping.

Output shape: $\mathbf{h}_4 \in \mathbb{R}^{4\times 8}$. Sequence length reduced from 12 to 4.

Stage 5: Strand Attention. Compute thermodynamic attention on $T' = 4$ positions: a 4×4 attention matrix using ΔG scores.

Output shape: $\mathbf{h}_5 \in \mathbb{R}^{4\times 8}$.

Stage 6: Epigenetic + Allosteric. With $M = 3$ memory slots and $k = 2$ broadcast sites:

- Epigenetic: read/write from 3 memory slots, $O(4 \cdot 3 \cdot 8) = O(96)$ operations.
- Allosteric: select top-2 positions, broadcast their mean to all 4 positions.

Output shape: $\mathbf{h}_6 \in \mathbb{R}^{4\times 8}$.

Expression Folder. Apply LayerNorm and MLP to produce the phenotype.

Final output shape: $\Phi \in \mathbb{R}^{4\times 8}$. From 12 input tokens, we have 4 phenotype vectors ready for realization.

Table 7.6: Tensor shapes through the six-stage pipeline for the worked example ($T = 12$, $d = 8$).

Stage	Operation	Input Shape	Output Shape	Key Change
—	Input embedding	—	12×8	—
1	TetraMemory	12×8	12×8	Local K_4 interactions
2	DNA Computing Block	12×8	12×8	4-path gated fusion
3	Regulation Gate	12×8	12×8	3 positions suppressed
4	Codon Translation	12×8	4×8	$3:1$ compression
5	Strand Attention	4×8	4×8	ΔG attention
6	Epi + Allosteric	4×8	4×8	Memory read/write
—	Expression Folder	4×8	4×8	Phenotype

7.9 The Evolutor State

The complete state of a DOGMA system at time t is captured by the *Evolutor state*:

Definition 7.5 (Evolutor State)**.** The *Evolutor state* at time t is the tuple:

$$\mathcal{E}_t = (\Gamma_t, H_t, \mathcal{A}_t, \mathcal{L}_t), \tag{7.54}$$

where:

- Γ_t is the current genome.
- H_t is the Tera hidden regime state (Chapter 9).
- $\mathcal{A}_t$ is the HelixHash archive of structural signatures (Chapter 8).
- $\mathcal{L}_t$ is the lineage history (record of evolutionary modifications).

The Evolutor state encapsulates everything needed to reproduce the system's behavior and continue its evolution. It is the DOGMA analogue of a cell's complete state: genome + epigenome + cellular memory.

7.10 Why DOGMA Is Not Just a Modified Transformer

It is tempting to view DOGMA as "a transformer with extra steps." This misses the fundamental architectural difference. Consider three levels of distinction:

1. **Ontological:** In a transformer, the primary computational object is the token sequence. In DOGMA, the primary computational object is the *genome*—a persistent, typed, structured state from which tokens are merely one possible output.

2. **Control flow:** In a transformer, all tokens interact uniformly through attention. In DOGMA, interaction is *regulated*—controlled by context-dependent activation maps that determine which regions participate.

3. **Evolutionary:** A transformer's parameters are fixed after training. A DOGMA genome can be *evolved*—modified through mutation, selection, and synthesis—enabling open-ended improvement without full retraining.

The relationship between DOGMA and transformers is analogous to the relationship between cells and chemical reactions. A cell *contains* chemical reactions, but a cell is not merely "chemistry with extra steps." The organizational structure—membranes, organelles, regulatory networks, genetic programs—is what makes a cell fundamentally different from a random mixture of the same chemicals.

Similarly, DOGMA may *contain* attention mechanisms (within its Strand Attention stage or DNA Computing Block modules), but the organizational structure—TetraMemory, four-path DNA computing, regulation gating, codon translation, thermodynamic attention, epigenetic memory—is what makes it architecturally distinct.

7.11 Exercises

7.1. [**Fundamentals**] For each of the six DOGMA stages (TetraMemory, DNA Computing Block, Regulation Gate, Codon Translation, Strand Attention, Epigenetic+Allosteric), identify the corresponding step in the biological Central Dogma and explain the mapping.

7.2. [**Analysis**] Consider a DOGMA system processing a sequence of $T = 4096$ tokens with embedding dimension $d = 512$.

(a) How many tetrahedra does Stage 1 create?

(b) What is the sequence length after Stage 4 (Codon Translation)?

(c) Compare the cost of Stage 5 Strand Attention (on the compressed sequence) with standard self-attention on the original 4096 tokens.

7.3. [**Design**] The DNA Computing Block uses a learned softmax gate to merge four path outputs. Design an experiment to test whether all four paths are necessary. Describe what you would measure when ablating each path individually, and predict which path's removal would cause the largest performance drop for (a) a language modeling task, (b) a protein structure prediction task.

7.4. [**Coding**] Implement the four-equation Regulation Gate defined above in PyTorch. Create a test that verifies:

(a) The output has the same shape as the input.

(b) When the suppress gate outputs all ones, the regulated output is all zeros.

(c) When both gates output 0.5, the net regulation is $0.5 \times 0.5 = 0.25$ per element.

7.5. **[Theory]** Prove that if the regulation stage produces an activation map with at most k nonzero entries (hard sparsity), then the computational cost of Stage 5 Strand Attention is $O(k'^2 d)$ where $k' = k/3$, rather than $O((T/3)^2 d)$.

7.6. **[Research]** Read about Mixture-of-Experts (MoE) architectures [Brown et al., 2020]. Compare the MoE routing mechanism with DOGMA's Regulation Gate. What are the key similarities and differences? Which approach offers better interpretability, and why?

7.7. **[Analysis]** The Codon Translation stage uses a $64 \to 21$ soft mapping initialized from the biological genetic code. Discuss:

(a) Why is degeneracy (multiple codons $\to$ same amino acid) beneficial for robustness?

(b) Should the codon embeddings be frozen at their biological initialization or fine-tuned? Argue both sides.

7.8. **[Advanced]** The Evolutor state $\mathcal{E}_t = (\Gamma_t, H_t, \mathcal{A}_t, \mathcal{L}_t)$ maintains lineage history $\mathcal{L}_t$. Discuss the computational and memory tradeoffs of maintaining full lineage vs. compressed lineage (keeping only the most recent m checkpoints). Under what conditions would compressed lineage lose important information?

7.9. **[Worked Example]** Trace a sequence of $T = 24$ tokens through the six-stage pipeline, showing tensor shapes at each stage. How many codons are produced? What is the final phenotype length?

7.10. **[Comparison]** Using Table 7.5, explain why DOGMA is better suited than Mamba for tasks requiring bidirectional context, and why Mamba might be preferred for pure autoregressive generation at very long sequence lengths.

7.12 Key Takeaways

Key Takeaways

- DOGMA's six-stage pipeline processes data through TetraMemory → DNA Computing Block → Regulation Gate → Codon Translation → Strand Attention → Epigenetic+Allosteric, followed by an Expression Folder that produces the phenotype.
- The genomic base $b_i = (\sigma_i, \tau_i, \rho_i, \omega_i)$ is DOGMA's atomic unit, carrying a base embedding, type tag, compatibility vector, and weight.
- Four base types (A, C, G, T) provide functional roles (Activation, Composition, Generation, Termination) with complementary pairing and Klein four-group symmetry.
- TetraMemory achieves $O(T)$ complexity by computing K_4 local attention within groups of 4 tokens and propagating information through shared faces.
- The DNA Computing Block runs four parallel paths (Strand Displacement, Parallel Scan, Stochastic Computing, Double Strand), three of which are independently Turing-complete, merged by a learned softmax gate.
- The Regulation Gate implements express/suppress gating analogous to gene regulation, producing sparse activation patterns that reduce downstream computation.
- Codon Translation compresses the sequence $3:1$ through a differentiable $64 \to 21$ mapping initialized from the biological genetic code.
- Strand Attention uses thermodynamic ΔG binding energies (inspired by SantaLucia nearest-neighbor parameters) instead of dot-product scores.
- Epigenetic memory ($O(TM)$) provides persistent state across time steps; allosteric broadcast ($O(Tk)$) provides fast global context from k selected sites.
- The Expression Folder converts the final representation into a phenotype that can be realized into text, code, structured data, or other modalities.
- DOGMA is not a modified transformer but a fundamentally different organizational paradigm: genome-first, regulation-controlled, evolution-capable, with built-in redundancy through multiple Turing-complete paths.

Suggested Reading

- Crick [1970]: The original Central Dogma, DOGMA's biological inspiration.
- Vaswani et al. [2017a]: The transformer architecture, for comparison.
- Gu and Dao [2023]: State-space models, related to DOGMA's ParallelScan path.
- Zhang and Seelig [2011]: DNA strand displacement, the basis for DOGMA's Strand Displacement path.

- ENCODE Project Consortium [2012]: The ENCODE project on gene regulation and expression sparsity.
- Chapters 8–11 of this book: detailed treatments of DOGMA's subsystems.

Chapter 8

HelixHash — Structural Representation of Computational Genomes

The double helix is not merely a shape. It is a commitment to redundancy, error correction, and bidirectional reading—properties any serious computational substrate must share.

— DOGMA Design Manifesto

Chapter 7 introduced the DOGMA pipeline and its genomic bases. But a genome is more than a sequence of bases—it has *structure*. In molecular biology, structure determines function: the double helix enables replication, the codon frame enables translation, and chromatin folding enables regulation. In DOGMA, the analogous structural analysis framework is **HelixHash**.

HelixHash serves three roles within the DOGMA architecture. First, it provides a *double-helix representation* that encodes every computational genome as a pair of complementary strands, enabling bidirectional analysis. Second, it extracts *structural signatures*—motifs, sketches, and bond matrices—that characterize a genome's organization independently of its semantic content. Third, it maintains a *novelty archive* that drives evolutionary diversity, ensuring that the synthesis stage (Chapter 7) explores broadly rather than collapsing to a narrow optimum.

This chapter develops HelixHash from first principles, drawing on techniques from bioinformatics (k-mer analysis), NLP (n-gram models), and streaming algorithms (locality-sensitive hashing). We formalize each component, analyze its complexity, and show how the pieces compose into a unified structural embedding suitable for foundation model training.

8.1 The Double-Helix as a Data Structure

Before we define HelixHash's computational representation, let us carefully review the biological structure that inspires it. DNA is composed of two polynucleotide chains wound around a common axis, forming a right-handed helix. The two chains run in *opposite* directions—they are *antiparallel*. One strand runs $5' \to 3'$ (the *sense* or *coding* strand), while its partner runs $3' \to 5'$ (the *antisense* or *template* strand). The strands are held together by hydrogen bonds between complementary bases: adenine (A) pairs with thymine (T) via two hydrogen bonds, and guanine (G) pairs with cytosine (C) via three hydrogen bonds.

8.1.1 Step-by-Step Construction of the Double Helix

We now build DOGMA's computational double helix in four steps.

Step 1: The Sense Strand (Forward Strand). Given a computational genome G consisting of n fragment embeddings, the *sense strand* is simply the genome read in its natural order:

$$F(G) = [f_1,\ f_2,\ \ldots,\ f_n]. \quad (5' \to 3' \text{ direction}) \tag{8.1}$$

Each $f_i \in \mathbb{R}^d$ carries both a *type tag* $\tau_i \in \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}$ and a *semantic embedding* $\sigma_i \in \mathbb{R}^{d-1}$. The type tag determines how the fragment participates in bonding; the embedding carries content.

Step 2: The Complementation Operator. To form the antisense strand we need a function that maps every fragment to its "complement." In biology, this is dictated by Watson–Crick rules: $\mathrm{A} \leftrightarrow \mathrm{T}$ and $\mathrm{G} \leftrightarrow \mathrm{C}$. In DOGMA, the complementation operator is a learned linear map:

Definition 8.1 (Complementation Operator)**.** The *complementation operator* $\mathrm{rev} : \mathbb{R}^d \to \mathbb{R}^d$ satisfies two constraints:

1. **Type complementarity:** $\tau(\mathrm{rev}(f)) = \overline{\tau(f)}$, where $\overline{\mathsf{A}} = \mathsf{T}$, $\overline{\mathsf{T}} = \mathsf{A}$, $\overline{\mathsf{G}} = \mathsf{C}$, $\overline{\mathsf{C}} = \mathsf{G}$.

2. **Approximate involution:** $\|\mathrm{rev}(\mathrm{rev}(f)) - f\|_2 \leq \epsilon$ for a small tolerance $\epsilon > 0$.

In practice, rev is parameterized as a block-diagonal matrix that acts on the type tag deterministically and on the semantic embedding through a learned orthogonal transform.

The involution constraint ensures that complementing a strand twice recovers (approximately) the original, mirroring the biological reality that the complement of the complement of a DNA sequence is the original sequence.

Step 3: The Antisense Strand (Reverse Strand). The antisense strand is constructed by complementing every fragment and *reversing* the order:

Definition 8.2 (Forward and Reverse Strands)**.** Let $G = (f_1, f_2, \ldots, f_n)$ be a computational genome represented as a sequence of n fragment embeddings. The *forward strand* is:

$$F(G) = [f_1, f_2, \ldots, f_n], \tag{8.2}$$

and the *reverse strand* is:

$$R(G) = [\mathrm{rev}(f_n), \mathrm{rev}(f_{n-1}), \ldots, \mathrm{rev}(f_1)], \tag{8.3}$$

where $\mathrm{rev}(\cdot)$ is the complementation operator from Definition 8.1.

The reversal is essential. In biology, the antiparallel orientation means that if you read one strand left-to-right ($5' \to 3'$), you must read the other strand right-to-left. This asymmetry is what enables bidirectional transcription: genes on the sense strand are transcribed in one direction, while genes on the antisense strand are transcribed in the opposite direction.

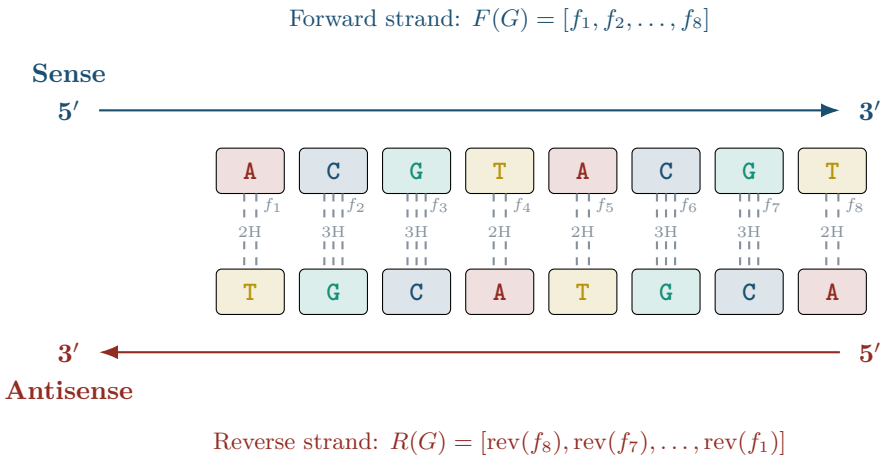


Figure 8.1: The double-helix representation of a computational genome with type sequence ACGTACGT. The sense strand runs 5′ →3′ (left to right); the antisense strand runs 3′ →5′ (right to left). Dashed lines represent hydrogen bonds: A–T pairs have 2 bonds, G–C pairs have 3 bonds. Each fragment f_i is paired with its complement $\mathrm{rev}(f_{n+1-i})$ on the opposite strand.

Step 4: Base Pairing. The two strands are "bonded" position-by-position. Position i on the forward strand pairs with position $n + 1 - i$ on the reverse strand. The bond strength at each position is determined by the type tags of the paired fragments, as formalized in the bond matrix (Section 8.2).

Why Two Strands?

In biology, the double-stranded structure of DNA serves three purposes: (i) *redundancy* for error correction—damage to one strand can be repaired using the other as a template; (ii) *bidirectional transcription*—genes on opposite strands can be read independently; and (iii) *structural stability*—base pairing provides thermodynamic stability to the molecule. In DOGMA, the double-helix representation inherits all three advantages:

1. **Robustness:** If a fragment embedding is corrupted (e.g., by mutation during evolutionary training), the reverse strand provides a recovery signal.
2. **Bidirectional context:** Forward and reverse strands capture different directional dependencies, analogous to bidirectional RNNs but with an explicit structural motivation.
3. **Structural fingerprinting:** The relationship between a genome and its reverse complement encodes structural information that single-strand analysis cannot capture.

8.1.2 Watson–Crick Complementarity as a Hash Function

An elegant way to view Watson–Crick complementarity is as a *hash function*. Define the complementarity map $\overline{(\cdot)} : \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\} \to \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}$ by:

$$\overline{\mathsf{A}} = \mathsf{T}, \quad \overline{\mathsf{T}} = \mathsf{A}, \quad \overline{\mathsf{G}} = \mathsf{C}, \quad \overline{\mathsf{C}} = \mathsf{G}. \tag{8.4}$$

This map is an involution ($\overline{\overline{x}} = x$) and a fixed-point-free permutation (no base is its own

complement). For a type sequence $\mathbf{t} = (t_1, t_2, \ldots, t_k)$, the reverse-complement hash is:

$$\mathrm{RC}(\mathbf{t}) = (\overline{t_k},\ \overline{t_{k-1}},\ \ldots,\ \overline{t_1}). \tag{8.5}$$

A type sequence and its reverse complement always hash to distinct values (since the map is fixed-point-free and reversal breaks palindromic symmetry for most sequences). This provides a natural way to canonicalize motifs: we can define the *canonical form* of a type sequence as the lexicographically smaller of $\mathbf{t}$ and $\mathrm{RC}(\mathbf{t})$. This halves the effective motif vocabulary and enforces strand symmetry in all downstream analyses.

Example 8.1 (Canonical Motif Forms)**.** Consider the 3-motif type sequence $\mathbf{t} = \mathsf{ACG}$. Its reverse complement is $\mathrm{RC}(\mathsf{ACG}) = \mathsf{CGT}$. Since $\mathsf{ACG} < \mathsf{CGT}$ lexicographically, the canonical form is ACG. Now consider $\mathbf{t} = \mathsf{GCA}$. Its reverse complement is $\mathrm{RC}(\mathsf{GCA}) = \mathsf{TGC}$. The canonical form is GCA. Notice that ACG and CGT share the same canonical form, reflecting that they represent the same structural motif read from opposite strands.

8.1.3 Strand Alignment Score

Given two genomes G_1 and G_2, we can measure their structural similarity by comparing strand representations:

$$S_{\text{strand}}(G_1, G_2) = \frac{1}{2}\big[\cos(F(G_1), F(G_2)) + \cos(R(G_1), R(G_2))\big], \tag{8.6}$$

where $\cos(\cdot, \cdot)$ denotes cosine similarity computed over averaged fragment embeddings. The strand alignment score is symmetric and bounded in $[-1, 1]$, with values near 1 indicating structurally similar genomes.

8.2 The Bond Matrix

The bond matrix is a central data structure in HelixHash that captures how strongly each pair of positions in a genome interacts. Its design is inspired by the hydrogen bonding patterns of biological DNA.

8.2.1 The 4×4 Hydrogen Bond Matrix

In molecular biology, the strength of a base pair is determined by the number of hydrogen bonds. We encode this in a *base interaction matrix*:

Definition 8.3 (Base Interaction Matrix)**.** The *base interaction matrix* $\mathbf{H} \in \mathbb{R}^{4\times 4}$ assigns a bond strength to every possible pair of base types:

$$\mathbf{H} = \begin{pmatrix} & \mathsf{A} & \mathsf{T} & \mathsf{G} & \mathsf{C} \\ \mathsf{A} & 0 & 2 & 0 & 0 \\ \mathsf{T} & 2 & 0 & 0 & 0 \\ \mathsf{G} & 0 & 0 & 0 & 3 \\ \mathsf{C} & 0 & 0 & 3 & 0 \end{pmatrix}. \tag{8.7}$$

Complementary pairs (A–T, T–A) receive a bond strength of 2 (two hydrogen bonds), while G–C and C–G pairs receive 3 (three hydrogen bonds). All non-complementary pairs receive 0.

Remark 8.1. The asymmetry between A–T (2 bonds) and G–C (3 bonds) has profound consequences. GC-rich regions of a genome are thermodynamically more stable than AT-rich regions. In DOGMA, this translates to stronger structural coherence in regions dominated by G and C type tags, making these regions more resistant to disruptive mutations.

Table 8.1: The 4×4 hydrogen bond matrix $\mathbf{H}$. Entries show the number of hydrogen bonds for each base pair. Non-zero entries occur only for Watson–Crick complementary pairs.

	A	**T**	**G**	**C**
A	0	2	0	0
T	2	0	0	0
G	0	0	0	3
C	0	0	3	0

8.2.2 Bond Energies as Attention Weights

A key insight of HelixHash is that the hydrogen bond strengths in $\mathbf{H}$ can serve as *natural attention weights*. In a transformer, the attention weight between positions i and j is a learned function of the content at those positions. In HelixHash, we have a biologically grounded alternative: the bond strength between two fragments is determined by their type tags.

Define the *type-based attention weight* between fragments f_i and f_j as:

$$a_{ij}^{\text{type}} = \frac{\mathbf{H}[\tau_i, \tau_j]}{\sum_{k=1}^{n} \mathbf{H}[\tau_i, \tau_k] + \epsilon}, \tag{8.8}$$

where $\epsilon > 0$ is a small constant for numerical stability. This gives a sparse attention pattern: each fragment attends only to fragments with complementary type tags. A-type fragments attend to T-type fragments (with weight proportional to 2), and G-type fragments attend to C-type fragments (with weight proportional to 3). All other attention weights are zero.

Sparse Biological Attention

Type-based attention is *extremely sparse*: at most 25% of positions have non-zero attention weight (assuming uniform type distribution). This contrasts sharply with dense transformer attention, where every position attends to every other position. The sparsity is not a limitation but a feature—it encodes the structural constraint that only complementary bases interact, dramatically reducing the computational cost of attention while preserving biologically meaningful interactions.

8.2.3 SantaLucia Nearest-Neighbor Parameters

While the simple hydrogen bond count (2 for A–T, 3 for G–C) captures the qualitative picture, real DNA thermodynamics depend on the *stacking interactions* between adjacent base pairs. The **SantaLucia nearest-neighbor model** [SantaLucia, 1998] provides experimentally measured free energy parameters for all 16 possible nearest-neighbor dinucleotide pairs.

Definition 8.4 (Nearest-Neighbor Free Energy)**.** The free energy contribution of a nearest-neighbor pair is denoted $\Delta G^\circ(XY/X'Y')$, where XY represents two adjacent bases on the sense strand and

$X'Y'$ represents the complementary bases on the antisense strand (read $3' \to 5'$). The total free energy of duplex formation for a sequence of length n is:

$$\Delta G^\circ_{\text{total}} = \Delta G^\circ_{\text{init}} + \sum_{i=1}^{n-1} \Delta G^\circ(s_i s_{i+1} / \overline{s_{i+1}}\,\overline{s_i}), \tag{8.9}$$

where $\Delta G^\circ_{\text{init}}$ is the initiation free energy and the sum runs over all adjacent pairs in the sequence.

The 16 unique nearest-neighbor ΔG° values (in kcal/mol at 37°C, 1 M NaCl) are listed in Table 8.2.

Table 8.2: SantaLucia nearest-neighbor ΔG°_{37} values (kcal/mol) for all 16 dinucleotide pairs. More negative values indicate stronger (more stable) stacking. Data from SantaLucia [1998].

Pair ($5' \to 3'/3' \to 5'$)	ΔG°_{37}	**Pair ($5' \to 3'/3' \to 5'$)**	ΔG°_{37}
AA/TT	−1.00	GA/CT	−1.30
AT/TA	−0.88	GC/CG	−2.24
AC/TG	−1.44	GG/CC	−1.84
AG/TC	−1.28	GT/CA	−1.44
TA/AT	−0.58	CA/GT	−1.45
TT/AA	−1.00	CC/GG	−1.84
TC/AG	−1.30	CG/GC	−2.17
TG/AC	−1.45	CT/GA	−1.28

Using SantaLucia Parameters in HelixHash

HelixHash uses the SantaLucia nearest-neighbor parameters to compute a *thermodynamic bond weight* for each adjacent pair in a genome. Given fragments f_i and f_{i+1} with type tags τ_i and τ_{i+1}, the stacking weight is:

$$w_{\text{stack}}(i, i+1) = \frac{|\Delta G^\circ(\tau_i \tau_{i+1} / \overline{\tau_{i+1}}\,\overline{\tau_i})|}{\max_{\mathbf{t}} |\Delta G^\circ(\mathbf{t})|} \in [0, 1], \tag{8.10}$$

where the denominator normalizes by the strongest possible stacking interaction (GC/CG at −2.24 kcal/mol). This normalized weight enters the bond matrix as an additional term beyond Jaccard overlap and boundary alignment.

8.2.4 Worked Example: Computing ΔG° for ACGTACGT

Let us compute the total free energy of duplex formation for the sequence ACGTACGT step by step.

Example 8.2 (Free Energy Calculation)**. Given:** Sense strand = ACGTACGT; Antisense strand = TGCATGCA (read $3' \to 5'$).

Step 1: Identify all nearest-neighbor pairs. Reading left to right along the sense strand, the seven adjacent dinucleotide pairs are:

i	1	2	3	4	5	6
Pair	AC/TG	CG/GC	GT/CA	TA/AT	AC/TG	CG/GC

and the seventh pair: TT/AA for position 7 (GT at positions 7–8 is actually GT/CA).

Let us be precise. The pairs (sense $5' \to 3'$ / antisense $3' \to 5'$) are:

$$\begin{aligned}
&\text{Position 1–2:} \quad \mathsf{AC/TG} \quad \Delta G^\circ = -1.44 \\
&\text{Position 2–3:} \quad \mathsf{CG/GC} \quad \Delta G^\circ = -2.17 \\
&\text{Position 3–4:} \quad \mathsf{GT/CA} \quad \Delta G^\circ = -1.44 \\
&\text{Position 4–5:} \quad \mathsf{TA/AT} \quad \Delta G^\circ = -0.58 \\
&\text{Position 5–6:} \quad \mathsf{AC/TG} \quad \Delta G^\circ = -1.44 \\
&\text{Position 6–7:} \quad \mathsf{CG/GC} \quad \Delta G^\circ = -2.17 \\
&\text{Position 7–8:} \quad \mathsf{GT/CA} \quad \Delta G^\circ = -1.44
\end{aligned}$$

Step 2: Sum the contributions.

$$\sum_{i=1}^{7} \Delta G_i^\circ = (-1.44) + (-2.17) + (-1.44) + (-0.58) + (-1.44) + (-2.17) + (-1.44) = -10.68 \text{ kcal/mol}.$$

Step 3: Add initiation energy. For a sequence starting with A and ending with T, the initiation free energy is $\Delta G^\circ_{\text{init}} = +2.2$ kcal/mol (accounting for both initiation terms):

$$\Delta G^\circ_{\text{total}} = 2.2 + (-10.68) = -8.48 \text{ kcal/mol}.$$

Interpretation: The negative total free energy (-8.48 kcal/mol) indicates that duplex formation is thermodynamically favorable. In HelixHash, this sequence would receive a normalized bond energy of $|-8.48|/|\Delta G^\circ_{\max}| \approx 0.54$, indicating moderate structural stability.

8.2.5 The Full Bond Matrix

While the hydrogen bond matrix $\mathbf{H}$ captures type-level interactions, the full HelixHash bond matrix also incorporates content-level similarity:

Definition 8.5 (Bond Matrix)**.** Given a genome $G = (f_1, \ldots, f_n)$, the *bond matrix* $\mathbf{B} \in [0,1]^{n \times n}$ is defined by:

$$\mathbf{B}[i,j] = \alpha \cdot J(f_i, f_j) + (1 - \alpha) \cdot A(f_i, f_j), \tag{8.11}$$

where $J(f_i, f_j)$ is the Jaccard overlap between the motif sets of fragments f_i and f_j, $A(f_i, f_j)$ is the boundary alignment score between fragments f_i and f_j, and $\alpha = 0.75$ controls the balance between overlap and alignment.

The **Jaccard overlap** measures the fraction of shared k-motifs between two fragments:

$$J(f_i, f_j) = \frac{|\mathcal{M}_k(f_i) \cap \mathcal{M}_k(f_j)|}{|\mathcal{M}_k(f_i) \cup \mathcal{M}_k(f_j)|}. \tag{8.12}$$

The **boundary alignment** measures how well the ending motifs of fragment f_i match the starting motifs of fragment f_j, using cosine similarity over the last and first ℓ embeddings (default $\ell = 3$):

$$A(f_i, f_j) = \cos\big(f_i[-\ell{:}],\ f_j[{:}\ell]\big). \tag{8.13}$$

The 75%/25% weighting reflects the empirical finding that content overlap is more indicative of structural coherence than boundary smoothness alone, but that boundary alignment is essential for detecting well-assembled genomes versus randomly concatenated fragments.

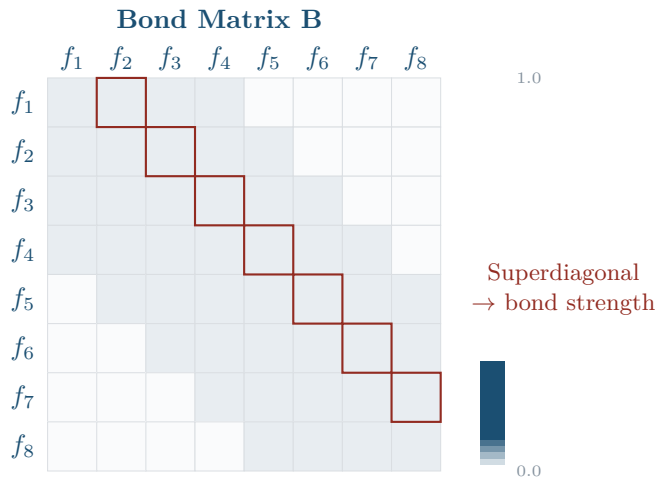


Figure 8.2: Visualization of the bond matrix $\mathbf{B}$ for an 8-fragment genome. Darker blue indicates stronger bonds. The matrix is naturally sparse: only nearby fragments share significant motif overlap. The superdiagonal (red outline) is averaged to compute bond strength $C_{\text{bond}}(G)$.

Sparse Bonds vs. Dense Attention

The bond matrix $\mathbf{B}$ superficially resembles a transformer's attention matrix, but differs in three critical respects:

1. **Sparsity:** Bond matrices are naturally sparse. Most fragment pairs share few motifs, so $\mathbf{B}[i, j] \approx 0$ for distant or unrelated fragments. In practice, over 85% of entries fall below a threshold of 0.1.

2. **Interpretation:** Each entry has a concrete structural meaning (shared motifs + boundary alignment), unlike attention weights which are learned correlations without guaranteed semantics.

3. **Persistence:** Bond matrices are computed once per genome and cached in the HelixHash archive, whereas attention matrices are recomputed at every forward pass.

8.2.6 Bond Strength

The overall *bond strength* of a genome quantifies its structural integrity as a single scalar:

$$C_{\text{bond}}(G) = \frac{1}{n-1} \sum_{i=1}^{n-1} \mathbf{B}[i, i+1], \tag{8.14}$$

i.e., the mean bond score along the main superdiagonal. A high bond strength indicates that consecutive fragments are coherent—they share motifs and have smooth boundaries. A low bond strength suggests fragmentation or poor assembly. Bond strength enters the selection objective as a regularization term (Section 8.6).

8.3 Motif Detection and Analysis

The basic unit of structural analysis in HelixHash is the *motif*—a short, contiguous subsequence of fragment embeddings extracted from a genome strand. Motifs are to HelixHash what k-mers are to bioinformatics and n-grams are to NLP.

8.3.1 Defining Genomic Motifs

Definition 8.6 (Genomic Motif)**.** A *k-motif* of a genome $G = (f_1, \ldots, f_n)$ is any contiguous subsequence of width k:

$$m_i^{(k)} = (f_i, f_{i+1}, \ldots, f_{i+k-1}), \quad 1 \leq i \leq n - k + 1. \tag{8.15}$$

The complete set of overlapping k-motifs from G is denoted $\mathcal{M}_k(G)$, with $|\mathcal{M}_k(G)| = n - k + 1$.

Why $k = 6$?

The default motif width in HelixHash is $k = 6$, chosen by analogy with biological codons. In molecular biology, codons are three-nucleotide units that specify amino acids. Since DOGMA uses four base types (A, T, C, G), a width-6 motif spans two "codons" worth of type information, yielding $4^6 = 4{,}096$ distinct type patterns. This provides sufficient granularity to capture structural variation while remaining computationally tractable. The choice also aligns with empirical findings in bioinformatics, where 6-mers are widely used for sequence classification tasks [Wood, 2014].

8.3.2 How HelixHash Detects Motifs

Motif detection in HelixHash proceeds in three phases:

Phase 1: Extraction. A sliding window of width k moves along both the forward and reverse strands, extracting all overlapping motifs. For a genome of length n, this yields $2(n - k + 1)$ motifs in total (counting both strands).

Phase 2: Type Quantization. Each motif is reduced to its *type sequence*—the ordered tuple of type tags $(\tau_i, \tau_{i+1}, \ldots, \tau_{i+k-1})$. This quantization maps the continuous fragment embeddings to a discrete alphabet, enabling efficient hashing and counting.

Phase 3: Frequency Counting. The frequency of each type pattern is tallied to produce the *motif frequency spectrum*.

8.3.3 Motif Frequency Spectrum

The *motif frequency spectrum* of a genome is the distribution of motif occurrences, quantized by their type patterns:

$$\phi_k(G)[\mathbf{t}] = \frac{|\{m_i^{(k)} \in \mathcal{M}_k(G) : \text{typeseq}(m_i^{(k)}) = \mathbf{t}\}|}{|\mathcal{M}_k(G)|}, \tag{8.16}$$

where $\mathbf{t} \in \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}^k$ is a type sequence and $\text{typeseq}(\cdot)$ extracts the type tags τ_i from each fragment in the motif. The motif frequency spectrum $\phi_k(G)$ is a probability distribution over the 4^k possible type patterns, serving as a structural fingerprint of the genome.

Example 8.3 (Computing a Motif Spectrum)**.** Consider a short genome with type sequence AATCGG and motif width $k = 3$. The four overlapping 3-motifs are:

Position	1	2	3
Motif	AAT	ATC	TCG

and the fourth motif at position 4: CGG. So the motif frequency spectrum has:

$$\phi_3(G)[\mathsf{AAT}] = \tfrac{1}{4}, \quad \phi_3(G)[\mathsf{ATC}] = \tfrac{1}{4}, \quad \phi_3(G)[\mathsf{TCG}] = \tfrac{1}{4}, \quad \phi_3(G)[\mathsf{CGG}] = \tfrac{1}{4}.$$

All other entries of $\phi_3(G)$ are zero. This uniform distribution indicates maximum diversity among the observed motifs—no single pattern dominates.

8.3.4 Hash Function Design for DNA-Like Sequences

Efficient motif detection requires a hash function that maps type sequences to integer indices quickly and with low collision rates. HelixHash uses a *radix-4 encoding* that treats the type sequence as a base-4 number:

$$h_{\text{radix}}(\mathbf{t}) = \sum_{j=0}^{k-1} \text{ord}(t_{j+1}) \cdot 4^{k-1-j}, \tag{8.17}$$

where $\text{ord}(\mathsf{A}) = 0$, $\text{ord}(\mathsf{C}) = 1$, $\text{ord}(\mathsf{G}) = 2$, $\text{ord}(\mathsf{T}) = 3$. This produces a perfect hash (no collisions) mapping 4^k type sequences to $\{0, 1, \ldots, 4^k - 1\}$. The hash can be updated incrementally as the sliding window advances: dropping the leftmost base and appending the rightmost base requires only a multiply, subtract, and add operation.

Comparison to NLP and Bioinformatics

The motif frequency spectrum occupies a middle ground between two well-studied tools:

- In **NLP**, character or word n-gram frequency profiles are used for language identification, authorship attribution, and text classification. HelixHash motifs are the genomic analogue, except that the "alphabet" is typed fragment embeddings rather than characters.
- In **bioinformatics**, k-mer frequency vectors are used for metagenomic binning, phylogenetic classification, and sequence comparison [Wood, 2014]. HelixHash inherits the computational efficiency of k-mer methods while operating over DOGMA's richer base representation.

The key difference is that HelixHash motifs carry both *type* information (from the τ_i tags) and *content* information (from the σ_i embeddings), enabling both structural and semantic analysis.

8.4 HelixHash as Locality-Preserving Encoding

A fundamental question in sequence representation is: *does the encoding preserve biological locality?* Two sequence regions that are biologically related (e.g., they regulate the same gene, or they fold into adjacent structural elements) should be "close" in the encoded representation. We now show that HelixHash's helical representation achieves this property more naturally than standard positional encodings.

8.4.1 Why Helical Representation Preserves Locality

Standard positional encodings (sinusoidal or learned) assign each position an embedding based solely on its linear index. This means that positions 1 and 100 are always "far apart," regardless of whether they participate in the same structural motif. The double-helix representation introduces a second axis of locality: the *complementary position* on the reverse strand.

Proposition 8.1 (Complementary Locality)**.** In the double-helix representation, the encoding of position i on the forward strand is structurally linked to the encoding of position $n + 1 - i$ on the reverse strand. If two positions i and j are both structurally linked to the same region of the reverse strand (i.e., $|n + 1 - i - j| \leq k$ for motif width k), then they share motif overlap and have correlated bond matrix entries.

This means that the helix encoding captures *three-dimensional* proximity (positions that interact across strands) in addition to linear proximity (adjacent positions on the same strand).

8.4.2 Comparison with Standard Positional Encodings

Table 8.3: Comparison of HelixHash helical encoding with standard positional encoding approaches.

Property	**Standard Positional**	**HelixHash Helical**
Locality	Linear only (position distance)	Linear + complementary (cross-strand)
Structural info	None (position index only)	Type tags, bond strengths, motif overlap
Bidirectionality	Requires separate forward/backward passes	Built-in via sense/antisense strands
Sparsity	Dense (all positions interact)	Sparse (only bonded positions interact)
Biological grounding	None	Watson–Crick base pairing
Scalability	$O(T^2)$ in attention	$O(T \cdot k)$ via motif windowing

8.4.3 Locality-Sensitive Hashing on Helical Structure

The helical structure enables a specialized form of locality-sensitive hashing (LSH) that respects both linear and complementary proximity.

Definition 8.7 (Helical LSH)**.** A *helical LSH* hash function $h_{\text{helix}} : \mathcal{G} \to \{0, 1\}^B$ maps a genome to a B-bit signature such that:

1. Genomes with similar forward-strand motif spectra have high probability of sharing hash bits.
2. Genomes with similar reverse-strand motif spectra *also* have high probability of sharing hash bits.
3. The hash is *strand-symmetric*: $h_{\text{helix}}(G) = h_{\text{helix}}(G')$ whenever G and G' have the same canonical motif spectrum (where each motif is replaced by its canonical form from Eq. 8.5).

Property (3) is what distinguishes helical LSH from standard LSH: it ensures that the hash respects the biological symmetry of complementary strands. Two genomes that differ only by a swap of sense and antisense strands receive the same hash.

8.5 Strand Sketches

Storing and comparing full motif frequency spectra is expensive for large genomes ($4^6 = 4{,}096$ dimensions even for the default width). HelixHash addresses this through *strand sketches*—compact, fixed-size summaries that support fast similarity estimation.

Definition 8.8 (Signed Count Sketch)**.** A *signed count sketch* of width w for a genome G is a vector $\mathbf{s} \in \mathbb{Z}^w$ constructed as follows. Let $h : \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}^k \to \{1, \ldots, w\}$ be a hash function and $g : \{\mathsf{A}, \mathsf{T}, \mathsf{C}, \mathsf{G}\}^k \to \{-1, +1\}$ be a sign function. For each motif $m_i^{(k)} \in \mathcal{M}_k(G)$ with type sequence $\mathbf{t}_i$:

$$\mathbf{s}[h(\mathbf{t}_i)] \;\mathrel{+}=\; g(\mathbf{t}_i). \tag{8.18}$$

The sketch $\mathbf{s}$ has three desirable properties. First, it is *linear*: the sketch of a union is the sum of individual sketches. Second, inner products of sketches approximate inner products of full frequency vectors, with error decreasing as $O(1/\sqrt{w})$. Third, construction is $O(n)$ in genome length, requiring a single pass over the fragment sequence.

8.5.1 MinHash Lineage and Locality-Sensitive Hashing

For clustering and nearest-neighbor search over large genome populations, HelixHash employs *locality-sensitive hashing* (LSH) derived from the MinHash family.

Definition 8.9 (HelixHash LSH Signature)**.** Given a genome G, its *LSH signature* is a vector of B hash values:

$$\mathrm{LSH}(G) = (h_1^{\min}(G), h_2^{\min}(G), \ldots, h_B^{\min}(G)) \in \{0, 1, \ldots, 2^b - 1\}^B, \tag{8.19}$$

where each $h_j^{\min}(G) = \min_{\mathbf{t} \in \mathrm{typeseq}(\mathcal{M}_k(G))} h_j(\mathbf{t})$ is a MinHash value computed from a distinct hash function h_j, and b is the bit width per hash. The default configuration uses $B = 12$ bands with $b = 12$ bits, yielding a 144-bit signature.

LSH Clustering Protocol

Two genomes G_1 and G_2 are assigned to the same *LSH cluster* if their signatures agree on at least one complete band of r consecutive hash values:

$$\mathrm{SameCluster}(G_1, G_2) \iff \exists\, j : \mathrm{LSH}(G_1)[j \cdot r : (j+1) \cdot r] = \mathrm{LSH}(G_2)[j \cdot r : (j+1) \cdot r]. \tag{8.20}$$

With $B = 12$ bands and $r = 1$ row per band, the probability that two genomes with Jaccard similarity J are placed in the same cluster is $1 - (1 - J)^{12}$, providing high recall for pairs with $J > 0.3$.

The 12-bit LSH clustering is the primary mechanism by which HelixHash maintains structural diversity during evolutionary training (Section 8.6).

8.6 Novelty Detection

Evolutionary search risks premature convergence: the population collapses to a narrow region of genome space, losing the diversity needed to discover novel solutions. HelixHash provides three complementary novelty mechanisms that counteract this tendency.

8.6.1 Archive-Based Novelty

The HelixHash archive $\mathcal{A}_t$ stores the strand sketches of all genomes encountered during evolutionary training. The *archive novelty* of a candidate genome G is its average distance to the K nearest archived sketches:

$$N_{\text{archive}}(G) = \frac{1}{K} \sum_{i=1}^{K} \|\mathbf{s}(G) - \mathbf{s}(G_i^{\text{nn}})\|_2, \tag{8.21}$$

where $G_1^{\text{nn}}, \ldots, G_K^{\text{nn}}$ are the K nearest neighbors of G in the archive under sketch distance, and $\mathbf{s}(\cdot)$ denotes the signed count sketch. High archive novelty indicates that the genome is structurally distinct from previously seen genomes.

8.6.2 Detecting Novel vs. Seen Patterns

A key application of HelixHash's novelty machinery is distinguishing genuinely novel genomic configurations from previously encountered patterns. This capability is critical during evolutionary training: mutations that produce structurally novel genomes should be encouraged (to maintain diversity), while mutations that merely recreate previously explored configurations should be deprioritized.

Definition 8.10 (Novelty Threshold)**.** A genome G is classified as *novel* at generation t if:

$$N_{\text{archive}}(G) > \mu_{\mathcal{A}} + \kappa \cdot \sigma_{\mathcal{A}}, \tag{8.22}$$

where $\mu_{\mathcal{A}}$ and $\sigma_{\mathcal{A}}$ are the mean and standard deviation of archive novelty scores across the current population, and $\kappa > 0$ is a sensitivity parameter (default $\kappa = 1.5$).

The threshold adapts over time: as the archive grows and the population explores more of genome space, $\mu_{\mathcal{A}}$ decreases (new genomes tend to be closer to something in the archive), and the threshold tightens accordingly. This progressive tightening is desirable—early in training, the bar for "novelty" is low (encouraging broad exploration), while later in training the bar rises (requiring genuinely innovative configurations).

Example 8.4 (Novel vs. Seen Classification)**.** Suppose the archive contains 500 genomes and the current population has 50 candidates. The archive novelty scores have mean $\mu_{\mathcal{A}} = 4.2$ and standard deviation $\sigma_{\mathcal{A}} = 1.8$. With $\kappa = 1.5$, the novelty threshold is $4.2 + 1.5 \times 1.8 = 6.9$.

- Genome G_a with $N_{\text{archive}} = 8.1$: **Novel**. This genome's structural signature is far from anything in the archive. It likely arose from a rare mutation combination.
- Genome G_b with $N_{\text{archive}} = 3.5$: **Seen**. This genome is structurally similar to archived genomes. It may be a minor variant of a previously explored configuration.

8.6.3 Cluster-Aware Selection

Using the LSH clusters from Section 8.5, HelixHash computes a *cluster novelty* score that penalizes genomes falling into already-crowded clusters:

$$N_{\text{cluster}}(G) = \frac{1}{|\mathcal{C}(G)|}, \tag{8.23}$$

where $\mathcal{C}(G)$ is the LSH cluster containing G and $|\mathcal{C}(G)|$ is its population count. Genomes in sparsely populated clusters receive higher novelty scores, encouraging exploration of underrepresented structural regions.

8.6.4 Temporal Novelty Decay

To prevent the archive from growing without bound and to prioritize *recent* novelty, HelixHash applies exponential temporal decay:

$$N_{\text{temporal}}(G, t) = \sum_{s=0}^{t-1} \gamma^{t-s} \cdot \mathbf{1}[\text{SameCluster}(G, G_s)], \tag{8.24}$$

where $\gamma = 0.9$ is the decay factor, t is the current generation, and G_s is the genome selected at generation s. The indicator $\mathbf{1}[\cdot]$ counts cluster co-occurrences with exponentially decreasing weight. Low temporal novelty (few recent cluster mates) is rewarded.

8.6.5 The Selection Objective

The complete HelixHash-informed selection objective combines task fitness with structural diversity pressures:

$$\boxed{\mathcal{L}(G) = J_{\text{task}} - \eta_1 N_{\text{archive}} - \eta_2 N_{\text{cluster}} - \eta_3 N_{\text{temporal}} - \eta_4 C_{\text{bond}} - \eta_5 C_{\text{strand}}} \tag{8.25}$$

where:

- J_{task} is the primary task fitness (e.g., accuracy, perplexity).
- N_{archive}, N_{cluster}, N_{temporal} are the three novelty terms (negated because higher novelty should *increase* the objective).
- C_{bond} is the bond strength (Eq. 8.14), regularizing structural coherence.
- C_{strand} is the strand alignment consistency between $F(G)$ and $R(G)$.
- $\eta_1, \ldots, \eta_5 > 0$ are weighting coefficients tuned per task.

Sign Convention

Note the negative signs on the novelty terms. Because $\mathcal{L}(G)$ is *maximized* during selection, and high novelty scores ($N_{\text{archive}}, N_{\text{cluster}}$) indicate *desirable* structural uniqueness, they are subtracted (i.e., added in effect) to increase the objective. The bond and strand coherence terms $C_{\text{bond}}, C_{\text{strand}}$ are similarly subtracted because higher coherence is desirable. The convention follows the evo-trainer codebase, where the objective is framed as a loss to be minimized in the outer loop, with the sign flip handled at the selection stage.

8.7 HelixHash as Embedding

Beyond its role in evolutionary selection, HelixHash provides a *motif-based embedding layer* that can serve as the input representation for foundation models. This section describes the dual-stream architecture that integrates HelixHash embeddings with standard token embeddings.

8.7.1 Motif Embedding Layer

Each k-motif is mapped to a dense vector through a learned embedding:

$$\mathbf{e}_i^{\text{motif}} = W_{\text{motif}} \cdot \text{pool}(m_i^{(k)}) + \mathbf{p}_i, \tag{8.26}$$

where $W_{\text{motif}} \in \mathbb{R}^{d \times (k \cdot d_f)}$ is the motif projection matrix, $\text{pool}(\cdot)$ concatenates the k fragment embeddings in the motif, and $\mathbf{p}_i$ is a positional encoding. The motif embedding captures local structural patterns in a form compatible with downstream transformer or DOGMA processing blocks.

8.7.2 Dual-Stream Architecture

The HelixHash embedding layer feeds into a *dual-stream* architecture that processes standard and helix channels in parallel:

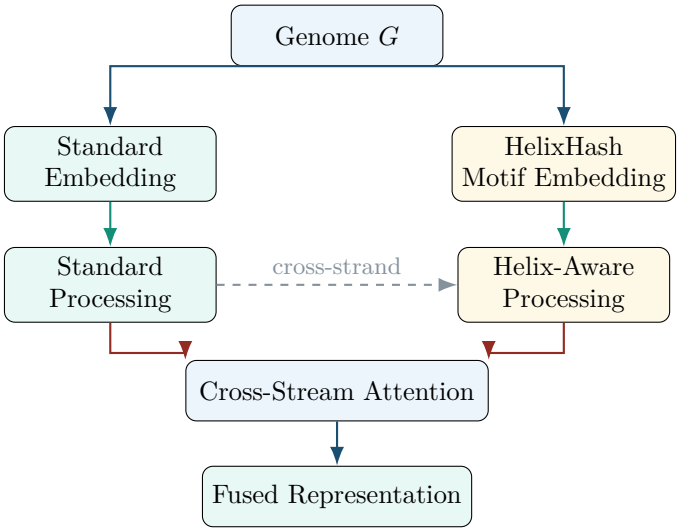


Figure 8.3: Dual-stream HelixHash embedding architecture. The standard channel processes token-level embeddings; the helix channel processes motif-level structural features. Cross-stream attention fuses both representations.

The standard stream processes fragment embeddings through conventional layers (attention or state-space blocks). The helix stream processes motif embeddings that encode local structural patterns from both the forward and reverse strands. A cross-stream attention mechanism allows information to flow between the two channels:

$$\mathbf{h}_{\text{fused}} = \text{CrossAttn}(\mathbf{h}_{\text{std}}, \mathbf{h}_{\text{helix}}) + \mathbf{h}_{\text{std}}, \tag{8.27}$$

where $\mathbf{h}_{\text{std}}$ and $\mathbf{h}_{\text{helix}}$ are the hidden states of the standard and helix streams, respectively. The residual connection ensures that the helix stream *augments* rather than replaces standard processing.

8.8 The HelixHash Algorithm

We now present the complete HelixHash computation as a unified algorithm. Given a genome, Algorithm 2 produces all structural signatures needed for the selection objective (Eq. 8.25).

Algorithm 2 HelixHash: Structural Signature Computation

Require: Genome $G = (f_1, \ldots, f_n)$, motif width k, sketch width w, LSH bands B
Ensure: Strand sketch $\mathbf{s}$, LSH signature ℓ, bond strength C_{bond}, strand consistency C_{strand}

// Step 1: Construct strands
$F \leftarrow [f_1, f_2, \ldots, f_n]$ ▷ Forward strand
$R \leftarrow [\text{rev}(f_n), \text{rev}(f_{n-1}), \ldots, \text{rev}(f_1)]$ ▷ Reverse strand
// Step 2: Extract motifs from both strands
$\mathcal{M}_F \leftarrow \{(f_i, \ldots, f_{i+k-1}) : 1 \le i \le n-k+1\}$
$\mathcal{M}_R \leftarrow \{(r_i, \ldots, r_{i+k-1}) : 1 \le i \le n-k+1\}$ where $r_j = \text{rev}(f_{n+1-j})$
$\mathcal{M} \leftarrow \mathcal{M}_F \cup \mathcal{M}_R$ ▷ Union of both strands
// Step 3: Build signed count sketch
Initialize $\mathbf{s} \leftarrow \mathbf{0} \in \mathbb{Z}^w$
for each motif $m \in \mathcal{M}$ **do**
    $\mathbf{t} \leftarrow \text{typeseq}(m)$
    $\mathbf{s}[h(\mathbf{t})] \leftarrow \mathbf{s}[h(\mathbf{t})] + g(\mathbf{t})$
end for
// Step 4: Compute LSH signature
for $j = 1$ to B **do**
    $\ell[j] \leftarrow \min_{\mathbf{t} \in \text{typeseq}(\mathcal{M})} h_j(\mathbf{t})$
end for
// Step 5: Compute bond matrix superdiagonal
for $i = 1$ to $n-1$ **do**
    $C_{\text{bond}} \mathrel{+}= 0.75 \cdot J(f_i, f_{i+1}) + 0.25 \cdot A(f_i, f_{i+1})$
end for
$C_{\text{bond}} \leftarrow C_{\text{bond}}/(n-1)$
// Step 6: Strand consistency
$C_{\text{strand}} \leftarrow \frac{1}{2}[\cos(\bar{F}, \bar{R}) + 1]$ where $\bar{F}, \bar{R}$ are mean-pooled strand representations
return $\mathbf{s}, \ell, C_{\text{bond}}, C_{\text{strand}}$

Complexity of HelixHash

Algorithm 2 runs in $O(n \cdot k)$ time and $O(w + B)$ auxiliary space, where n is the genome length, k is the motif width, w is the sketch width, and B is the number of LSH bands. Since k, w, and B are constants (typically $k = 6$, $w = 256$, $B = 12$), the algorithm is effectively $O(n)$ in genome length. This linear-time construction is critical for scalability: during evolutionary training, HelixHash must be recomputed for every candidate genome at every generation.

8.9 Theoretical Properties

We establish two key theoretical results about HelixHash representations.

Theorem 8.2 (Sketch Approximation Guarantee)**.** Let $\phi_k(G_1)$ and $\phi_k(G_2)$ be the motif frequency spectra of two genomes, and let $\mathbf{s}_1, \mathbf{s}_2$ be their signed count sketches of width w. Then:

$$\mathbb{E}\left[\frac{\langle \mathbf{s}_1, \mathbf{s}_2 \rangle}{|\mathcal{M}_k(G_1)| \cdot |\mathcal{M}_k(G_2)|}\right] = \langle \phi_k(G_1), \phi_k(G_2) \rangle, \tag{8.28}$$

with variance bounded by $O(1/w)$.

Proof sketch. This follows from the standard analysis of count sketches [Charikar, 2002]. The hash function h distributes motif types uniformly across w bins, and the sign function g ensures that cross-terms cancel in expectation. The variance bound follows from pairwise independence of the hash family. The full proof follows from standard techniques in the sketching literature. □

Theorem 8.3 (Bond Strength and Assembly Quality)**.** Let $G^* = (f_1, \ldots, f_n)$ be a "well-assembled" genome (one where consecutive fragments were originally adjacent) and let G^π be a random permutation of the same fragments. Then in expectation:

$$\mathbb{E}[C_{\text{bond}}(G^*)] > \mathbb{E}[C_{\text{bond}}(G^\pi)], \tag{8.29}$$

with the gap increasing with the average motif sharing between genuinely adjacent fragments.

This result confirms that bond strength is a meaningful quality measure: well-organized genomes have higher bond strength than randomly shuffled ones, providing a useful signal for the evolutionary selection objective.

8.10 Connection to the DOGMA Pipeline

HelixHash connects to every major component of the DOGMA architecture:

- **Genome (Ch. 7):** The double-helix representation defines the structural format of the genome. Every genome maintained by the Evolutor has an associated HelixHash signature in the archive $\mathcal{A}_t$.
- **Tera (Ch. 9):** The Tera regime state uses HelixHash novelty scores to modulate mutation pressure. When novelty drops below a threshold, Tera increases mutation rates to restore diversity.
- **Evolutionary Training (Ch. 12):** The selection objective (Eq. 8.25) is the primary interface between HelixHash and the evolutionary loop. HelixHash signatures determine which genomes survive to the next generation.
- **Foundation Models:** The dual-stream embedding (Section 8.7) provides helix-aware input representations for foundation model pretraining, discussed further in Chapter 1.

8.11 Exercises

8.1. [**Fundamentals**] Compute the motif frequency spectrum $\phi_3(G)$ for a genome with type sequence AATCGGATCG. How many distinct 3-motif type patterns appear? What is the most frequent pattern?

8.2. [**Bond Matrix**] Given the hydrogen bond matrix $\mathbf{H}$ from Table 8.1, compute the total hydrogen bond count for each of the following duplexes: (a) AAAA paired with TTTT; (b) GGGG paired with CCCC; (c) ACGT paired with TGCA. Which duplex is most thermodynamically stable and why?

8.3. [**Nearest-Neighbor Model**] Using the SantaLucia parameters from Table 8.2, compute $\Delta G^{\circ}_{\text{total}}$ for the sequence GCGCGCGC. Compare your answer with the result from Example 8.2 for ACGTACGT. Which sequence forms a more stable duplex?

8.4. [**Analysis**] A genome of length $n = 1{,}000$ has its HelixHash computed with $k = 6$, $w = 256$, and $B = 12$. Calculate: (a) the number of overlapping motifs extracted from each strand; (b) the total sketch construction time in terms of hash evaluations; (c) the memory footprint of the LSH signature in bits.

8.5. [**Design**] The bond matrix weighting uses $\alpha = 0.75$ for Jaccard overlap and $1 - \alpha = 0.25$ for boundary alignment. Design an experiment to determine whether this ratio is optimal. What metric would you use to evaluate different values of α? Suggest at least three candidate values and predict their effects.

8.6. [**Canonical Forms**] Enumerate all canonical 2-motif type sequences (i.e., one representative from each $\{\mathbf{t}, \mathrm{RC}(\mathbf{t})\}$ pair). How many are there? Show that this is always exactly half of 4^k for even k (hint: there are no palindromic reverse complements for even $k > 0$).

8.7. [**Coding**] Implement the signed count sketch (Definition 8.8) in Python. Use `mmh3` (MurmurHash3) for h and a separate MurmurHash with different seed for g (mapping to $\{-1, +1\}$). Generate two random genomes of length 500 with 70% shared fragments and verify that the sketch inner product approximates the true motif frequency inner product.

8.8. [**Locality**] Consider two genomes G_1 and G_2 that differ only by a single point mutation at position i. How does the motif frequency spectrum $\phi_k(G_1)$ differ from $\phi_k(G_2)$? At most how many motif entries can change? Use this to argue that HelixHash is locality-preserving.

8.9. [**Theory**] Prove that the strand alignment score $S_{\text{strand}}(G_1, G_2)$ (Eq. 8.6) satisfies the property $S_{\text{strand}}(G, G) = 1$ for any genome G whose complementation operator is exactly involutory ($\text{rev}(\text{rev}(f)) = f$ for all fragments f).

8.10. [**Research**] The temporal novelty decay (Eq. 8.24) uses $\gamma = 0.9$. Investigate how different values of γ affect evolutionary dynamics. At $\gamma = 0$, only the current generation matters; at $\gamma = 1$, all history is weighted equally. Derive the effective memory horizon $\tau_{\text{eff}} = 1/(1 - \gamma)$ and discuss its implications for a 500-generation evolutionary run.

8.12 Key Takeaways

Key Takeaways

- HelixHash represents every computational genome as a double helix with forward strand $F(G)$ (sense, $5' \to 3'$) and reverse strand $R(G)$ (antisense, $3' \to 5'$), enabling bidirectional structural analysis.
- Watson–Crick complementarity ($\overline{\mathsf{A}} = \mathsf{T}$, $\overline{\mathsf{G}} = \mathsf{C}$) acts as an involutory hash function; canonical motif forms exploit this symmetry to halve the effective vocabulary.
- The 4×4 hydrogen bond matrix $\mathbf{H}$ (A–T: 2 bonds, G–C: 3 bonds) provides biologically grounded, sparse attention weights. SantaLucia nearest-neighbor parameters refine this to 16 stacking free-energy values.
- Width-k overlapping motifs (default $k = 6$) are the basic unit of structural analysis, analogous to k-mers in bioinformatics and n-grams in NLP.
- Signed count sketches provide $O(n)$ construction of compact similarity-preserving summaries; 12-band LSH signatures enable fast clustering of genome populations.
- Bond matrices capture inter-fragment coherence through 75% Jaccard overlap + 25% boundary alignment, yielding sparse structural maps that contrast with dense attention matrices.
- Three novelty mechanisms—archive, cluster, and temporal ($\gamma = 0.9$ decay)—prevent evolutionary collapse and maintain structural diversity. Novelty detection classifies genomes as novel or seen using an adaptive threshold.
- The helical encoding is designed to preserve both linear proximity and complementary (cross-strand) proximity. Whether that inductive bias improves on standard positional encodings is an open empirical question that requires matched-compute ablations.
- The dual-stream embedding architecture integrates HelixHash motif features with standard token embeddings for foundation model training.

Suggested Reading

- Charikar [2002]: Similarity estimation via count sketches, the theoretical foundation for HelixHash sketches.
- Broder [1997]: MinHash and locality-sensitive hashing, the basis for HelixHash's LSH clustering.
- Wood [2014]: Kraken and k-mer methods in metagenomics, a bioinformatics parallel to motif extraction.
- SantaLucia [1998]: Unified nearest-neighbor thermodynamic parameters, the source of the 16 ΔG° values used in HelixHash's stacking weights.
- Lehman and Stanley [2011]: Novelty search in evolutionary algorithms, the inspiration for archive-based novelty scoring.

- Chapter 7 of this book: the six-stage pipeline that HelixHash supports.
- Chapter 12 of this book: the evolutionary training loop where HelixHash's selection objective is applied.

Chapter 9

Tera — The Hidden Regime State

An organism does not optimize one thing. It optimizes its continued existence across a landscape of competing pressures—and it remembers which pressures mattered most.

— DOGMA Design Notes, 2025

In Chapter 7, we introduced the six-stage DOGMA pipeline and noted that the Evolutor maintains a persistent *Tera regime state* that guides regulation and synthesis. In Chapter 8, we saw how HelixHash encodes structural novelty. This chapter reveals the hidden engine beneath both: **Tera**, DOGMA's multi-objective evolutionary pressure system.

Tera is to DOGMA what the epigenome is to the genome—a layer of dynamic, heritable state that modulates how the underlying structure is read, modified, and expressed. Just as epigenetic marks do not change DNA sequence but profoundly alter gene expression [Allis and Jenuwein, 2016], the Tera regime state does not change the genomic bases themselves but determines *which mutation strategies are applied*, *how aggressively exploration proceeds*, and *what trade-offs the system prioritizes* at any given moment.

This chapter is organized as follows. We begin with an intuitive explanation of what a regime state is and why it matters (Section 9.1). We then formalize the multi-objective pressure system (Section 9.2), define the Tera regime state mathematically (Section 9.3), explore mutation families and how they are steered (Section 9.4), present the Tera state machine (Section 9.5), discuss meta-policy learning (Section 9.6), connect to biological evolution (Section 9.7), and walk through a detailed worked example (Section 9.9). We conclude with the full Tera Double-Helix Complex (Section 9.11) and exercises.

9.1 What Is a Regime State?

Before diving into formalism, let us build an intuitive understanding of what a "regime state" means.

9.1.1 Everyday Analogies

Example 9.1 (Traffic Flow Regimes)**.** Consider traffic on a highway. At low density, cars move freely at high speed—this is the *free-flow regime*. As density increases, drivers slow down and form platoons—the *synchronized flow regime*. Beyond a critical density, traffic jams form and propagate backward—the *congested regime*. Each regime has its own dynamics: the rules governing how

individual cars behave change depending on which regime the system is in. A driver who accelerates aggressively in free flow may need to brake constantly in congestion. The regime is a hidden variable that determines which behavioral strategy is optimal.

Example 9.2 (Weather Systems)**.** A weather system operates in distinct regimes: clear skies, scattered clouds, overcast, light rain, thunderstorms. Each regime has its own physics. In the clear-sky regime, solar heating dominates; in the thunderstorm regime, convective dynamics take over. Meteorologists do not model weather with a single equation—they identify the current regime and apply the appropriate physical model. The regime is not directly observed; it is *inferred* from measurements of temperature, pressure, humidity, and wind speed.

Example 9.3 (A Student's Study Strategy)**.** Think about how you study for exams. Early in the semester, you might focus on understanding concepts (exploration). As exams approach, you shift to practice problems (precision). If you realize you are weak in one topic, you focus there (targeted repair). Your "study regime" changes based on feedback (grades, practice test scores) and time pressure. You do not use a single fixed strategy all semester—you adapt.

The Core Idea of a Regime State

A **regime state** is a compact summary of which behavioral strategy a system should currently follow. It changes over time in response to feedback. It is "hidden" because it is not directly observed but inferred from noisy signals. In DOGMA, the regime state determines which evolutionary mutations are applied to the genome—it is the system's adaptive strategy selector.

9.1.2 Why Does DOGMA Need a Regime State?

Consider what happens without one. Suppose DOGMA always applied the same mutation strategy—say, always trying to improve output precision. The system would become excellent at precise answers but might produce verbose, incoherent outputs that ignore context. Conversely, if it always explored novel configurations, it would be creative but unreliable.

The regime state solves this by continuously monitoring multiple dimensions of performance and directing evolutionary effort where it is most needed. It is a *dynamic priority system* that prevents the evolutionary search from getting stuck in any single optimization direction.

9.2 The Problem of Multi-Objective Evolution

9.2.1 Why Single-Objective Optimization Fails

Most machine learning systems optimize a single scalar loss $\mathcal{L}(\theta)$. Even when practitioners combine multiple terms—accuracy, regularization, auxiliary losses—they reduce the result to a weighted sum:

$$\mathcal{L}_{\text{total}}(\theta) = \sum_{i=1}^{k} \lambda_i \mathcal{L}_i(\theta). \tag{9.1}$$

This scalarization is convenient but fundamentally limiting. The weights λ_i are fixed hyperparameters chosen before training begins, embedding a *static* preference over objectives. If the system encounters a regime where coherence matters more than efficiency, or where exploration should temporarily dominate exploitation, the fixed weights cannot adapt.

To see why this is a problem, consider a concrete scenario. Suppose you train a language model with a combined loss: 70% accuracy + 20% coherence + 10% efficiency. Early in training, the

model produces incoherent text—coherence should receive more weight. Later, the model becomes coherent but verbose—efficiency should increase. With fixed weights, you cannot adapt to these shifting needs without restarting training.

Biological Multi-Objective Optimization

A bacterium simultaneously optimizes nutrient uptake, waste removal, membrane integrity, DNA repair, motility, and stress response. It does not reduce these to a single scalar. Instead, it maintains a *regulatory network* that dynamically rebalances priorities based on environmental signals. When nutrients are scarce, metabolic pathways are upregulated; when DNA damage is detected, repair mechanisms take precedence. The "weights" are not fixed—they are functions of the organism's internal state and external context.

9.2.2 The Three Core Pressures: Quality, Diversity, and Coherence

Before introducing all six Tera dimensions, let us focus on three fundamental pressures that illustrate how multi-objective balancing works.

1. **Quality pressure** pushes the system to produce correct, precise outputs. Left unchecked, quality pressure leads to overfitting: the system memorizes solutions to known problems but cannot generalize.

2. **Diversity pressure** pushes the system to explore new solution strategies. Left unchecked, diversity pressure leads to chaos: the system tries random configurations without converging on anything useful.

3. **Coherence pressure** pushes the system to maintain internal consistency. Left unchecked, coherence pressure leads to stagnation: the system refuses to change because any modification risks breaking existing structure.

These three pressures exist in fundamental tension: quality wants precision, diversity wants exploration, and coherence wants stability. The Tera system must balance all three simultaneously.

Example 9.4 (Balancing Three Pressures: A Restaurant)**.** Imagine you are running a restaurant. Quality pressure is customer satisfaction with current dishes. Diversity pressure is the need to innovate the menu. Coherence pressure is the need to keep the kitchen running smoothly (staff know how to make the dishes, ingredients are available). If you only optimize quality, you never change the menu. If you only diversify, staff cannot keep up with constant changes. If you only maintain coherence, you serve the same dishes forever regardless of customer feedback.

9.2.3 The Pareto Landscape

In multi-objective optimization, solutions that cannot be improved on one objective without degrading another lie on the *Pareto front*. For k objectives, this front is a $(k-1)$-dimensional surface in objective space.

Definition 9.1 (Pareto Dominance)**.** A solution $\mathbf{x}$ *dominates* solution $\mathbf{y}$ (written $\mathbf{x} \succ \mathbf{y}$) if and only if $f_i(\mathbf{x}) \leq f_i(\mathbf{y})$ for all objectives $i \in \{1, \ldots, k\}$ and $f_j(\mathbf{x}) < f_j(\mathbf{y})$ for at least one objective j.

The challenge is not merely finding points on the Pareto front but *navigating* it—moving to the region of the front that best matches current needs. Tera's contribution is to maintain a *dynamic pressure vector* that steers the evolutionary search toward different regions of the Pareto front as conditions change—a biological thermostat for multi-objective evolution.

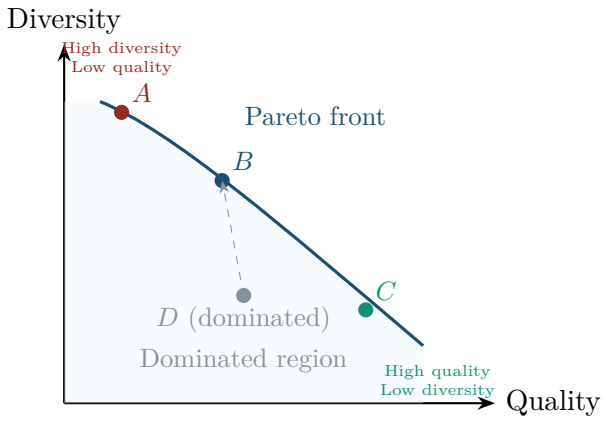


Figure 9.1: The Pareto front for two objectives (quality and diversity). Points A, B, and C lie on the front: none dominates the others. Point D is dominated by B (worse on both objectives). The Tera system navigates along this front, moving toward A when diversity is needed and toward C when quality is paramount.

9.3 The Tera Regime State

9.3.1 The Six Pressure Dimensions

Tera Regime State

The **Tera regime state** at time t is a six-dimensional pressure vector:

$$\mathbf{R}_t = (p_t^{(\text{gnd})}, p_t^{(\text{ext})}, p_t^{(\text{sch})}, p_t^{(\text{eff})}, p_t^{(\text{exp})}, p_t^{(\text{coh})}) \in [0,1]^6, \tag{9.2}$$

where each component quantifies the evolutionary pressure along one dimension of system performance.

Each pressure dimension captures a specific aspect of system behavior:

1. **Grounding pressure** $p^{(\text{gnd})}$: Measures contextual fidelity—how well the system's outputs are anchored to provided context and factual grounding. High grounding pressure drives mutations that strengthen retrieval pathways and context integration. *Think of this as:* "Is the system paying attention to the information it was given?"

2. **Exactness pressure** $p^{(\text{ext})}$: Measures answer precision—the degree to which outputs match expected formats, values, and content. This pressure increases when evaluations reveal imprecise or partially correct responses. *Think of this as:* "Is the answer correct and precise?"

3. **Schema pressure** $p^{(\text{sch})}$: Measures structural compliance—adherence to required output schemas (JSON, XML, structured formats). When outputs fail schema validation, this pressure rises. *Think of this as:* "Does the output have the right structure?"

4. **Efficiency pressure** $p^{(\text{eff})}$: Measures resource economy—the ratio of output quality to computational cost. This pressure penalizes verbose, redundant, or computationally wasteful solutions. *Think of this as:* "Is the system being concise and efficient?"

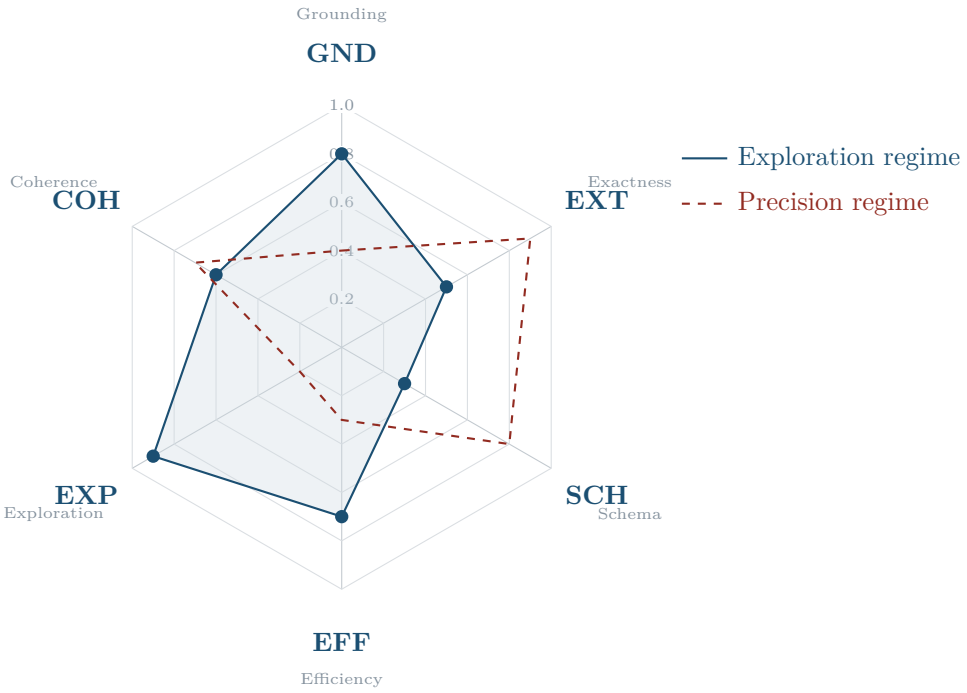


Figure 9.2: The six-dimensional Tera pressure space visualized as a radar chart. The solid blue polygon shows an exploration-dominant regime ($p^{(\text{exp})} = 0.9$); the dashed red polygon shows a precision-dominant regime ($p^{(\text{ext})} = 0.9$, $p^{(\text{sch})} = 0.8$). The Tera system dynamically transitions between such regimes based on evaluation history.

5. **Exploration pressure** $p^{(\text{exp})}$: Measures novelty seeking—the system's tendency to explore new regions of solution space rather than exploit known strategies. This pressure increases when fitness plateaus. *Think of this as:* "Is the system stuck in a rut and needs to try something new?"

6. **Coherence pressure** $p^{(\text{coh})}$: Measures structural stability—the internal consistency of the genome and the logical coherence of outputs. This pressure counterbalances exploration, preventing the system from destabilizing productive configurations. *Think of this as:* "Is the system still internally consistent?"

9.3.2 Visualizing the Pressure Space

The six-dimensional pressure vector can be visualized as a radar chart (also called a spider plot), where each axis represents one pressure dimension. Different regime states produce different polygon shapes.

9.3.3 How Pressures Interact and Balance

The six pressures do not operate in isolation. They form natural alliances and tensions:

- **Precision alliance:** Grounding, exactness, and schema pressures often rise together. When the system produces incorrect outputs, all three dimensions suffer.

- **Exploration–coherence tension:** Exploration and coherence pressures are natural antagonists. Exploration introduces change; coherence resists it. The balance between them determines how "bold" the system's mutations are.
- **Efficiency–quality trade-off:** Making outputs more concise (efficiency) can sacrifice detail (exactness). The system must find the sweet spot.

Pressure Correlation Monitoring

In the EVO-TRAINER implementation, the system monitors the correlation matrix of the six pressure time series. Strong positive correlations (e.g., between grounding and exactness) indicate that these dimensions are coupled and may benefit from joint mutations. Strong negative correlations (e.g., exploration vs. coherence) indicate active tension that requires careful balancing.

9.3.4 Exponential Memory Decay

The regime state is not computed from a single evaluation but from a *decaying window* of evaluation history. Each pressure component is updated using an exponential moving average (EMA):

$$p_t^{(d)} = (1 - \alpha)\, p_{t-1}^{(d)} + \alpha\, s_t^{(d)}, \tag{9.3}$$

where $s_t^{(d)} \in [0, 1]$ is the raw signal from the most recent evaluation along dimension d, and $\alpha \in (0, 1)$ is the decay rate.

Example 9.5 (Understanding EMA Intuitively)**.** Think of α as a "forgetfulness" parameter. If $\alpha = 1$, the system only remembers the most recent evaluation—it has no memory. If $\alpha = 0$, the system never updates—it is frozen. In between, $\alpha = 0.15$ means each new evaluation contributes 15% of the new pressure value, while 85% comes from the accumulated history. This is like a weighted average where recent grades count more than old grades, but old grades still matter.

Equivalently, the pressure at time t is a weighted sum over all past signals:

$$p_t^{(d)} = \alpha \sum_{k=0}^{t-1} (1 - \alpha)^k\, s_{t-k}^{(d)}, \tag{9.4}$$

giving recent evaluations exponentially more influence than older ones. The effective memory horizon is $\tau_{\text{eff}} = 1/\alpha$ steps.

Choosing the Decay Rate

In the EVO-TRAINER implementation, $\alpha = 0.15$ provides a memory horizon of approximately 6–7 evaluations. This balances responsiveness to changing conditions against stability in the face of noisy evaluations. For rapidly shifting task distributions, a higher α (shorter memory) may be appropriate; for stable training regimes, a lower α preserves longer-term trends.

Worked Numerical Example: Pressure Update Over Time

Let us trace the grounding pressure through five time steps with $\alpha = 0.2$ and initial pressure $p_0^{(\text{gnd})} = 0.50$.

Suppose the evaluation signals are: $s_1 = 0.80$, $s_2 = 0.90$, $s_3 = 0.70$, $s_4 = 0.30$, $s_5 = 0.40$.

Table 9.1: Worked example: pressure update with $\alpha = 0.2$, $p_0 = 0.50$.

Step t	**Signal** s_t	**Old** p_{t-1}	**Calculation**	**New** p_t
1	0.80	0.50	$0.8 \times 0.50 + 0.2 \times 0.80$	0.560
2	0.90	0.56	$0.8 \times 0.56 + 0.2 \times 0.90$	0.628
3	0.70	0.628	$0.8 \times 0.628 + 0.2 \times 0.70$	0.642
4	0.30	0.642	$0.8 \times 0.642 + 0.2 \times 0.30$	0.574
5	0.40	0.574	$0.8 \times 0.574 + 0.2 \times 0.40$	0.539

Notice how the pressure responds to the signals but is smoothed. The high signals at $t = 1, 2$ push the pressure up gradually. The low signal at $t = 4$ begins pulling it down, but the effect is gradual—the system does not overreact to a single bad evaluation.

9.3.5 Regime Detection from Evaluation History

Given the pressure vector $\mathbf{R}_t$, the system identifies a *dominant regime* by finding the dimension under greatest pressure:

$$d^* = \arg \max_{d \in \{\text{gnd,ext,sch,eff,exp,coh}\}} p_t^{(d)}. \tag{9.5}$$

However, simple argmax regime detection is brittle. The Tera system uses a *pressure gap* threshold δ: the dominant regime is only asserted if $p_t^{(d^*)} - \bar{p}_t > \delta$, where $\bar{p}_t$ is the mean pressure. When no single dimension dominates, the system operates in a *balanced regime* that distributes mutation effort proportionally across all dimensions.

Example 9.6 (Regime Detection with Gap Threshold)**.** Suppose the current pressure vector is $\mathbf{R}_t = (0.6, 0.3, 0.2, 0.5, 0.8, 0.4)$ and $\delta = 0.15$. The mean pressure is $\bar{p}_t = (0.6 + 0.3 + 0.2 + 0.5 + 0.8 + 0.4)/6 = 0.467$. The maximum is $p^{(\text{exp})} = 0.8$, and the gap is $0.8 - 0.467 = 0.333 > 0.15$. So the system enters the *exploration regime*.

Now suppose $\mathbf{R}_t = (0.5, 0.45, 0.5, 0.48, 0.52, 0.47)$. The mean is 0.487, the maximum is 0.52, and the gap is $0.52 - 0.487 = 0.033 < 0.15$. No single dimension dominates, so the system operates in the *balanced regime*.

The Regime as Hidden State

The Tera regime state is "hidden" in the same sense as the hidden state of a hidden Markov model. It is not directly observed; it is *inferred* from a sequence of noisy evaluation signals. The regime vector $\mathbf{R}_t$ is the system's best estimate of which evolutionary pressures currently demand attention. This is directly analogous to how biological organisms infer environmental state from noisy sensory signals and adjust their regulatory responses accordingly.

9.4 Mutation Families

9.4.1 What Are Mutation Families?

In biological evolution, mutations come in different types: point mutations change a single nucleotide, insertions add new genetic material, deletions remove it, and transposons move segments around. Each type of mutation has a different effect on the organism. DOGMA mirrors this with **mutation families**—groups of related mutation operators that each address a specific type of weakness.

Definition 9.2 (Mutation Family)**.** A **mutation family** is a collection of related mutation operators that share a common purpose. Each family is associated with one pressure dimension and is activated when that dimension's pressure is high. Formally, family d contains a set of operators $\mathcal{F}_d = \{\mu_d^{(1)}, \mu_d^{(2)}, \ldots, \mu_d^{(m_d)}\}$, where each operator $\mu_d^{(i)} : \mathcal{G} \times \mathcal{C} \to \mathcal{G}$ maps a genome and context to a modified genome.

9.4.2 The Six Mutation Families

Table 9.2: Mutation families and their corresponding regime pressures.

Family	Pressure	Effect	Bio. Analogue
`context_grounding`	$p^{(\text{gnd})}$	Strengthen retrieval pathways and context anchoring	Promoter enhancement
`exact_answer`	$p^{(\text{ext})}$	Sharpen output precision and format adherence	Codon optimization
`strict_json`	$p^{(\text{sch})}$	Enforce structural output compliance	Reading frame correction
`brevity`	$p^{(\text{eff})}$	Reduce verbosity and computational waste	Intron splicing
`structural_explore`	$p^{(\text{exp})}$	Introduce novel genomic configurations	Transposon insertion
`bond_repair`	$p^{(\text{coh})}$	Restore internal consistency after mutations	DNA mismatch repair

Let us examine each family in more detail:

- **Context grounding family.** When the system's outputs drift from provided context, this family strengthens the genomic regions responsible for context retrieval. Operators include: enhancing context-attention bonds, adding retrieval markers, and strengthening source-output linkages. Biologically, this is analogous to *promoter enhancement*—making a gene more responsive to its regulatory signals.
- **Exact answer family.** When outputs are imprecise, this family sharpens the genome's answer-generation regions. Operators include: narrowing output distributions, strengthening pattern-matching bases, and adding precision markers. The biological analogue is *codon optimization*—selecting the specific nucleotide triplets that most efficiently encode a protein.
- **Strict JSON family.** When outputs fail structural validation, this family repairs and reinforces schema-compliance regions. Operators include: inserting structural delimiters, repairing bracket-matching bonds, and adding format-enforcement bases. This is analogous to *reading frame correction*—ensuring that the ribosome reads the mRNA in the correct triplet groupings.

- **Brevity family.** When outputs are verbose, this family trims unnecessary genomic material. Operators include: removing redundant bases, compressing repetitive regions, and streamlining output pathways. The biological analogue is *intron splicing*—removing non-coding sequences from mRNA before translation.

- **Structural exploration family.** When fitness plateaus, this family introduces novel configurations. Operators include: inserting new base sequences, rearranging genomic regions, and creating novel bond patterns. This is analogous to *transposon insertion*—mobile genetic elements that jump to new positions, potentially creating new gene functions.

- **Bond repair family.** When mutations destabilize the genome's internal structure, this family restores consistency. Operators include: recalculating bond energies, repairing broken cross-references, and re-establishing complementary strand agreement. The biological analogue is *DNA mismatch repair*—enzymatic systems that detect and correct base-pairing errors.

9.4.3 Adaptive Selection of Mutation Strategies

At each synthesis step, the system selects a mutation family with probability proportional to the corresponding pressure component, using a softmax function. Let $\mathbf{w}_t$ be the mutation selection distribution:

$$w_t^{(d)} = \frac{\exp(\beta\, p_t^{(d)})}{\sum_{d'} \exp(\beta\, p_t^{(d')})}, \tag{9.6}$$

where $\beta > 0$ is a temperature parameter controlling the sharpness of selection. As $\beta \to \infty$, only the highest-pressure family is selected; as $\beta \to 0$, all families are selected uniformly. In practice, β is set to maintain a moderate selection entropy, ensuring that subdominant pressures are not entirely neglected.

Example 9.7 (Computing Mutation Selection Probabilities)**.** Suppose $\mathbf{R}_t = (0.6, 0.3, 0.2, 0.5, 0.8, 0.4)$ and $\beta = 3$. Then:

$$\begin{aligned} &\exp(3 \times 0.6) = 6.05, \quad \exp(3 \times 0.3) = 2.46, \quad \exp(3 \times 0.2) = 1.82, \\ &\exp(3 \times 0.5) = 4.48, \quad \exp(3 \times 0.8) = 11.02, \quad \exp(3 \times 0.4) = 3.32. \end{aligned}$$

The sum is $6.05 + 2.46 + 1.82 + 4.48 + 11.02 + 3.32 = 29.15$. The selection probabilities are:

Family	GND	EXT	SCH	EFF	EXP	COH
$w_t^{(d)}$	0.208	0.084	0.063	0.154	**0.378**	0.114

The exploration family has a 37.8% chance of being selected—the highest—but all other families still have nonzero probability. This prevents the system from completely ignoring any dimension.

Definition 9.3 (Mutation Operator)**.** A **mutation operator** $\mu_d : \mathcal{G} \times \mathcal{C} \to \mathcal{G}$ maps a genome $G \in \mathcal{G}$ and context $C \in \mathcal{C}$ to a modified genome $G' = \mu_d(G, C)$, where d indexes the mutation family. Each operator is specialized to address the weakness identified by its corresponding pressure dimension.

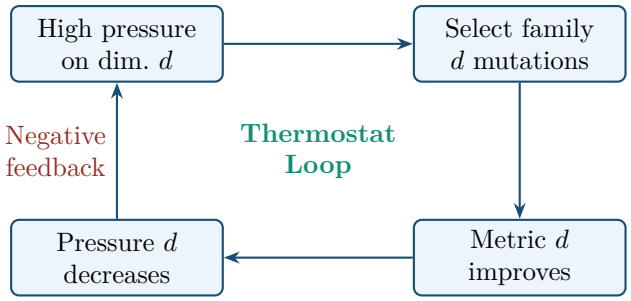


Figure 9.3: The pressure thermostat creates a homeostatic negative feedback loop. When a dimension's pressure is high, it receives more mutation effort, which improves its metric, which reduces its pressure, freeing resources for other dimensions.

9.4.4 Within-Family Operator Selection

Each mutation family contains multiple operators of varying intensity. When a family is selected, the specific operator within that family is chosen based on the magnitude of the pressure:

- **Low pressure** ($p^{(d)} < 0.3$): Apply mild operators that make small, conservative changes.
- **Medium pressure** ($0.3 \leq p^{(d)} < 0.7$): Apply moderate operators that make targeted adjustments.
- **High pressure** ($p^{(d)} \geq 0.7$): Apply aggressive operators that make substantial restructuring changes.

This two-level selection—first choosing a family, then choosing an operator within that family—gives the Tera system fine-grained control over both the *type* and *intensity* of mutations.

9.4.5 The Pressure Thermostat

A critical design feature of Tera is *automatic rebalancing*. When a mutation family succeeds—improving its corresponding evaluation metric—the associated pressure naturally decreases (since the evaluation signal $s_t^{(d)}$ improves). This creates a negative feedback loop:

1. High pressure on dimension $d \Rightarrow$ mutations from family d are preferentially selected.
2. Successful mutations improve the metric $\Rightarrow$ evaluation signal $s_t^{(d)}$ decreases.
3. Decreased signal $\Rightarrow$ pressure $p_{t+1}^{(d)}$ decreases via Eq. (9.3).
4. Lower pressure $\Rightarrow$ other dimensions receive more mutation effort.

Homeostatic Regulation

This negative feedback loop is directly analogous to homeostasis in biological systems. A thermostat does not heat and cool simultaneously; it senses the current temperature and applies the appropriate correction. Similarly, the Tera pressure thermostat senses which aspects of system performance are weakest and directs evolutionary effort accordingly. The result is a system that autonomously maintains balance across multiple competing objectives without manual tuning of loss weights.

The thermostat can be formalized as a dynamical system. Define the *deficit* $\delta_t^{(d)} = 1 - f_t^{(d)}$, where $f_t^{(d)} \in [0, 1]$ is the fitness on dimension d. Then the pressure update becomes:

$$p_{t+1}^{(d)} = (1 - \alpha)\, p_t^{(d)} + \alpha\, \delta_t^{(d)}. \tag{9.7}$$

At equilibrium ($p_{t+1}^{(d)} = p_t^{(d)}$), we have $p_\infty^{(d)} = \delta_\infty^{(d)}$: pressure equals deficit. The system reaches equilibrium when mutation effort is distributed in proportion to remaining deficiencies—a provably optimal allocation under mild assumptions about mutation effectiveness.

9.5 The Tera State Machine

9.5.1 Formal Definition

The Tera system can be modeled as a finite state machine that transitions between discrete regime modes based on the continuous pressure vector. This formalization helps us reason about the system's long-term behavior.

Tera State Machine

The Tera state machine is a tuple $\mathcal{T} = (S, s_0, \Sigma, \delta_T, \omega)$ where:

- $S = \{\textsc{Balanced}, \textsc{Grounding}, \textsc{Precision}, \textsc{Schema}, \textsc{Efficiency}, \textsc{Exploration}, \textsc{Coherence}, \textsc{Crisis}\}$ is the set of regime modes.
- $s_0 = \textsc{Balanced}$ is the initial state.
- $\Sigma = [0, 1]^6$ is the input alphabet (evaluation signal vectors).
- $\delta_T : S \times \Sigma \to S$ is the transition function.
- $\omega : S \to \Delta(\mathcal{F})$ maps each state to a mutation family distribution.

The eight regime modes are:

1. Balanced: No single pressure dominates. All mutation families are applied with roughly equal probability. This is the default, "healthy" state.
2. Grounding–Coherence: One of the six single-pressure regimes. The system has detected a clear weakness on one dimension and focuses mutation effort there.
3. Crisis: Multiple pressures are simultaneously high (all above a threshold θ_c). The system is failing on many dimensions and enters an emergency mode with aggressive, broad-spectrum mutations.

9.5.2 Transition Rules

The transition function δ_T is defined by the following rules, evaluated in order:

1. **Crisis entry:** If $\min_d p_t^{(d)} > \theta_c$ (all pressures above crisis threshold, typically $\theta_c = 0.7$), transition to Crisis regardless of current state.

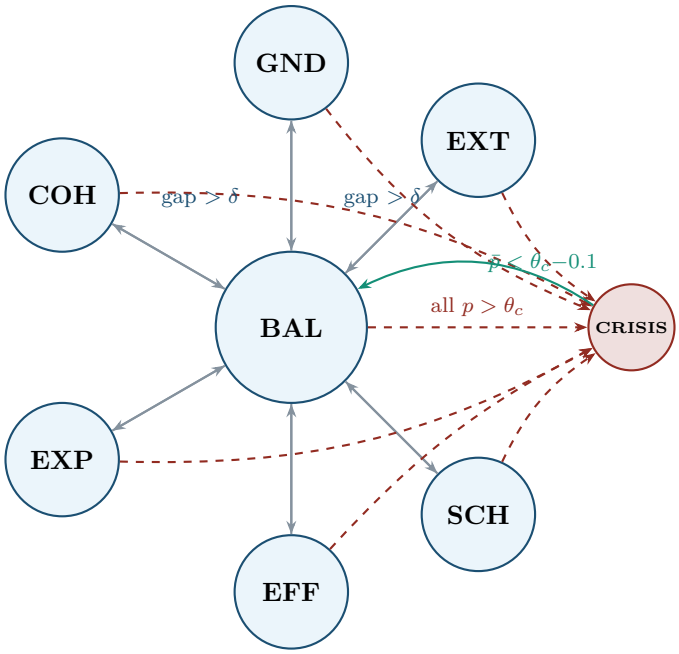


Figure 9.4: The Tera state machine. The system starts in the BALANCED state (center) and transitions to a specific regime when one pressure dimension dominates. It enters CRISIS when all pressures are simultaneously high. Transitions require hysteresis (h consecutive steps meeting the condition) to prevent oscillation.

2. **Crisis exit:** If in CRISIS and $\bar{p}_t < \theta_c - 0.1$ (mean pressure has dropped sufficiently), transition to BALANCED.

3. **Regime entry:** If $p_t^{(d^*)} - \bar{p}_t > \delta$ (pressure gap exceeds threshold), transition to the regime corresponding to d^*.

4. **Regime exit:** If in a single-pressure regime and $p_t^{(d^*)} - \bar{p}_t \leq \delta$ (the gap has closed), transition to BALANCED.

5. **Hysteresis:** To prevent rapid oscillation, a regime transition requires the condition to hold for h consecutive steps (typically $h = 2$).

9.5.3 State Diagram

9.5.4 Output Function: Mutation Distribution by State

Each state produces a different mutation family distribution:

Table 9.3: Mutation family selection weights $\omega(s)$ for each regime state. Higher values indicate stronger selection. The actual probabilities are computed via softmax (Eq. 9.6).

Regime	GND	EXT	SCH	EFF	EXP	COH
Balanced	1.0	1.0	1.0	1.0	1.0	1.0
Grounding	**3.0**	1.0	0.5	0.5	0.5	1.0
Precision	1.0	**3.0**	1.5	0.5	0.3	1.0
Schema	0.5	1.0	**3.0**	0.5	0.3	1.0
Efficiency	0.5	0.5	0.5	**3.0**	0.5	0.8
Exploration	0.5	0.3	0.3	0.5	**3.0**	1.5
Coherence	1.0	1.0	1.0	0.8	0.3	**3.0**
Crisis	2.0	2.0	2.0	1.0	2.5	2.5

Notice the design choices: the Exploration regime boosts coherence alongside exploration (to keep the genome stable during restructuring), and the Crisis regime elevates exploration and coherence (to find new solutions while preventing collapse).

9.6 Meta-Policy: Learning Which Mutations Work Best

9.6.1 The Problem of Starting from Scratch

Individual training runs are episodic: each run starts with an initial genome, evolves it over some number of generations, and produces a final evolved genome. Without cross-run memory, each run begins from scratch, unable to benefit from the regime dynamics learned in previous runs.

Consider this analogy: if you had to prepare for the same type of exam multiple times, you would not start from scratch each time. You would remember which study strategies worked best: "For math exams, I should start with practice problems early. For history exams, I need to focus on timeline memorization." This accumulated wisdom about *which strategies work in which situations* is exactly what Tera's meta-policy captures.

9.6.2 Cross-Run Persistent Learning

Tera addresses this through a **meta-policy memory**—a persistent store that records regime trajectories and their outcomes across runs:

$$\mathcal{M} = \{(\mathbf{R}_0^{(j)}, \mathbf{R}_1^{(j)}, \ldots, \mathbf{R}_{T_j}^{(j)}, \ \Delta F^{(j)})\}_{j=1}^{N}, \tag{9.8}$$

where $\mathbf{R}_t^{(j)}$ is the regime state at step t of run j, T_j is the length of run j, and $\Delta F^{(j)}$ is the overall fitness improvement achieved during that run.

The meta-policy memory stores three types of information:

1. **Regime trajectories:** The full sequence of regime states from each run, recording how pressures evolved over time.

2. **Transition success rates:** For each pair of regime modes (s_i, s_j), the average fitness change when the system transitioned from s_i to s_j.

3. **Mutation effectiveness:** For each mutation family in each regime mode, the average fitness improvement per application.

9.6.3 Shared Regime Priors

From the meta-policy memory, the system extracts *regime priors*—initial pressure vectors that reflect the typical pressure landscape for a given class of tasks. When a new run begins, instead of initializing $\mathbf{R}_0$ uniformly, the system computes:

$$\mathbf{R}_0^{(\text{new})} = \frac{\sum_{j=1}^{N} \Delta F^{(j)} \cdot \mathbf{R}_0^{(j)}}{\sum_{j=1}^{N} \Delta F^{(j)}}, \tag{9.9}$$

a fitness-weighted average of initial regime states from successful prior runs. This gives the new run a "warm start," directing early mutation effort toward dimensions that historically mattered most.

Example 9.8 (Warm Start in Practice)**.** Suppose three prior runs yielded these results:

Run	$\mathbf{R}_0$ (initial)	ΔF
1	$(0.3, 0.8, 0.5, 0.2, 0.4, 0.6)$	0.15
2	$(0.7, 0.3, 0.4, 0.6, 0.8, 0.3)$	0.25
3	$(0.5, 0.6, 0.3, 0.4, 0.7, 0.5)$	0.10

The warm-start prior is:

$$\mathbf{R}_0^{(\text{new})} = \frac{0.15 \cdot (0.3, 0.8, \ldots) + 0.25 \cdot (0.7, 0.3, \ldots) + 0.10 \cdot (0.5, 0.6, \ldots)}{0.15 + 0.25 + 0.10}.$$

Run 2 (the most successful, $\Delta F = 0.25$) contributes the most to the prior. If run 2 succeeded by starting with high grounding and exploration pressures, the new run will also start by prioritizing those dimensions.

9.6.4 Comparison to Meta-Learning

The meta-policy memory bears a structural resemblance to gradient-based meta-learning algorithms, particularly MAML (Model-Agnostic Meta-Learning) and Reptile.

Table 9.4: Comparison of Tera meta-policy memory with gradient-based meta-learning.

Property	**MAML / Reptile**	**Tera Meta-Policy**
What is learned	Initial model parameters θ_0	Initial regime state $\mathbf{R}_0$
Inner loop	Gradient descent on task loss	Evolutionary pressure-guided mutation
Outer loop	Meta-gradient across tasks	Fitness-weighted regime averaging
Dimensionality	$\lvert\theta\rvert$ (millions–billions)	6 (pressure dimensions)
Computational cost	Second-order gradients (MAML)	Negligible (vector averaging)
Memory footprint	Full parameter checkpoints	6-element vectors per run

Low-Dimensional Meta-Learning

Tera's meta-policy operates in a six-dimensional space—orders of magnitude smaller than the parameter spaces of MAML or Reptile. This makes meta-learning computationally trivial while capturing the essential information: *which pressures matter*. The insight is that one need not meta-learn the entire model; meta-learning the *regulatory strategy* is sufficient when the model itself is evolved, not trained by gradient descent.

9.7 Connection to Biological Evolution

9.7.1 Organisms Adapt to Environmental Pressures

The Tera regime system is not merely inspired by biology—it is a direct computational analogue of how organisms adapt to multi-pressure environments. Let us trace the parallels in detail.

The Lac Operon as a Regime Switch

The *lac operon* in *E. coli* is a classic example of a biological regime switch. When glucose is abundant, the bacterium uses glucose metabolism (the "glucose regime"). When glucose is scarce but lactose is present, the lac operon activates, switching the bacterium to lactose metabolism (the "lactose regime"). The regime is determined by two environmental signals: glucose level and lactose level. The regime change alters which genes are expressed—exactly analogous to how Tera's regime state determines which mutation families are activated.

Table 9.5: Mapping between biological regulatory concepts and Tera components.

Concept	Biological System	Tera System
Hidden state	Epigenetic marks, chromatin state	Regime vector $\mathbf{R}_t$
Environmental signal	Nutrient levels, temperature, pH	Evaluation signals $s_t^{(d)}$
State memory	Epigenetic inheritance	Exponential moving average
Behavioral switch	Gene regulatory networks	State machine transitions
Mutation type	Point mutation, transposon, recombination	Mutation families
Cross-generation learning	Lamarckian-like epigenetic inheritance	Meta-policy memory
Homeostasis	Feedback regulation (e.g., blood sugar)	Pressure thermostat

9.7.2 Multi-Scale Adaptation

Biological organisms adapt at multiple timescales:

1. **Immediate** (seconds–minutes): Enzyme regulation, allosteric control. *Tera analogue:* within-step mutation operator selection.

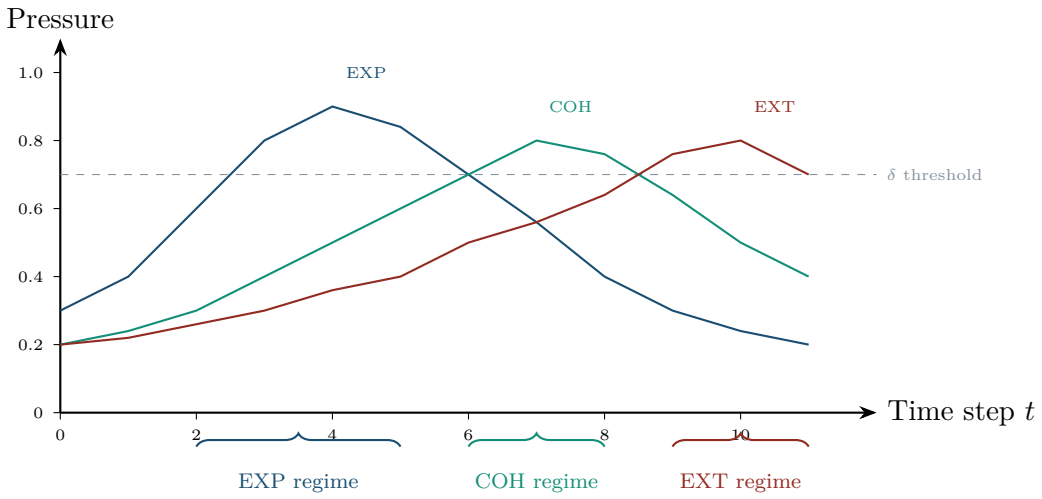


Figure 9.5: Pressure landscape over a training run showing three regime transitions. Initially, exploration pressure dominates (steps 2–5), driving the system to try new configurations. As exploration succeeds, coherence pressure rises (steps 6–8) to stabilize the new structures. Finally, exactness pressure dominates (steps 9–11) as the system refines precision. This sequence—explore, stabilize, refine—is a common pattern in DOGMA evolution.

2. **Short-term** (hours–days): Gene expression changes, stress responses. *Tera analogue:* pressure vector updates via EMA.

3. **Medium-term** (generations): Epigenetic inheritance, developmental plasticity. *Tera analogue:* regime state transitions across multiple steps.

4. **Long-term** (many generations): Genetic evolution, speciation. *Tera analogue:* meta-policy memory across runs.

The Tera system captures this multi-scale structure: the pressure vector changes smoothly within a run (short-term), regime transitions represent qualitative shifts in strategy (medium-term), and meta-policy memory accumulates wisdom across runs (long-term).

The Waddington Landscape

Conrad Waddington's "epigenetic landscape" depicts cell fate as a ball rolling down a landscape of valleys and ridges. The valleys represent stable cell types (regimes), and the ridges represent barriers between them. Tera's regime state machine is a computational analogue: each regime mode is a valley (attractor state), and the transition rules define the ridges (barriers). The hysteresis parameter h determines how deep the valleys are—higher hysteresis means deeper valleys and more stable regimes.

9.8 Pressure Landscape Visualization

To understand how pressures evolve during a training run, it helps to visualize the pressure landscape as a time series.

9.9 Worked Example: Five Evolutionary Steps

Let us walk through a complete example showing how the Tera system guides evolution over five steps. We use $\alpha = 0.2$, $\beta = 3$, $\delta = 0.15$, and initial regime state $\mathbf{R}_0 = (0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$ (balanced).

9.9.1 Step 1: Discovery of a Grounding Problem

Evaluation signals: $\mathbf{s}_1 = (0.9, 0.3, 0.2, 0.4, 0.3, 0.3)$.

The grounding signal is high (0.9), indicating the system is struggling with context fidelity.

Pressure update (Eq. 9.3):

$$\begin{aligned}\mathbf{R}_1 &= 0.8 \times (0.5, 0.5, 0.5, 0.5, 0.5, 0.5) + 0.2 \times (0.9, 0.3, 0.2, 0.4, 0.3, 0.3) \\ &= (0.58, 0.46, 0.44, 0.48, 0.46, 0.46).\end{aligned}$$

Regime detection: $\bar{p}_1 = 0.48$, max $= 0.58$ (GND), gap $= 0.10 < 0.15$. Remains in BALANCED.

Action: All mutation families selected with roughly equal probability. The softmax distribution gives GND a slight edge (≈20% vs. 15–16% for others).

9.9.2 Step 2: Grounding Problem Persists

Evaluation signals: $\mathbf{s}_2 = (0.95, 0.25, 0.2, 0.35, 0.2, 0.25)$.

Grounding signal remains very high.

Pressure update:

$$\begin{aligned}\mathbf{R}_2 &= 0.8 \times (0.58, 0.46, 0.44, 0.48, 0.46, 0.46) + 0.2 \times (0.95, 0.25, 0.2, 0.35, 0.2, 0.25) \\ &= (0.654, 0.418, 0.392, 0.454, 0.418, 0.418).\end{aligned}$$

Regime detection: $\bar{p}_2 = 0.459$, max $= 0.654$ (GND), gap $= 0.195 > 0.15$. **Transition to Grounding regime!**

Action: The `context_grounding` family receives boosted selection weight (3.0 vs. 0.5–1.0 for others). The system applies context-strengthening mutations.

9.9.3 Step 3: Grounding Improves, Schema Problem Emerges

Evaluation signals: $\mathbf{s}_3 = (0.4, 0.3, 0.85, 0.3, 0.25, 0.3)$.

Grounding improved (signal dropped to 0.4), but schema signal jumped to 0.85—the grounding mutations apparently disrupted output structure.

Pressure update:

$$\begin{aligned}\mathbf{R}_3 &= 0.8 \times (0.654, 0.418, 0.392, 0.454, 0.418, 0.418) + 0.2 \times (0.4, 0.3, 0.85, 0.3, 0.25, 0.3) \\ &= (0.603, 0.394, 0.484, 0.423, 0.384, 0.394).\end{aligned}$$

Regime detection: $\bar{p}_3 = 0.447$, max $= 0.603$ (GND), gap $= 0.156 > 0.15$. Remains in GROUNDING regime, but the gap is narrowing.

9.9.4 Step 4: Regime Transition to Schema

Evaluation signals: $\mathbf{s}_4 = (0.3, 0.35, 0.9, 0.3, 0.3, 0.35)$.
Grounding has been mostly fixed (0.3), but schema problems persist (0.9).
Pressure update:

$$\begin{aligned}\mathbf{R}_4 &= 0.8 \times (0.603, 0.394, 0.484, 0.423, 0.384, 0.394) + 0.2 \times (0.3, 0.35, 0.9, 0.3, 0.3, 0.35) \\ &= (0.542, 0.385, 0.567, 0.398, 0.367, 0.385).\end{aligned}$$

Regime detection: $\bar{p}_4 = 0.441$, max $= 0.567$ (SCH), gap $= 0.126 < 0.15$. Gap is just below threshold—system remains in GROUNDING (with hysteresis).

9.9.5 Step 5: Schema Takes Over

Evaluation signals: $\mathbf{s}_5 = (0.2, 0.3, 0.95, 0.25, 0.2, 0.3)$.
Grounding is fully resolved (0.2), schema is critical (0.95).
Pressure update:

$$\begin{aligned}\mathbf{R}_5 &= 0.8 \times (0.542, 0.385, 0.567, 0.398, 0.367, 0.385) + 0.2 \times (0.2, 0.3, 0.95, 0.25, 0.2, 0.3) \\ &= (0.474, 0.368, 0.644, 0.368, 0.334, 0.368).\end{aligned}$$

Regime detection: $\bar{p}_5 = 0.426$, max $= 0.644$ (SCH), gap $= 0.218 > 0.15$. **Transition to Schema regime!**
Action: The `strict_json` family now receives boosted selection weight, and the system focuses on repairing output structure.

Regime Cascades

This example illustrates a common pattern: fixing one problem can reveal or create another. The grounding mutations at step 2–3 improved context fidelity but disrupted output schema. The Tera system detected this and smoothly transitioned to address the new dominant weakness. This cascading behavior is a feature, not a bug—it reflects the reality that complex systems have interconnected failure modes.

Table 9.6: Summary of the five-step worked example showing regime transitions.

Step	Regime	Max Press.	Gap	Key Event
1	BALANCED	GND: 0.58	0.10	Grounding problem detected but below threshold
2	GROUNDING	GND: 0.654	0.195	Gap exceeds δ; grounding mutations activated
3	GROUNDING	GND: 0.603	0.156	Grounding improving; schema emerging as issue
4	GROUNDING	SCH: 0.567	0.126	Schema rising but gap below threshold
5	SCHEMA	SCH: 0.644	0.218	Schema mutations now dominant

9.10 Extended Mutation Family Catalog

Each mutation family contains operators of varying intensity. The following table catalogs the primary operators and their characteristics.

Table 9.7: Extended mutation operator catalog by family and intensity level.

Family	Intensity	Operator	Description
ctx_gnd	Low	Bond strengthen	Increase weight of context-retrieval bonds
	Medium	Path insert	Add new retrieval pathways between context and output
	High	Anchor rebuild	Restructure context integration architecture
exact_ans	Low	Precision tune	Fine-adjust output distribution sharpness
	Medium	Pattern graft	Insert pattern-matching base sequences
	High	Answer rewrite	Rebuild answer-generation genomic region
strict_json	Low	Delimiter fix	Repair bracket/brace matching bonds
	Medium	Schema insert	Add structural enforcement base sequences
	High	Format rebuild	Complete restructuring of output formatting
brevity	Low	Trim redundant	Remove obviously redundant base sequences
	Medium	Compress region	Merge repetitive genomic regions
	High	Pathway prune	Remove entire unnecessary processing pathways
struct_exp	Low	Local shuffle	Rearrange bases within a small region
	Medium	Region insert	Add a novel base sequence from the archive
	High	Major restructure	Large-scale rearrangement of genomic architecture
bond_rep	Low	Bond recalc	Recalculate bond energies for consistency
	Medium	Strand sync	Resynchronize forward and reverse strands
	High	Full repair	Complete structural integrity restoration

9.11 The Tera Double-Helix Complex

The regime state $\mathbf{R}_t$ is one component of a larger structure called the **Tera Double-Helix Complex**—the complete internal state of the Evolutor at time t.

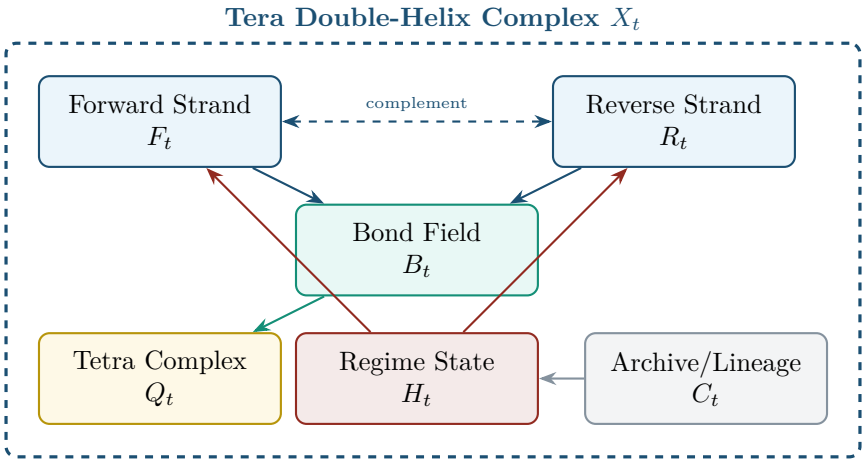


Figure 9.6: The six components of the Tera Double-Helix Complex. The forward and reverse strands are complementary (dashed arrow). The bond field mediates their interaction. The regime state controls mutation strategy selection. The archive and lineage context provides historical information.

Tera Double-Helix Complex

The full Evolutor state at time t is the six-tuple:

$$X_t = (F_t,\ R_t,\ B_t,\ Q_t,\ H_t,\ C_t), \tag{9.10}$$

where:

- $F_t \in \mathcal{G}$ is the **forward strand state**—the primary genomic sequence.
- $R_t \in \mathcal{G}^*$ is the **reverse strand state**—a complementary representation that enables bidirectional processing and error detection.
- $B_t \in \mathcal{B}^{n\times n}$ is the **typed bond field**—a sparse matrix of inter-base bonds with type annotations (hydrogen, covalent, stacking), encoding the secondary structure of the genome.
- $Q_t \in \mathcal{T}$ is the **tetra structural complex**—a tetrahedral mesh over the genomic bases providing three-dimensional spatial organization (see Chapter 8).
- $H_t \in [0,1]^6$ is the **hidden latent regime state**—the Tera pressure vector $\mathbf{R}_t$ defined in Eq. (9.2).
- $C_t = (\mathcal{A}_t, \mathcal{L}_t)$ is the **archive and lineage context**—the HelixHash novelty archive $\mathcal{A}_t$ and the evolutionary lineage history $\mathcal{L}_t$.

The double-helix metaphor is deliberate: the forward strand F_t and reverse strand R_t are complementary representations of the same underlying information, analogous to the Watson and Crick strands of biological DNA. The bond field B_t records how these strands interact, while the tetra complex Q_t provides the three-dimensional scaffold within which these interactions take place.

Proposition 9.1 (State Completeness)**.** The Tera Double-Helix Complex X_t is a *sufficient statistic* for the Evolutor's behavior: given X_t and a context input c_{t+1}, the system's output and next state X_{t+1} are fully determined (up to stochastic mutation choices). No external state is required.

Proof sketch. Each component of X_t contributes to a distinct aspect of the Evolutor's computation. The forward/reverse strands provide the genomic content for transcription. The bond field determines which regions interact during regulation. The tetra complex provides the spatial structure for local propagation. The regime state H_t determines mutation strategy selection. The archive/lineage context provides novelty computation and fitness history. Since the six DOGMA stages (Chapter 7) depend only on these quantities and the incoming context, the state is complete. □

9.11.1 Forward and Reverse Strand Dynamics

The forward and reverse strands evolve according to coupled dynamics:

$$F_{t+1} = \mu_{d^*}(F_t,\ c_{t+1}) + \eta_F(B_t, R_t), \tag{9.11}$$
$$R_{t+1} = \text{Complement}(F_{t+1}) + \eta_R(B_t, F_{t+1}), \tag{9.12}$$

where μ_{d^*} is the mutation operator from the dominant regime family, Complement$(\cdot)$ computes the type-complementary strand, and η_F, η_R are bond-mediated correction terms that enforce structural consistency. The correction terms act as a form of *proofreading*—when the bond field indicates that a mutation has disrupted critical interactions, the correction terms partially restore them, analogous to DNA polymerase proofreading in biology.

9.12 Toward Post-Transformer Architectures

The Tera regime system, combined with the full Double-Helix Complex, suggests a radical architectural direction.

The Post-Transformer Thesis

Replace dense attention with regulated propagation over tetra-helix state space.

In a standard transformer, information flow between positions is mediated by dense attention: every position attends to every other position, computing $O(T^2)$ pairwise interactions. In DOGMA with the Tera regime, information flow is *regulated*:

1. **Spatial locality from the tetra complex.** Information propagates through the tetrahedral mesh Q_t, with each base interacting only with its local neighborhood. Long-range communication requires multiple propagation steps, analogous to signal transduction in gene regulatory networks (Chapter 3).

2. **Selective activation from the regime state.** The regime vector H_t determines which genomic regions are active, creating sparse activation patterns that reduce effective sequence length.

3. **Structural routing from the bond field.** The typed bond field B_t provides learned "wiring" between distant regions, enabling long-range interactions without quadratic cost.

The combined complexity is $O(T \cdot k \cdot d)$ per propagation step, where k is the average neighborhood size in the tetra mesh—linear in sequence length T rather than quadratic. This is not merely a computational optimization; it is a different *computational paradigm*: one where structure determines interaction, rather than interaction being universal.

Comparison with State-Space Models

The regulated-propagation paradigm shares features with state-space models (SSMs) such as Mamba, which also achieve linear-time sequence processing. The key difference is that SSMs use a fixed state-transition structure, while DOGMA's propagation structure is *evolved* and *regime-dependent*. The tetra complex and bond field are not fixed matrices but dynamic, evolvable structures that change during training and even during inference. This gives DOGMA a form of *architectural plasticity* absent from SSMs.

9.13 Exercises

9.1. **[Conceptual]** In your own words, explain what a "regime state" is using an analogy from everyday life that was *not* mentioned in this chapter. Your analogy should capture: (a) the idea of hidden state, (b) transitions between regimes, and (c) different behavior in different regimes.

9.2. **[Fundamentals]** Consider a system with $\alpha = 0.2$ and initial pressure $p_0^{(\text{gnd})} = 0.5$. If the grounding evaluation signal is $s_t^{(\text{gnd})} = 0.9$ for all $t \geq 1$, compute $p_1^{(\text{gnd})}, p_2^{(\text{gnd})}, p_3^{(\text{gnd})}$ using Eq. (9.3). How many steps does it take for the pressure to reach 0.85?

9.3. **[Analysis]** Prove that the exponential moving average in Eq. (9.4) assigns a total weight of $1 - (1-\alpha)^t$ to the most recent t observations. What fraction of the total weight lies within the effective memory horizon $\tau_{\text{eff}} = 1/\alpha$?

9.4. **[Computation]** Given the pressure vector $\mathbf{R} = (0.7, 0.4, 0.3, 0.5, 0.6, 0.35)$ and $\beta = 4$, compute the full mutation family selection distribution using Eq. (9.6). Which family has the highest selection probability? What is the entropy of the distribution?

9.5. **[State Machine]** Draw the state machine transitions for a system that starts in BALANCED and receives the following dominant pressure sequence: GND, GND, GND, SCH, SCH, (balanced), (balanced), EXP, EXP. Assume $\delta = 0.15$ and hysteresis $h = 2$. At each step, state the current regime mode and whether a transition occurs.

9.6. **[Design]** Design a seventh pressure dimension that you believe is missing from the Tera regime state. Specify: (a) what it measures, (b) how it is computed from evaluation signals, (c) what mutation family it would drive, and (d) what biological process it is analogous to. Argue for or against its inclusion.

9.7. **[Coding]** Implement the Tera pressure thermostat in Python. Your implementation should: (a) maintain a 6-dimensional pressure vector, (b) accept evaluation signals and update pressures via Eq. (9.3), (c) compute mutation family selection probabilities via Eq. (9.6), and (d) detect the dominant regime with a gap threshold $\delta = 0.1$. Test with a synthetic sequence of evaluation signals that transitions from a precision-dominant to an exploration-dominant regime.

9.8. **[Theory]** Show that the pressure thermostat (Eq. (9.7)) has a unique fixed point for any constant deficit vector $\boldsymbol{\delta}$. Analyze the convergence rate and show that it depends on α but not on the dimensionality of the pressure space.

9.9. **[Biology]** The lac operon in *E. coli* can be modeled as a two-state regime system (glucose mode vs. lactose mode). Define this as a formal state machine in the style of Definition 9.5.

What are the "evaluation signals"? What are the "mutation families" (i.e., which genes are expressed in each regime)?

9.10. **[Research]** Compare the Tera meta-policy memory (Eq. (9.8)) with the outer loop of Reptile. Both compute a form of averaged update across tasks. What are the theoretical advantages and disadvantages of operating in a 6-dimensional pressure space versus a high-dimensional parameter space? Under what conditions would the low-dimensional approach fail?

9.11. **[Worked Example]** Extend the five-step worked example (Section 9.9) by five more steps. Invent evaluation signals that cause the system to enter Crisis mode at step 7 and recover to Balanced by step 10. Show all pressure calculations.

9.12. **[Integration]** Explain how the Tera regime state interacts with the HelixHash novelty archive (Chapter 8). Specifically: when the exploration pressure $p^{(\text{exp})}$ is high, how does the novelty archive influence which `structural_explore` mutations are selected? How does the archive's novelty score feed back into the exploration pressure signal?

9.14 Key Takeaways

Key Takeaways

- A **regime state** is a hidden variable that determines which behavioral strategy a system should follow. It is inferred from noisy evaluation signals, not directly observed.
- Single-objective optimization is insufficient for complex systems; biological organisms simultaneously optimize across many dimensions via dynamic regulatory networks.
- The **Tera regime state** is a six-dimensional pressure vector $\mathbf{R}_t \in [0,1]^6$ capturing grounding, exactness, schema, efficiency, exploration, and coherence pressures.
- **Exponential memory decay** (Eq. (9.3)) gives recent evaluations more influence while preserving longer-term trends, with effective memory horizon $\tau_{\text{eff}} = 1/\alpha$.
- Six **mutation families**—`context_grounding`, `exact_answer`, `strict_json`, `brevity`, `structural_explore`, `bond_repair`—are adaptively selected via softmax over regime pressures.
- The **pressure thermostat** creates a homeostatic negative feedback loop that automatically rebalances evolutionary effort across objectives. At equilibrium, pressure equals deficit.
- The **Tera state machine** formalizes regime transitions with eight modes (Balanced + six single-pressure + Crisis), with hysteresis to prevent oscillation.
- **Meta-policy memory** enables cross-run learning in a six-dimensional space—computationally trivial meta-learning of regulatory strategy via fitness-weighted averaging.
- The full **Tera Double-Helix Complex** $X_t = (F_t, R_t, B_t, Q_t, H_t, C_t)$ is a sufficient statistic for Evolutor behavior.
- Tera's design mirrors biological regulation at multiple timescales, from immediate enzyme control to long-term evolutionary adaptation.
- Regulated propagation over tetra-helix state space is a candidate post-transformer architecture with linear-time complexity.

Suggested Reading

- Allis and Jenuwein [2016]: Epigenetics—the biological inspiration for Tera's hidden regulatory layer.
- Deb [2002]: NSGA-II and multi-objective evolutionary algorithms, foundational to the Pareto perspective.
- Finn et al. [2017]: MAML—the gradient-based meta-learning algorithm compared with Tera's meta-policy memory.
- Gu and Dao [2023]: Mamba and state-space models, the closest existing architecture to DOGMA's regulated propagation.

- Chapter 7 of this book: the six-stage pipeline that Tera regulates.
- Chapter 12 of this book: the evolutionary training loop where Tera's mutation steering operates.

Chapter 10

TetraData Structure and TetraMemory

The tetrahedron is the simplest possible solid. Any geometry must begin here.

— After Buckminster Fuller

Computer graphics has long relied on *triangles* as the fundamental rendering primitive. Triangle meshes approximate surfaces efficiently and map naturally to GPU hardware. But triangles are inherently *two-dimensional*: they describe surfaces, not volumes. For a computational architecture inspired by three-dimensional molecular biology—where proteins fold into globular structures, DNA coils into superhelical conformations, and tetrahedral carbon bonds define organic chemistry—a volumetric primitive is needed.

This chapter introduces the **TetraData Structure** (TDS): a computational framework built on *tetrahedra* rather than triangles. We begin with a patient, step-by-step construction that builds a tetrahedron one vertex at a time, then show how tetrahedra link together into chains that map directly onto DNA sequences. We then develop **TetraMemory**, a structured memory system that replaces self-attention in DOGMA with local tetrahedral interactions that compose into global computation.

10.1 Building a Tetrahedron from Scratch

Before defining the TetraData Structure formally, let us build our fundamental primitive—the tetrahedron—one vertex at a time. This construction will reveal *why* the tetrahedron is the minimal solid that carries volume and chirality, and why no simpler primitive will do.

10.1.1 Step 0: A Single Vertex (0D)

We begin with nothing: an empty space. We place a single point v_0 and assign it a value (say, a DNA base, a token embedding, or simply a label).

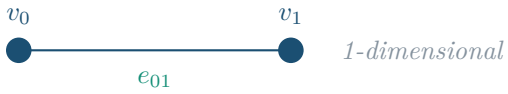
Vertices: 1 Edges: 0 Faces: 0 Volume: 0

A single vertex is a 0-dimensional object. It can store a value—but it encodes no *relationships*. There is no notion of distance, connection, area, or volume. In a sequence model, a single vertex is a single token in isolation, with no context whatsoever.

Remark 10.1. In graph theory terms, we have the graph K_1: one vertex, zero edges. The adjacency matrix is the 1×1 zero matrix. This is the least informative data structure possible.

10.1.2 Step 1: Add a Second Vertex (1D — an Edge)

Now add a second vertex v_1 and connect it to v_0 with an edge e_{01}.



Vertices: 2 Edges: 1 Faces: 0 Volume: 0

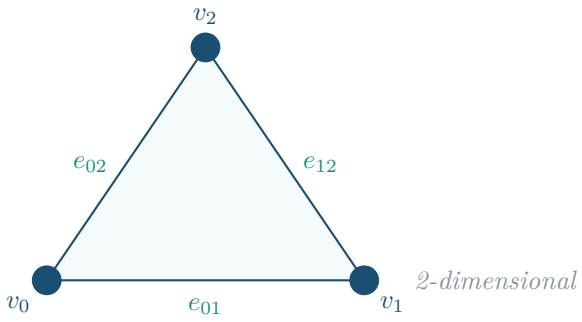
An edge is a 1-dimensional object. It encodes exactly one relationship: the pairwise interaction between v_0 and v_1. The edge can carry a weight $w_{01} \in \mathbb{R}$ representing the strength or character of that interaction.

In the language of neural networks, this is a single attention weight: how much does token v_0 attend to token v_1? One edge captures one pairwise interaction. This is the graph K_2: the complete graph on 2 vertices.

Remark 10.2. An edge has no *area*, no *volume*, and no *handedness*. You cannot distinguish a "left-handed" line segment from a "right-handed" one. An edge is symmetric under reflection.

10.1.3 Step 2: Add a Third Vertex (2D — a Triangle)

Add a third vertex v_2 and connect it to both existing vertices. We now have three vertices and three edges forming a triangle.



Vertices: 3 Edges: 3 Faces: 1 Volume: 0

A triangle is a 2-dimensional object. It has *area* but no *volume*. Three edges encode all $\binom{3}{2} = 3$ pairwise interactions among the three vertices, forming the complete graph K_3.

This is the primitive used in computer graphics: triangle meshes tile surfaces. GPUs are optimized for rasterizing triangles. But triangles have two critical limitations for our purposes:

1. **No volume.** A triangle is flat. It cannot enclose space. It describes a boundary, not an interior.

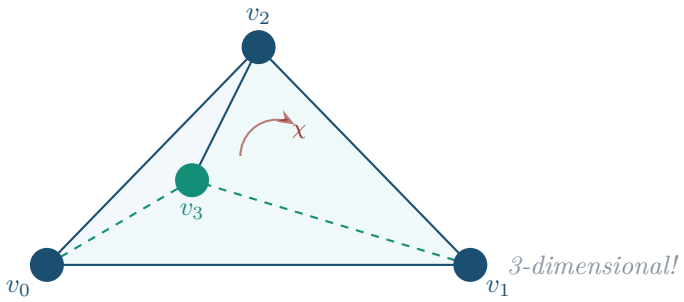
2. **No chirality.** A triangle in 2D can be reflected into its mirror image by a rotation. In 3D, a triangle embedded in a plane does have two "sides," but this orientation is not an intrinsic property—it depends on the choice of normal direction.

Why Triangles Are Not Enough

In molecular biology, chirality is not optional—it is essential. All naturally occurring amino acids are L-amino acids (left-handed). DNA's double helix is always right-handed (B-form). Enzymes distinguish between mirror-image molecules with exquisite specificity. A computational primitive that cannot represent chirality is fundamentally incapable of encoding molecular structure faithfully. A triangle mesh can approximate the *surface* of a protein, but it cannot capture the *handed* internal packing that determines function.

10.1.4 Step 3: Add a Fourth Vertex (3D — the Tetrahedron!)

Now the crucial step. Add a fourth vertex v_3 that is *not in the plane* of the first three, and connect it to all existing vertices. We obtain a tetrahedron: the simplest possible solid.



Vertices: 4 Edges: 6 Faces: 4 Volume: > 0

A tetrahedron is the *first* object in our construction that possesses:

- **Volume:** It encloses a region of 3D space. The signed volume is computed from the 3×3 determinant of the edge vectors.
- **Chirality:** The sign of the determinant ($+1$ or -1) defines a handedness that cannot be changed by any rotation—only by reflection.
- **Complete pairwise interactions:** Six edges encode all $\binom{4}{2} = 6$ pairwise relationships. This is the complete graph K_4.
- **Four faces:** Each face is a triangle that can serve as an interface for connecting to another tetrahedron.

Definition 10.1 (Tetrahedron)**.** A *tetrahedron* $\mathcal{T}$ in $\mathbb{R}^3$ is the convex hull of four non-coplanar points $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$:

$$\mathcal{T} = \left\{ \sum_{i=0}^{3} \lambda_i \mathbf{v}_i \;\middle|\; \lambda_i \geq 0,\ \sum_{i=0}^{3} \lambda_i = 1 \right\}. \tag{10.1}$$

The *signed volume* is:

$$V_{\text{signed}}(\mathcal{T}) = \frac{1}{6} \det \begin{pmatrix} \mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 & \mathbf{v}_3 - \mathbf{v}_0 \end{pmatrix}, \tag{10.2}$$

and the *chirality* is:

$$\chi(\mathcal{T}) = \text{sign}(V_{\text{signed}}(\mathcal{T})) \in \{+1, -1\}. \tag{10.3}$$

The Tetrahedron Is the Minimal Solid

In all of Euclidean geometry, the tetrahedron is the *simplest* object that simultaneously possesses volume and chirality. You cannot build a solid with fewer than 4 vertices. This is not a design choice—it is a mathematical necessity. Just as the triangle is the minimal polygon, the tetrahedron is the minimal polyhedron.
DOGMA's choice of the tetrahedron as its fundamental primitive is therefore not arbitrary: it is the minimal structure that can encode 3D geometry, handedness, and complete pairwise interactions among a set of elements.

10.1.5 The Construction at a Glance

The following figure summarizes the four steps side by side, highlighting what each new vertex contributes:

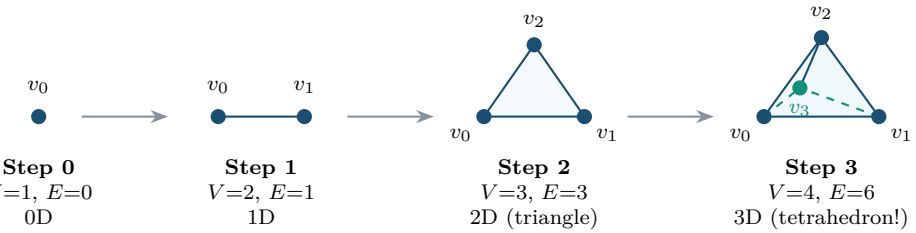


Figure 10.1: Building a tetrahedron one vertex at a time. Each step adds a new dimension: a vertex (0D), an edge (1D), a triangle (2D), and finally a tetrahedron (3D). The tetrahedron is the first object with volume and chirality.

Example 10.1 (Counting Combinatorics)**.** At each step k (adding vertex v_k to the existing structure), the new vertex connects to all k existing vertices, adding exactly k new edges:

Step	Vertices	New edges	Total edges	Graph
0	1	0	0	K_1
1	2	1	1	K_2
2	3	2	3	K_3
3	4	3	6	K_4

In general, the complete graph K_n has $\binom{n}{2} = \frac{n(n-1)}{2}$ edges. For K_4, this gives $\binom{4}{2} = 6$ edges—exactly the six edges of a tetrahedron.

10.2 Why Tetrahedra? — A Deeper Look

With the construction complete, we can appreciate why DOGMA chooses the tetrahedron over simpler primitives.

10.2.1 Tetrahedra vs. Triangles: A Detailed Comparison

Property	Triangle Mesh	Tetrahedral Mesh (TDS)
Dimension	2D surface in 3D	3D volume
Vertices per element	3	4
Edges per element	3 (K_3)	6 (K_4) — 2× more interactions
Faces per element	1	4 triangular faces
Interior	None (boundary only)	Enclosed volume
Chirality	None	±1 handedness
Adjacency	Edge-sharing	Face-sharing (richer interface)
DNA mapping	—	4 vertices ↔ {A, C, G, T}
DOF per element	9 (positions only)	36 (positions + edges + bases + vol + chir)
Info. density	3 edges	6 edges — 4× information density

The last row deserves emphasis. A triangle encodes 3 pairwise interactions. A tetrahedron encodes 6. But the tetrahedron has only one more vertex. In terms of pairwise interactions per vertex, the triangle achieves $3/3 = 1$ edge per vertex, while the tetrahedron achieves $6/4 = 1.5$ edges per vertex—a 50% improvement in information density.

10.2.2 The Biological Justification

The biological case for tetrahedra is compelling:

- **Carbon chemistry:** The carbon atom—the backbone of all organic molecules—forms four bonds arranged tetrahedrally (sp^3 hybridization). Methane (CH_4) is a perfect tetrahedron. Protein backbone geometry is fundamentally tetrahedral at each C_α center.

- **DNA base count:** DNA uses exactly four bases—A, C, G, T. A tetrahedron has exactly four vertices. This numerical coincidence becomes the foundation of DOGMA's data structure: each vertex of a tetrahedron is assigned one DNA base.

- **DNA base stacking:** Adjacent base pairs in the double helix create local geometries that are naturally described by tetrahedral coordinates.

- **Protein structure:** Alpha-carbon (C_α) geometry in protein backbones forms tetrahedral arrangements that are native to TDS representation.

- **Chirality in biology:** All naturally occurring amino acids are L-amino acids; DNA forms a right-handed helix. Chirality is *information* that must be preserved, and only volumetric primitives (3D and above) can encode it.

Four Bases, Four Vertices

The fact that DNA uses exactly four nucleotide bases is one of the most fundamental constraints in molecular biology. This four-fold alphabet maps perfectly onto the four vertices of a tetrahedron. This is not merely a convenient analogy: it is the mathematical foundation of the TetraData Structure. Each vertex carries a base type, and the six edges encode all pairwise interactions among those bases—including the Watson–Crick pairings (A–T and G–C) as geometrically distinguished edges.

10.3 Linking Tetrahedra — The LinkedTetraChain

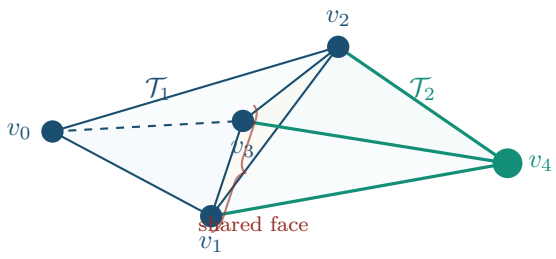
A single tetrahedron encodes the interactions among four elements. But real sequences are longer than four tokens. How do we handle sequences of arbitrary length? The answer is *linking*: we chain tetrahedra together by sharing triangular faces, exactly as the PowerPoint construction shows.

10.3.1 Adding Vertex 4: The Second Tetrahedron

We continue the construction from Section 10.1. Having built the first tetrahedron $\mathcal{T}_1 = (v_0, v_1, v_2, v_3)$, we now add vertex v_4 and connect it to the three most recent vertices: v_1, v_2, v_3. This creates a *second* tetrahedron:

$$\mathcal{T}_2 = (v_1, v_2, v_3, v_4).$$

The two tetrahedra share the triangular face (v_1, v_2, v_3). The new vertex v_4 adds exactly 3 new edges: e_{14}, e_{24}, e_{34}.



Remark 10.3. The shared face (v_1, v_2, v_3) contains three edges and three vertices that belong to *both* tetrahedra. This is a much richer interface than the edge-sharing used by triangle meshes: a shared face transmits three pairwise relationships, while a shared edge transmits only one.

10.3.2 Adding Vertex 5: The Third Tetrahedron

Continuing the pattern, vertex v_5 connects to v_2, v_3, v_4, creating:

$$\mathcal{T}_3 = (v_2, v_3, v_4, v_5),$$

which shares face (v_2, v_3, v_4) with $\mathcal{T}_2$.

10.3.3 The General Pattern: LinkedTetraChain

The pattern is now clear. At each step, a new vertex v_{k+3} connects to the three most recent vertices (v_k, v_{k+1}, v_{k+2}), forming tetrahedron:

$$\mathcal{T}_{k+1} = (v_k, v_{k+1}, v_{k+2}, v_{k+3}),$$

which shares face (v_k, v_{k+1}, v_{k+2}) with the previous tetrahedron $\mathcal{T}_k = (v_{k-1}, v_k, v_{k+1}, v_{k+2})$.

Definition 10.2 (LinkedTetraChain)**.** A *LinkedTetraChain* of length n is an ordered sequence of tetrahedra $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n)$ constructed from a vertex sequence $(v_0, v_1, \ldots, v_{n+2})$ where:

$$\mathcal{T}_k = (v_{k-1}, v_k, v_{k+1}, v_{k+2}), \quad k = 1, 2, \ldots, n, \tag{10.4}$$

and consecutive tetrahedra $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$ share the triangular face (v_k, v_{k+1}, v_{k+2}).

Proposition 10.1 (Edge Count Formula)**.** A LinkedTetraChain of n tetrahedra built from $n+3$ vertices has exactly $3n+3$ edges.

Proof. The first tetrahedron $\mathcal{T}_1$ contributes $\binom{4}{2} = 6$ edges. Each subsequent tetrahedron $\mathcal{T}_k$ for $k \geq 2$ adds one new vertex that connects to 3 existing vertices, contributing exactly 3 new edges. Thus the total is $6 + 3(n-1) = 3n+3$. □

The following diagram shows a chain of four linked tetrahedra:

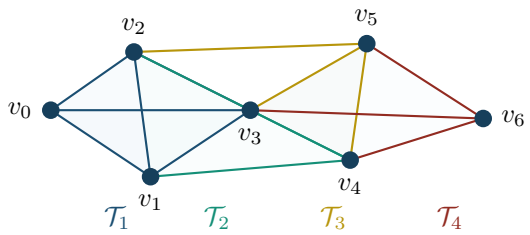


Figure 10.2: A LinkedTetraChain of four tetrahedra. Each tetrahedron shares a triangular face with its neighbors. Colors distinguish the edges added by each new vertex. Total: 7 vertices, $3(4)+3 = 15$ edges.

The Sliding Window Analogy

The LinkedTetraChain is equivalent to a *sliding window of width 4 with overlap 3* over the vertex sequence. Each tetrahedron "sees" four consecutive vertices, and consecutive tetrahedra overlap in three vertices. This means that each vertex participates in up to four tetrahedra (except near the ends), ensuring that information about each vertex is propagated through multiple local interaction contexts.

Compare this to the sliding window in signal processing: a window of width W with overlap $W-1$ slides one position at a time. Here, $W = 4$ and overlap $= 3$, so each step advances by one vertex. But unlike a flat sliding window, each "window" is a tetrahedron with volume, chirality, and full K_4 connectivity.

10.3.4 Growth Table

The following table tracks the combinatorics as we add vertices:

Vertex	Total $\lvert V\rvert$	New edges	Total $\lvert E\rvert$	Total $\lvert\mathcal{T}\rvert$	Shared face	Graph
v_0	1	0	0	0	—	K_1
v_1	2	1	1	0	—	K_2
v_2	3	2	3	0	—	K_3
v_3	4	3	6	1 ($\mathcal{T}_1$)	—	K_4
v_4	5	3	9	2 ($\mathcal{T}_2$)	(v_1, v_2, v_3)	—
v_5	6	3	12	3 ($\mathcal{T}_3$)	(v_2, v_3, v_4)	—
v_6	7	3	15	4 ($\mathcal{T}_4$)	(v_3, v_4, v_5)	—
v_{k+3}	k+4	3	3(k+1)+3	k+1	(v_k, v_{k+1}, v_{k+2})	—

Notice that after the first tetrahedron is formed, each new vertex adds exactly 3 new edges and exactly 1 new tetrahedron. The growth is perfectly linear.

10.4 DNA Mapping — The Key Insight

We now arrive at the central insight of the TetraData Structure: the mapping between tetrahedra and DNA sequences.

10.4.1 Four Vertices, Four Bases

A tetrahedron has four vertices. DNA has four bases: adenine (A), cytosine (C), guanine (G), and thymine (T). In a LinkedTetraChain, each vertex is assigned one base type from $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$.

TetraData–DNA Correspondence

In the TetraData Structure, each vertex v_i carries a base assignment $b_i \in \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. A tetrahedron $\mathcal{T}_k = (v_{k-1}, v_k, v_{k+1}, v_{k+2})$ with base assignments $(b_{k-1}, b_k, b_{k+1}, b_{k+2})$ represents a *window* of four consecutive bases in a DNA-like sequence.

10.4.2 Worked Example: Mapping 5′-ACGTACGT-3′

Consider the DNA sequence 5′-ACGTACGT-3′, which has 8 bases. We assign these to vertices v_0 through v_7:

Vertex	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Base	A	C	G	T	A	C	G	T

This sequence of 8 vertices produces a LinkedTetraChain of $8 - 3 = 5$ tetrahedra:

Tetra	Vertices	Bases	Shared face with prev.
$\mathcal{T}_1$	(v_0, v_1, v_2, v_3)	(A, C, G, T)	—
$\mathcal{T}_2$	(v_1, v_2, v_3, v_4)	(C, G, T, A)	$(v_1, v_2, v_3) = (\text{C, G, T})$
$\mathcal{T}_3$	(v_2, v_3, v_4, v_5)	(G, T, A, C)	$(v_2, v_3, v_4) = (\text{G, T, A})$
$\mathcal{T}_4$	(v_3, v_4, v_5, v_6)	(T, A, C, G)	$(v_3, v_4, v_5) = (\text{T, A, C})$
$\mathcal{T}_5$	(v_4, v_5, v_6, v_7)	(A, C, G, T)	$(v_4, v_5, v_6) = (\text{A, C, G})$

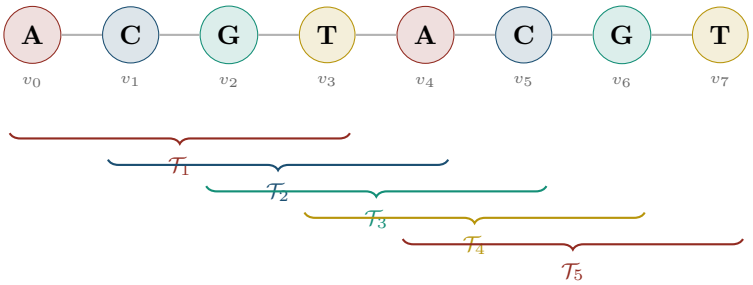


Figure 10.3: DNA sequence `ACGTACGT` mapped to a LinkedTetraChain. Each brace shows which four consecutive vertices form a tetrahedron. Note the overlap: each tetrahedron shares three vertices with its neighbors.

Notice the structure: each tetrahedron is a sliding window of width 4 that advances by one base at a time. The overlap of 3 bases ensures that *every pair of adjacent bases* is captured within at least one tetrahedron, and in fact every pair of bases separated by at most 2 positions is captured.

10.4.3 Why Not Triangles? An Information Density Argument

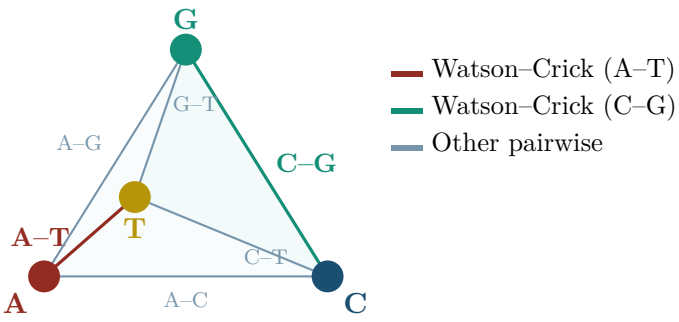
If we used triangles instead of tetrahedra, our sliding window would be width 3 with overlap 2. Each triangle would be a K_3 with only 3 edges. Let us compare the two approaches for the same 8-base sequence:

	Triangle mesh (K_3)	**Tetra mesh** (K_4)
Window width	3	4
Overlap	2	3
Elements for 8 bases	6 triangles	5 tetrahedra
Edges per element	3	6
Total unique edges	7	10
Pairwise interactions	3 per element	6 per element
Has volume?	No	Yes
Has chirality?	No	Yes

The tetrahedron captures 2× the pairwise interactions per element, while also encoding volume and chirality. For the same sequence, the tetrahedral representation is strictly richer.

10.4.4 The Six Edges: Pairwise Base Interactions

Within each tetrahedron $\mathcal{T}_k = (b_{k-1}, b_k, b_{k+1}, b_{k+2})$, the six edges encode all pairwise interactions among the four bases:



The six edges naturally partition into:

- **2 Watson–Crick edges:** A–T and C–G. These are the complementary base pairings that form hydrogen bonds in the double helix.
- **4 non-Watson–Crick edges:** A–C, A–G, C–T, G–T. These encode stacking interactions, non-canonical base pairing, and other local sequence features.

Each edge carries a learnable weight $w_{ij} \in \mathbb{R}$ that represents the strength and character of the pairwise interaction. These weights are analogous to attention scores in a transformer, but they are *local* (within a single tetrahedron) and *structured* (constrained by the tetrahedral geometry).

10.5 The TetraData Structure (TDS) — Formal Definition

We now give the complete formal definition of the TetraData Structure, incorporating everything we have built up.

Definition 10.3 (TetraData Structure)**.** A *TetraData Structure* (TDS) element is defined by:

$$\mathcal{T}_i = (\mathbf{V}_i, \mathbf{E}_i, \mathbf{B}_i, v_i, \chi_i), \tag{10.5}$$

where:

- $\mathbf{V}_i \in \mathbb{R}^{4\times 3}$: four vertex positions in 3D space.
- $\mathbf{E}_i \in \mathbb{R}^6$: six edge weights (forming the complete graph K_4).
- $\mathbf{B}_i \in \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}^4$: base type assignments for each vertex.
- $v_i \in \mathbb{R}$: enclosed volume (computed from vertex positions).
- $\chi_i \in \{+1, -1\}$: chirality (handedness), defined by the sign of the determinant.

10.5.1 Chirality

Chirality—the property of non-superimposability on a mirror image—is fundamental to molecular biology. Amino acids are left-handed (L-amino acids), sugars are right-handed (D-sugars), and the DNA double helix is right-handed. TDS preserves chirality through the sign of the vertex determinant:

$$\chi_i = \text{sign}\left(\det\begin{pmatrix}\mathbf{v}_1 - \mathbf{v}_0 & \mathbf{v}_2 - \mathbf{v}_0 & \mathbf{v}_3 - \mathbf{v}_0\end{pmatrix}\right). \tag{10.6}$$

Chirality-preserving transformations are those that maintain the sign of χ. This constraint ensures that biological structures retain their handedness through computational operations.

Example 10.2 (Computing Chirality)**.** Consider a tetrahedron with vertices:

$$\mathbf{v}_0 = (0,0,0), \quad \mathbf{v}_1 = (1,0,0), \quad \mathbf{v}_2 = (0,1,0), \quad \mathbf{v}_3 = (0,0,1).$$

The edge matrix is:

$$M = \begin{pmatrix}1-0 & 0-0 & 0-0\\ 0-0 & 1-0 & 0-0\\ 0-0 & 0-0 & 1-0\end{pmatrix} = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} = I_3.$$

We have $\det(M) = +1$, so $\chi = +1$ (right-handed). The signed volume is $V_{\text{signed}} = \frac{1}{6}(+1) = +\frac{1}{6}$.

Now reflect $\mathbf{v}_3$ across the xy-plane: $\mathbf{v}_3' = (0,0,-1)$. The new determinant is -1, giving $\chi = -1$ (left-handed). Reflection reverses chirality, as expected.

10.5.2 Face-Sharing Adjacency

Adjacent tetrahedra share triangular faces, forming *tetrahedral chains* or *tetrahedral meshes*. This face-sharing adjacency defines a natural graph structure:

Definition 10.4 (Tetrahedral Mesh)**.** A *tetrahedral mesh* is a collection of tetrahedra $\{\mathcal{T}_1, \ldots, \mathcal{T}_M\}$ where adjacent elements share triangular faces. The *dual graph* has one node per tetrahedron and edges between face-sharing neighbors.

Face-sharing adjacency is richer than the edge-sharing adjacency of triangle meshes: each tetrahedron has exactly four potential neighbors (one per face), creating a bounded-degree graph that enables efficient local computation.

10.6 36 Degrees of Freedom

Each TDS element carries a rich set of information. Let us count the degrees of freedom precisely.

10.6.1 Detailed DOF Breakdown

DOF	Component	Count	Description
1–12	Vertex positions $\mathbf{V}_i \in \mathbb{R}^{4\times 3}$	12	4 vertices $\times$ 3 coordinates each
13–18	Edge weights $\mathbf{E}_i \in \mathbb{R}^6$	6	All $\binom{4}{2} = 6$ pairwise interactions
19–34	Base encoding onehot$(\mathbf{B}_i) \in \mathbb{R}^{4\times 4}$	16	4 vertices $\times$ 4-dim one-hot encoding
35	Volume $v_i \in \mathbb{R}$	1	Signed volume of the tetrahedron
36	Chirality $\chi_i \in \{+1, -1\}$	1	Handedness (sign of the determinant)
	Total	**36**	

10.6.2 Comparison: Triangle vs. Tetrahedron DOF

Component	Triangle	Tetrahedron
Vertex positions	$3 \times 3 = 9$	$4 \times 3 = 12$
Edge weights	$\binom{3}{2} = 3$	$\binom{4}{2} = 6$
Base encoding	—	$4 \times 4 = 16$
Volume	0 (area only)	1
Chirality	0 (no handedness)	1
Total DOF	**12**	**36**
Ratio	$36/12 = 3\times$ richer	

If we count only the positional DOF (no base encoding), the triangle has 9 DOF and the tetrahedron has $12 + 6 + 1 + 1 = 20$ DOF—still more than 2$\times$ richer. The base encoding adds another 16 DOF that are entirely absent in triangle meshes.

10.6.3 TetraTokenization

For foundation model integration, each TDS element is serialized into a 36-dimensional vector:

$$\text{TetraToken}(\mathcal{T}_i) = [\underbrace{\mathbf{V}_i^{\text{flat}}}_{12} \mid \underbrace{\mathbf{E}_i}_{6} \mid \underbrace{\text{onehot}(\mathbf{B}_i)}_{16} \mid \underbrace{v_i}_{1} \mid \underbrace{\chi_i}_{1}] \in \mathbb{R}^{36}. \tag{10.7}$$

This tokenization enables DOGMA foundation models to ingest and produce 3D structural data alongside text, code, and genomic sequences.

TetraToken Design Study

In the proposed multimodal DOGMA design, the 36-dimensional TetraToken is projected to the model's hidden dimension d_{model} via a learned linear layer:

$$\mathbf{h}_i = \mathbf{W}_{\text{proj}} \cdot \text{TetraToken}(\mathcal{T}_i) + \mathbf{b}_{\text{proj}}, \quad \mathbf{W}_{\text{proj}} \in \mathbb{R}^{d_{\text{model}} \times 36}.$$

The proposed reverse direction uses a symmetric linear projection from d_{model} back to 36 dimensions, followed by extraction of vertex positions, edge weights, base assignments, volume, and chirality. This path is not part of the measured baseline in Chapter 14.

10.7 TetraMemory: Structured Memory for DOGMA

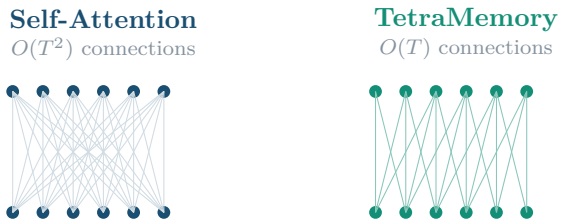
TetraMemory is DOGMA's alternative to self-attention. Instead of computing pairwise interactions between all positions (quadratic cost), TetraMemory organizes state elements into a tetrahedral mesh where interactions are *local but compose into global computation* through propagation.

10.7.1 The Quadratic Attention Problem

Standard self-attention computes:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}, \quad \text{cost: } O(T^2 d). \tag{10.8}$$

For a sequence of length $T = 100{,}000$ (a modest genomic sequence), the attention matrix has 10^{10} entries—impractical for both memory and computation. Even for $T = 10{,}000$, storing the $T \times T$ attention matrix in float32 requires ∼400 MB per head, per layer.



10.7.2 Face Propagation: Information Flow Through Shared Faces

The key mechanism of TetraMemory is *face propagation*: information flows from one tetrahedron to its neighbor through their shared triangular face.

In a LinkedTetraChain, $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$ share three vertices. When we compute the state of $\mathcal{T}_{k+1}$, three of its four vertices already carry information from $\mathcal{T}_k$. Only one vertex is new. This means that 75% of each tetrahedron's information comes from its predecessor—a natural form of *causal propagation*.

Definition 10.5 (TetraMemory State Update)**.** At layer ℓ, the state of tetrahedron $\mathcal{T}_i$ is updated by:

$$\mathbf{h}_i^{(\ell+1)} = \phi\left(\mathbf{h}_i^{(\ell)} + \sum_{j \in \mathcal{N}_4(i)} \alpha_{ij} \cdot \mathbf{W}^{(\ell)} \mathbf{h}_j^{(\ell)}\right), \tag{10.9}$$

where:

- $\mathcal{N}_4(i)$ is the set of (at most 4) face-sharing neighbors of $\mathcal{T}_i$.
- α_{ij} are attention weights computed from edge properties and shared-face geometry.
- $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d \times d}$ is the layer's learnable weight matrix.
- ϕ is a nonlinear activation function.

The cost per layer is $O(T \cdot k \cdot d)$ where $k \leq 4$, making it *linear* in sequence length.

10.7.3 Volume as Attention Weight

Each tetrahedron's signed volume provides a natural attention modulator:

$$\alpha_{ij} = \sigma\left(\frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d}} + \beta \cdot V_{\text{signed}}(\mathcal{T}_i)\right), \tag{10.10}$$

where σ is a sigmoid function and β is a learnable temperature parameter. The volume term acts as a *geometric bias*: tetrahedra with larger volume (more "open" configurations) attend more strongly to their neighbors, while degenerate (near-flat) tetrahedra attend weakly.

Volume as Importance

In molecular biology, the volume of a protein's active site determines its binding affinity. A large, open active site can accommodate many substrates; a narrow, constrained one is highly specific. TetraMemory's use of volume as an attention modulator mirrors this: the "openness" of a tetrahedral configuration controls how much information flows through it.

10.7.4 Edge Weights as Pairwise Compatibility

The six edge weights $\mathbf{E}_i \in \mathbb{R}^6$ within each tetrahedron encode pairwise compatibility scores between the four vertices. These are analogous to the pairwise attention scores in self-attention, but:

1. They are *local*: only the 6 pairs within a tetrahedron are computed, not the $\binom{T}{2}$ global pairs.
2. They are *geometrically constrained*: the edge weights must be consistent with a valid 3D tetrahedron (triangle inequality on faces, positive volume).
3. They are *typed*: the Watson–Crick edges (A–T, C–G) can carry different learned biases than the non-Watson–Crick edges.

10.7.5 Long-Range Information Flow

How does local interaction enable global computation? Through *propagation*: after L layers of TetraMemory updates, information from tetrahedron $\mathcal{T}_i$ has propagated to all tetrahedra within graph distance L in the dual graph.

For a LinkedTetraChain of n tetrahedra, the dual graph is a path graph of length n, so the diameter is n and full propagation requires $L = n$ layers. However, for a 3D tetrahedral mesh (not just a chain), the dual graph diameter is $O(n^{1/3})$, so $O(n^{1/3})$ layers suffice for global information flow.

Theorem 10.2 (TetraMemory Propagation)**.** For a tetrahedral mesh with n tetrahedra and dual-graph diameter D, after $L \geq D$ layers of TetraMemory updates (Equation 10.9), every tetrahedron has received information from every other tetrahedron.

Proof. At each layer, each tetrahedron receives information from its face-sharing neighbors (at most 4). After L layers, the receptive field of each tetrahedron is the set of all tetrahedra within graph distance L in the dual graph. When $L \geq D$, this includes all tetrahedra. □

10.7.6 Complexity Comparison

	Self-Attention	**TetraMemory**
Cost per layer	$O(T^2 d)$	$O(T \cdot k \cdot d)$, $k \leq 4$
Memory per layer	$O(T^2)$	$O(T \cdot k)$
Layers for global info	1 (instant)	$O(D)$ where D is mesh diameter
Total cost for global	$O(T^2 d)$	$O(D \cdot T \cdot k \cdot d)$
For 3D mesh ($D = O(T^{1/3})$)	$O(T^2 d)$	$O(T^{4/3} k d)$
Structural locality	None	Built-in (mesh topology)
Chirality encoding	None	Built-in (signed volume)

TetraMemory vs. Attention: The Biological Argument

Biological neural networks do not use all-to-all connectivity. Instead, neurons interact with local neighbors, and global computation emerges from multi-hop signal propagation through network topology. TetraMemory follows this principle: local interactions compose into global computation, just as local gene regulatory interactions compose into genome-wide expression patterns.

The key advantage is not just computational efficiency. It is *structural locality*: the mesh topology encodes which elements should interact most strongly, providing an architectural prior that dense attention lacks.

10.8 The DoubleHelixMesh

In real DNA, information is stored in two complementary strands running antiparallel to each other, forming the iconic double helix. DOGMA mirrors this with the **DoubleHelixMesh**: two complementary LinkedTetraChains—the *sense* strand and the *antisense* strand—cross-linked by Watson–Crick hydrogen bonds.

10.8.1 Sense and Antisense Strands

Given a sense-strand sequence $S = s_0 s_1 s_2 \cdots s_{n-1}$, the antisense strand is the reverse complement:

$$\bar{S} = \overline{s_{n-1}}\ \overline{s_{n-2}}\ \cdots\ \overline{s_1}\ \overline{s_0}, \tag{10.11}$$

where the complement map is $\overline{\mathsf{A}} = \mathsf{T}$, $\overline{\mathsf{T}} = \mathsf{A}$, $\overline{\mathsf{C}} = \mathsf{G}$, $\overline{\mathsf{G}} = \mathsf{C}$.

Each strand forms its own LinkedTetraChain:

- **Sense chain:** $\mathcal{C}_{\text{sense}} = (\mathcal{T}_1^+, \mathcal{T}_2^+, \ldots, \mathcal{T}_{n-3}^+)$

- **Antisense chain:** $\mathcal{C}_{\text{anti}} = (\mathcal{T}_1^-, \mathcal{T}_2^-, \ldots, \mathcal{T}_{n-3}^-)$

The antisense chain runs in the reverse direction (antiparallel), so $\mathcal{T}_k^-$ aligns spatially with $\mathcal{T}_{n-3-k}^+$.

10.8.2 Watson–Crick Cross-Links

The two strands are connected by hydrogen bonds between complementary base pairs:

- **A–T pairing:** 2 hydrogen bonds (weaker).
- **G–C pairing:** 3 hydrogen bonds (stronger).

In the DoubleHelixMesh, these cross-links are represented as inter-chain edges with weights proportional to hydrogen bond strength:

$$w_{\text{cross}}(b, \bar{b}) = \begin{cases} w_{\text{AT}} & \text{if } (b, \bar{b}) \in \{(\mathsf{A}, \mathsf{T}), (\mathsf{T}, \mathsf{A})\}, \\ w_{\text{GC}} & \text{if } (b, \bar{b}) \in \{(\mathsf{G}, \mathsf{C}), (\mathsf{C}, \mathsf{G})\}, \end{cases} \tag{10.12}$$

where $w_{\text{GC}} > w_{\text{AT}}$ reflects the stronger G–C bond.

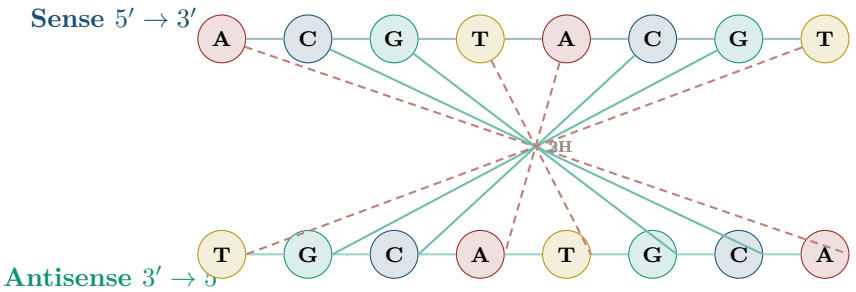


Figure 10.4: The DoubleHelixMesh for sequence `ACGTACGT`. The sense strand (top, $5' \to 3'$) and antisense strand (bottom, $3' \to 5'$) are connected by Watson–Crick hydrogen bonds. A–T pairs have 2 hydrogen bonds (dashed); G–C pairs have 3 hydrogen bonds (solid).

10.8.3 Advantages of the Double Helix Structure

The DoubleHelixMesh provides several computational advantages:

1. **Redundancy and error detection:** Each base on the sense strand has a complementary base on the antisense strand. Discrepancies can be detected and corrected, mirroring DNA's own error-correction mechanism.
2. **Bidirectional reading:** The sense strand reads $5' \to 3'$ while the antisense reads $3' \to 5'$. This provides a natural mechanism for bidirectional sequence processing without the need for separate forward and backward passes.
3. **Cross-strand attention:** The hydrogen bond cross-links create additional attention pathways between the two chains, enriching the information flow beyond what a single chain provides.

4. **Melting dynamics:** In biology, the double helix can "melt" (denature) at high temperatures, separating the strands. In DOGMA, a learned "temperature" parameter controls the strength of cross-strand interactions, enabling the model to dynamically decouple the strands when independent processing is beneficial.

DoubleHelixMesh Efficiency

Despite having two chains, the DoubleHelixMesh does *not* double the computational cost. The antisense chain shares vertex positions (by symmetry) with the sense chain, and the cross-strand hydrogen bond weights are computed from the base assignments (which are determined by complementarity). The additional cost is $O(T)$ for the cross-strand edges, which is dominated by the $O(T \cdot k \cdot d)$ per-layer cost of TetraMemory updates on each chain.

10.9 Delaunay Tetrahedralization

While the LinkedTetraChain is the primary structure for sequence data, DOGMA also supports general 3D point clouds through Delaunay tetrahedralization.

Given a set of points $P \subset \mathbb{R}^3$, the *Delaunay tetrahedralization* partitions the convex hull of P into tetrahedra such that no point lies inside the circumsphere of any tetrahedron.

Definition 10.6 (Delaunay Tetrahedralization)**.** A tetrahedralization of point set P is *Delaunay* if, for every tetrahedron $\mathcal{T}$, the open circumsphere of $\mathcal{T}$ contains no point of P.

The Delaunay condition maximizes the minimum solid angle among all tetrahedralizations, producing well-shaped elements that avoid degeneracies. This quality guarantee is important for numerical stability in TetraMemory computations.

Applications relevant to DOGMA:

- **Protein structure:** Given C_α coordinates, Delaunay tetrahedralization produces a volumetric mesh capturing local packing geometry.
- **Medical imaging:** CT/MRI voxels can be converted to Delaunay tetrahedral meshes for volumetric analysis.
- **Molecular dynamics:** Voronoi tessellation (dual of Delaunay) defines natural atomic neighborhoods.

10.10 Multimodal Applications of TDS

The TDS representation enables DOGMA to handle diverse data modalities through a unified geometric framework:

Modality	TDS Representation	Application
DNA/RNA	LinkedTetraChain	Sequence modeling, variant calling
3D Vision	Volumetric scene → tetra mesh	Object recognition, scene understanding
Proteins	C_α coordinates → Delaunay	Structure prediction, drug design
Medical	CT/MRI voxels → tetra mesh	Segmentation, anomaly detection
Game engines	Physics bodies → deformable tetra	Destruction simulation, soft bodies
Video	Temporal tetrahedra	Motion tracking, action recognition
Audio	Waveform helix encoding	Speaker recognition, music analysis

10.11 Worked Examples

Example 10.3 (Building a LinkedTetraChain from a DNA Sequence)**.** Given the sequence 5′-CGTACG-3′ (6 bases), construct the LinkedTetraChain.

Step 1: Assign vertices.

Vertex	v_0	v_1	v_2	v_3	v_4	v_5
Base	C	G	T	A	C	G

Step 2: Enumerate tetrahedra. The chain has $6 - 3 = 3$ tetrahedra:

$$\mathcal{T}_1 = (v_0, v_1, v_2, v_3) = (\mathsf{C}, \mathsf{G}, \mathsf{T}, \mathsf{A}),$$
$$\mathcal{T}_2 = (v_1, v_2, v_3, v_4) = (\mathsf{G}, \mathsf{T}, \mathsf{A}, \mathsf{C}),$$
$$\mathcal{T}_3 = (v_2, v_3, v_4, v_5) = (\mathsf{T}, \mathsf{A}, \mathsf{C}, \mathsf{G}).$$

Step 3: Count combinatorics.

- Vertices: 6
- Edges: $3(3) + 3 = 12$
- Shared faces: (v_1, v_2, v_3) between $\mathcal{T}_1$ and $\mathcal{T}_2$; (v_2, v_3, v_4) between $\mathcal{T}_2$ and $\mathcal{T}_3$.

Step 4: Compute volume and chirality for $\mathcal{T}_1$. Suppose we embed the vertices at: $\mathbf{v}_0 = (0, 0, 0)$, $\mathbf{v}_1 = (1, 0, 0)$, $\mathbf{v}_2 = (0.5, 0.87, 0)$, $\mathbf{v}_3 = (0.5, 0.29, 0.82)$.

Then:

$$V_{\text{signed}} = \frac{1}{6} \det \begin{pmatrix} 1 & 0.5 & 0.5 \\ 0 & 0.87 & 0.29 \\ 0 & 0 & 0.82 \end{pmatrix} = \frac{1}{6}(1 \cdot 0.87 \cdot 0.82) = \frac{0.7134}{6} \approx 0.119.$$

Since $V_{\text{signed}} > 0$, the chirality is $\chi = +1$ (right-handed).

Example 10.4 (TetraMemory vs. Self-Attention Cost)**.** Consider a genomic sequence of length $T = 50{,}000$ with embedding dimension $d = 512$.

Self-attention cost per layer:

$$O(T^2 d) = 50{,}000^2 \times 512 = 1.28 \times 10^{12} \text{ FLOPs.}$$

TetraMemory cost per layer (with $k = 4$ neighbors):

$$O(T \cdot k \cdot d) = 50{,}000 \times 4 \times 512 = 1.024 \times 10^8 \text{ FLOPs.}$$

Speedup per layer: $\frac{1.28 \times 10^{12}}{1.024 \times 10^8} \approx 12{,}500\times$.

Even accounting for the need for $O(T^{1/3}) \approx 37$ layers for global propagation in a 3D mesh, the total TetraMemory cost is:

$$37 \times 1.024 \times 10^8 = 3.79 \times 10^9 \text{ FLOPs,}$$

which is still $\sim$338$\times$ faster than a single self-attention layer.

10.12 Exercises

10.1. **[Fundamentals]** Compute the volume and chirality of the tetrahedron with vertices $(0, 0, 0)$, $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$. Verify that reflecting one vertex across a coordinate plane reverses the chirality.

10.2. **[Construction]** Build the LinkedTetraChain for the DNA sequence 5′-`GATTACA`-3′. List all tetrahedra, their base assignments, and the shared faces between consecutive tetrahedra. How many total edges does the chain have?

10.3. **[Analysis]** For a tetrahedral mesh with M elements, each with at most 4 face-sharing neighbors, compare the computational cost of one TetraMemory layer ($O(4Md)$) with one self-attention layer ($O(M^2 d)$). For what value of M does TetraMemory become 100$\times$ faster?

10.4. **[Information Density]** Compute the number of unique pairwise interactions captured by (a) a triangle mesh with sliding window of width 3 and overlap 2, and (b) a tetrahedral mesh with sliding window of width 4 and overlap 3, both applied to a sequence of length $T = 100$. Express the ratio.

10.5. **[DOF Counting]** A hexahedron (cube) has 8 vertices, 12 edges, and 6 faces. Compute the number of degrees of freedom for a "HexaData" structure analogous to TDS (vertex positions + edge weights + base encoding for 8 bases + volume + chirality). Is the DOF-per-vertex ratio better or worse than TDS?

10.6. **[Coding]** Implement the TetraTokenizer (Equation 10.7) in Python. Given a list of tetrahedra (each specified by 4 vertex coordinates and 4 base types), produce the 36-dimensional token representation for each.

10.7. **[DoubleHelix]** Given the sense strand 5′-`ATGCATGC`-3′, write out the antisense strand. Construct the DoubleHelixMesh and count the total number of cross-strand hydrogen bonds. How many are A–T bonds (2H) and how many are G–C bonds (3H)?

10.8. **[Theory]** Prove that the graph diameter of a Delaunay tetrahedralization of M uniformly distributed points in a unit cube is $O(M^{1/3})$.

10.9. **[Design]** Propose a scheme for encoding a protein backbone (sequence of C_α coordinates) as a TDS mesh. How would you assign base types $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ to vertices based on amino acid properties?

10.10. **[Research]** Compare TetraMemory with graph neural networks (GNNs). Both use local message passing. What does the tetrahedral structure provide that a generic graph does not? Consider: bounded degree, volume, chirality, and face-sharing.

10.13 Key Takeaways

Key Takeaways

- A tetrahedron is the *minimal solid*: four vertices, six edges, four faces, and one volume. It is the simplest 3D object with both volume and chirality. No simpler primitive can encode these properties.
- The TetraData Structure uses tetrahedra as computational primitives, mapping four vertices to DNA base types $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ and encoding 36 degrees of freedom per element.
- The LinkedTetraChain links tetrahedra by shared triangular faces, acting as a sliding window of width 4 with overlap 3 over a sequence. Each new vertex adds exactly 3 edges and 1 tetrahedron.
- DNA sequences map naturally to LinkedTetraChains: each tetrahedron represents a window of 4 consecutive bases, and adjacent tetrahedra share 3 bases through their common face.
- TetraMemory replaces $O(T^2)$ self-attention with $O(T \cdot k)$ local tetrahedral interactions ($k \leq 4$) that compose into global computation through face propagation. Volume serves as a geometric attention weight; edge weights encode local pairwise compatibility.
- The DoubleHelixMesh pairs a sense and antisense LinkedTetraChain with Watson–Crick cross-links (A–T: 2 hydrogen bonds, G–C: 3 hydrogen bonds), providing redundancy, bidirectional reading, and cross-strand attention.
- Delaunay tetrahedralization extends TDS beyond sequences to general 3D point clouds, enabling multimodal applications from protein structure to medical imaging.

Suggested Reading

- Chapter 7, Section on DNAComputeBlock: TetraMemory's architectural context.
- Jumper et al. [2021]: AlphaFold's use of structural representations for proteins.
- Gu and Dao [2023]: Alternative approach to sub-quadratic sequence processing.

Chapter 11

CodonForge — The DNA Compiler and SDANS

A compiler is a bridge between the language of intent and the language of execution. In DOGMA, that bridge is biological.

— DOGMA Design Manifesto

Every sophisticated computational system needs a compiler: a principled mechanism for translating high-level specifications into executable low-level operations. In conventional computing, compilers translate C into assembly, or Python into bytecode. In DOGMA, **CodonForge** translates prompt bundles and genomic specifications into executable codon programs—using the same triplet encoding that biology uses to translate mRNA into protein.

This chapter presents CodonForge as a *symbolic DNA programming layer* that treats codons as opcodes and genes as execution blocks. We begin with the biological foundations that inspired the design, then build up the full compilation pipeline piece by piece. We conclude with **SDANS** (Strand Displacement Attention Neural System or Stochastic DNA-Algebraic Neural Simulation), a compilation target that formally bridges CodonForge programs and neural network execution.

Learning objectives. After studying this chapter, students should be able to:

- Explain how biological codon translation (mRNA → ribosome → amino acid) inspired CodonForge's design.
- List and categorize all 64 CodonForge opcodes across the five opcode families.
- Trace a prompt through the full Parse → Compile → Optimize pipeline.
- Construct forward and reverse DNA strands for a CodonForge program.
- Encode and decode text using the Word2FASTA bridge.
- Describe how SDANS maps symbolic codon programs to differentiable neural architectures.
- Compare the CodonForge compilation pipeline with traditional compilers such as LLVM.

11.1 Biological Foundations: Codon Translation in Nature

Before we can understand CodonForge, we must first understand the biological process that inspired it. In every living cell, the information stored in DNA is translated into functional proteins through a remarkably elegant pipeline. This section reviews that pipeline in detail, because CodonForge directly mirrors each of its stages.

11.1.1 The Central Dogma of Molecular Biology

The *central dogma* describes the flow of genetic information:

$$\text{DNA} \xrightarrow{\text{Transcription}} \text{mRNA} \xrightarrow{\text{Translation}} \text{Protein} \tag{11.1}$$

1. **Transcription.** The enzyme RNA polymerase reads one strand of the DNA double helix (the *template strand*) and synthesizes a complementary messenger RNA (mRNA) molecule. The mRNA is a portable copy of the gene's instructions.

2. **Translation.** The ribosome—a large molecular machine—reads the mRNA three nucleotides at a time. Each triplet of nucleotides is called a *codon*. Transfer RNA (tRNA) molecules carry individual amino acids to the ribosome, where they are matched to the appropriate codon via anticodon base-pairing.

3. **Protein folding.** The resulting chain of amino acids folds into a three-dimensional protein that performs a specific function in the cell.

Why Triplets?

Why does biology use three-letter codons instead of two or four? With four possible nucleotides (A, U, G, C), a two-letter code would provide only $4^2 = 16$ combinations—not enough to encode 20 amino acids plus stop signals. A three-letter code provides $4^3 = 64$ combinations, which is more than sufficient and provides redundancy (multiple codons can encode the same amino acid). A four-letter code ($4^4 = 256$) would be wasteful. Three is the minimum length that encodes sufficient information, a principle that CodonForge inherits directly.

11.1.2 Step-by-Step: From mRNA to Amino Acid

Let us walk through translation in detail, as each step has a computational analogue in CodonForge.

Step 1: Initiation. The ribosome assembles on the mRNA at the *start codon* `AUG`, which encodes the amino acid methionine. This is analogous to a program's entry point or `main()` function. In CodonForge, this corresponds to the *promoter codon* that marks the beginning of a gene.

Step 2: Elongation. The ribosome moves along the mRNA, reading one codon at a time. For each codon, a tRNA molecule with the complementary anticodon delivers the correct amino acid. The ribosome catalyzes a peptide bond between the new amino acid and the growing chain. In CodonForge, this corresponds to sequential opcode execution: the "ribosome" is the interpreter, and each codon triggers a specific computational operation.

Step 3: Termination. When the ribosome encounters a *stop codon* (`UAA`, `UAG`, or `UGA`), translation halts. No tRNA matches these codons; instead, release factors cause the ribosome to release the completed protein. In CodonForge, the terminator codon halts gene execution and triggers output emission.

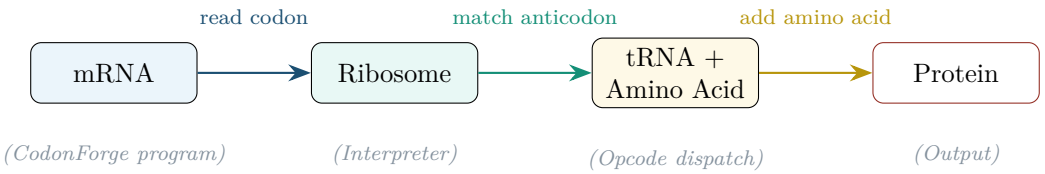


Figure 11.1: Biological translation and its CodonForge analogy. The ribosome reads mRNA codons sequentially, just as CodonForge's interpreter reads codon opcodes.

11.1.3 The Complete Biological Codon Table

The following table shows all 64 codons of the standard genetic code, organized by the first and second nucleotide positions. Understanding this table is essential because CodonForge's opcode table is structured as a direct computational analogue.

Table 11.1: The standard genetic code: all 64 RNA codons and their corresponding amino acids. Start codon (`AUG`) shown in bold; stop codons shown with **Stop**.

		Second Position				
		U	**C**	**A**	**G**	
U	U	Phe (UUU)	Ser (UCU)	Tyr (UAU)	Cys (UGU)	U
	U	Phe (UUC)	Ser (UCC)	Tyr (UAC)	Cys (UGC)	C
	U	Leu (UUA)	Ser (UCA)	**Stop** (UAA)	**Stop** (UGA)	A
	U	Leu (UUG)	Ser (UCG)	**Stop** (UAG)	Trp (UGG)	G
C	C	Leu (CUU)	Pro (CCU)	His (CAU)	Arg (CGU)	U
	C	Leu (CUC)	Pro (CCC)	His (CAC)	Arg (CGC)	C
	C	Leu (CUA)	Pro (CCA)	Gln (CAA)	Arg (CGA)	A
	C	Leu (CUG)	Pro (CCG)	Gln (CAG)	Arg (CGG)	G
A	A	Ile (AUU)	Thr (ACU)	Asn (AAU)	Ser (AGU)	U
	A	Ile (AUC)	Thr (ACC)	Asn (AAC)	Ser (AGC)	C
	A	Ile (AUA)	Thr (ACA)	Lys (AAA)	Arg (AGA)	A
	A	**Met** (AUG)	Thr (ACG)	Lys (AAG)	Arg (AGG)	G
G	G	Val (GUU)	Ala (GCU)	Asp (GAU)	Gly (GGU)	U
	G	Val (GUC)	Ala (GCC)	Asp (GAC)	Gly (GGC)	C
	G	Val (GUA)	Ala (GCA)	Glu (GAA)	Gly (GGA)	A
	G	Val (GUG)	Ala (GCG)	Glu (GAG)	Gly (GGG)	G

Degeneracy Is Not Redundancy

Notice that 61 codons encode only 20 amino acids. This means most amino acids have multiple codons—a property called *degeneracy*. Critically, degeneracy is not useless redundancy. Different codons for the same amino acid are translated at different speeds, affecting protein folding and expression levels. This is called *codon usage bias*, and organisms optimize their codon choices for translational efficiency. CodonForge inherits this principle: multiple codons map to the same operation but with different optimization hints.

11.2 DNA as a Programming Language

The genetic code maps 64 codons (three-nucleotide sequences) to 20 amino acids plus stop signals. CodonForge extends this idea: 64 codons are mapped to *computational opcodes*, organizing computation into gene-like blocks with promoter/terminator control flow.

Definition 11.1 (Codon Opcode)**.** A *codon opcode* is a mapping from a three-letter DNA sequence to a computational operation:

$$\text{opcode} : \Sigma_{\text{DNA}}^3 \to \mathcal{O}, \tag{11.2}$$

where $\Sigma_{\text{DNA}} = \{\texttt{A}, \texttt{T}, \texttt{G}, \texttt{C}\}$ is the DNA alphabet and $\mathcal{O}$ is the set of available operations. Since $|\Sigma_{\text{DNA}}^3| = 4^3 = 64$, CodonForge defines exactly 64 opcodes organized into functional families.

DNA vs. RNA Alphabets

Biology uses two slightly different alphabets: DNA uses {A, T, G, C} while RNA uses {A, U, G, C}, where uracil (U) replaces thymine (T). CodonForge uses the DNA alphabet because its programs are "stored" like DNA and "executed" like mRNA. When we refer to biological codons (e.g., the start codon `AUG`), we use the RNA alphabet. When we write CodonForge codons (e.g., `ATG`), we use the DNA alphabet. The mapping is straightforward: replace every T with U to go from DNA to RNA, and every U with T for the reverse.

11.2.1 Opcode Families

CodonForge organizes its 64 opcodes into five families, mirroring the functional categories of biological codons. The families are:

Family	Count	Example Ops	Biological Analogue
Sense	12	`LOAD_CONTEXT`, `MATCH_KEY`, `SCAN_MOTIF`	Sensor proteins
Binding	12	`BIND_VALUE`, `STORE_REG`, `ASSOCIATE`	Molecular binding
Regulation	14	`IF_ACTIVE`, `GATE_CHECK`, `THRESHOLD`	Transcription factors
Expression	14	`EMIT_JSON`, `EMIT_TEXT`, `FOLD_STRUCT`	Protein expression
Termination	12	`STOP`, `CHECKPOINT`, `CLEANUP`	Stop codons

Let us examine each family in detail.

Sense Family (12 opcodes)

The Sense family handles all input perception and context loading. Just as sensor proteins in a cell detect environmental signals, Sense opcodes detect and load relevant information from the input context.

Codon	Opcode	Description
AAA	LOAD_CONTEXT	Load the full input context (query + environment) into register R0
AAT	LOAD_QUERY	Load only the user query portion into R0
AAG	LOAD_SYSTEM	Load system-level instructions into R0
AAC	LOAD_HISTORY	Load conversation history into R0
ATA	SCAN_MOTIF	Scan current context for recurring motifs; store matches in R1
ATT	SCAN_ENTITY	Extract named entities from context; store in R1
ATG	MATCH_KEY	Find passages matching key terms; store in R2
ATC	MATCH_PATTERN	Match regex-like patterns in context; store in R2
AGA	SENSE_TONE	Detect emotional tone of input; store classification in R3
AGT	SENSE_INTENT	Classify user intent (question, command, statement); store in R3
AGG	SENSE_LANG	Detect input language; store language code in R3
AGC	SENSE_DOMAIN	Classify input domain (science, law, casual); store in R3

Binding Family (12 opcodes)

The Binding family handles data association and storage. Just as molecular binding in biology involves specific lock-and-key interactions between molecules, Binding opcodes create structured associations between data elements.

Codon	Opcode	Description
TAA	BIND_VALUE	Bind a retrieved value to a template slot; R2 → R4
TAT	BIND_PAIR	Create a key-value pair from R1 and R2; store in R4
TAG	BIND_LIST	Aggregate multiple values into an ordered list in R4
TAC	BIND_STRUCT	Bind values into a structured record in R4
TTA	STORE_REG	Store current accumulator value to specified register
TTT	STORE_MEM	Store value to named memory location
TTG	LOAD_REG	Load value from specified register to accumulator
TTC	LOAD_MEM	Load value from named memory location
TGA	ASSOCIATE	Create a weighted association between two registers
TGT	CROSS_REF	Cross-reference two data structures for shared elements
TGG	MERGE	Merge two registers into one combined structure
TGC	COPY	Copy value from one register to another

Regulation Family (14 opcodes)

The Regulation family implements control flow, conditional execution, and gating mechanisms. These opcodes mirror the role of transcription factors in biology—proteins that determine whether a gene is activated or silenced.

Codon	Opcode	Description
GAA	IF_ACTIVE	Conditional: execute next opcodes only if register is non-empty
GAT	IF_MATCH	Conditional: execute if pattern match succeeded
GAG	IF_ABOVE	Conditional: execute if register value exceeds threshold
GAC	IF_BELOW	Conditional: execute if register value is below threshold
GTA	GATE_CHECK	Gate: pass data only if quality score exceeds minimum
GTT	GATE_TOXICITY	Gate: block output if toxicity score exceeds limit
GTG	GATE_RELEVANCE	Gate: block if content relevance score is too low
GTC	GATE_LENGTH	Gate: enforce output length constraints
GGA	THRESHOLD	Set a numeric threshold value for subsequent comparisons
GGT	BRANCH	Unconditional jump to specified gene
GGG	LOOP_START	Begin a loop block (with max iteration counter)
GGC	LOOP_END	End a loop block and check continuation condition
GCA	MUTEX_LOCK	Acquire exclusive access to a shared register
GCT	MUTEX_UNLOCK	Release exclusive access to a shared register

Expression Family (14 opcodes)

The Expression family handles output generation, formatting, and structural assembly. In biology, "expression" refers to the process by which a gene's information is synthesized into a functional product (usually a protein). In CodonForge, Expression opcodes synthesize the computational output.

Codon	Opcode	Description
CAA	EMIT_TEXT	Emit plain text output from register contents
CAT	EMIT_JSON	Emit structured JSON output from register contents
CAG	EMIT_CODE	Emit source code with syntax formatting
CAC	EMIT_TABLE	Emit tabular data output
CTA	FOLD_STRUCT	Fold flat data into a hierarchical structure
CTT	FOLD_SUMMARY	Compress data into a summary representation
CTG	FOLD_TREE	Organize data into a tree structure
CTC	FOLD_GRAPH	Organize data into a graph structure
CGA	FORMAT_MD	Apply Markdown formatting to output
CGT	FORMAT_HTML	Apply HTML formatting to output
CGG	FORMAT_LATEX	Apply LaTeX formatting to output
CGC	FORMAT_PLAIN	Strip all formatting; emit raw text
CCA	CONCAT	Concatenate multiple register values into output stream
CCT	TEMPLATE	Fill a template string with register values

Termination Family (12 opcodes)

The Termination family handles program termination, checkpointing, cleanup, and error handling. These opcodes are analogous to the three biological stop codons (`UAA`, `UAG`, `UGA`), but CodonForge's richer set enables graceful shutdown, state persistence, and error recovery.

Codon	Opcode	Description
CCC	STOP	Halt execution immediately; return current output
CCG	STOP_SUCCESS	Halt with success status code
GCG	STOP_ERROR	Halt with error status code and error message
GCC	CHECKPOINT	Save full execution state for later resumption or lineage
ATG	CHECKPOINT_PARTIAL	Save partial state (registers only, no context)
ACT	CLEANUP	Deallocate temporary registers and memory
ACC	CLEANUP_ALL	Deallocate all non-output registers
ACA	LOG_STATE	Write current state to execution log
ACG	LOG_ERROR	Write error information to execution log
TCA	RETRY	Re-execute current gene from its promoter
TCT	FALLBACK	Switch to fallback gene on error
TCC	YIELD	Pause execution; yield partial output; resume later

Shared Codons

Observant readers will notice that ATG appears in both the Sense family (MATCH_KEY) and the Termination family (CHECKPOINT_PARTIAL). This is intentional: just as biological AUG serves as both the start codon and the codon for methionine, CodonForge allows context-dependent interpretation. When ATG appears as the first codon after a promoter, it functions as a Sense opcode. When it appears later in a gene, context determines its family membership. This is resolved at compile time by the contextual disambiguation pass.

Degeneracy and Optimization

Just as the biological genetic code is *degenerate* (multiple codons encoding the same amino acid), CodonForge supports *opcode aliasing*: multiple codon patterns can invoke the same operation with different optimization hints. For example, several codons might all map to LOAD_CONTEXT, but with different implicit priorities for cache behavior or precision level. This degeneracy provides a mechanism for codon optimization analogous to biological codon usage bias.

11.3 Gene Specifications

In CodonForge, a *gene* is an ordered block of codon opcodes bracketed by a promoter and a terminator. This mirrors the biological concept of a gene: a segment of DNA that encodes a functional product, flanked by regulatory sequences.

Definition 11.2 (CodonForge Gene)**.** A *gene* is a triple $G = (P, \mathbf{c}, T)$ where:

- P is a promoter codon (determines activation conditions).
- $\mathbf{c} = (c_1, c_2, \ldots, c_m)$ is an ordered sequence of operational codons.
- T is a terminator codon (signals end of gene execution).

A *genome program* is an ordered list of genes: $\Pi = (G_1, G_2, \ldots, G_K)$. Execution proceeds gene by gene; within each gene, codons execute sequentially. Promoter conditions determine whether a gene fires in the current context—implementing the regulatory control described in Chapter 3.

11.3.1 Gene Structure in Detail

Let us examine each component of a gene more closely.

Promoters. A promoter codon does not perform computation itself; instead, it defines the *conditions under which the gene is activated*. Think of it as an `if` statement that guards an entire function. Promoters can specify:

- **Always-on:** The gene always executes (unconditional promoter).
- **Register-conditional:** The gene executes only if a specified register is non-empty or exceeds a threshold.
- **Context-conditional:** The gene executes only when the input matches a specific pattern (e.g., a question-type input, a code-generation request).
- **Priority-conditional:** The gene executes only if no higher-priority gene has already handled the input.

Opcode body. The body of a gene is a sequence of codon opcodes executed in order. Opcodes may read from and write to registers, consume input from the context, and produce intermediate or final output. The body implements the gene's computational function.

Terminators. A terminator codon marks the end of the gene. It can trigger cleanup of temporary data, emit output, checkpoint execution state for lineage tracking, or signal error conditions.

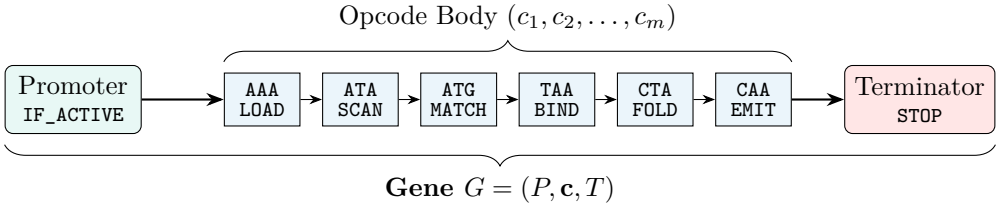


Figure 11.2: Structure of a CodonForge gene. The promoter gates execution, the opcode body performs computation, and the terminator signals completion.

11.3.2 Forward and Reverse Strands

Following DNA's antiparallel structure, every CodonForge program automatically generates a *reverse strand*: the reverse complement of the forward program. The reverse strand provides:

- **Redundancy:** Information is encoded twice, enabling error detection.
- **Bidirectional reading:** Some operations can read the reverse strand for complementary computation.
- **HelixHash compatibility:** The double-strand representation feeds directly into HelixHash's structural analysis (Chapter 8).

Constructing the Reverse Complement

To construct the reverse complement, we apply two operations:

1. **Complement:** Replace each nucleotide with its Watson-Crick complement: A↔T and G↔C.
2. **Reverse:** Read the complemented sequence from right to left.

Example 11.1 (Forward and Reverse Strand Construction)**.** Consider a simple gene with the forward strand:

```
5'-ATG AAA ATA TAA CAA CCC-3'
```

This encodes: `MATCH_KEY` → `LOAD_CONTEXT` → `SCAN_MOTIF` → `BIND_VALUE` → `EMIT_TEXT` → `STOP`.

Step 1: Complement each base:

```
TAC TTT TAT ATT GTT GGG
```

Step 2: Reverse the entire sequence:

```
3'-GGG TTG TTA TAT TTT CAT-5'
```

The reverse complement strand reads (in the 5'→3' direction):

```
5'-GGG TTG TTA TAT TTT TAC-3'
```

The double-stranded representation is:

```
  5'-ATG AAA ATA TAA CAA CCC-3'
| | |  | | |  | | |  | | |  | | |  | | |
  3'-TAC TTT TAT ATT GTT GGG-5'
```

11.4 The Compilation Pipeline

CodonForge implements a three-stage compilation pipeline that transforms human-readable prompt bundles into executable codon programs:

$$\text{PromptBundle} \xrightarrow{\text{Parse}} \text{Genomic IR} \xrightarrow{\text{Compile}} \text{Codon Program} \xrightarrow{\text{Optimize}} \text{Executable} \tag{11.3}$$

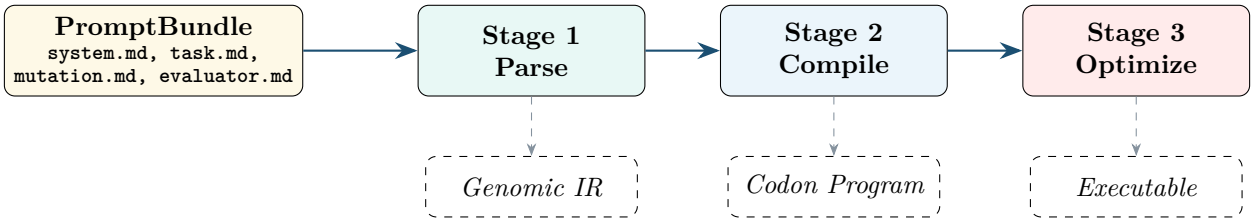


Figure 11.3: The CodonForge three-stage compilation pipeline. Each stage produces an intermediate representation that feeds the next stage.

11.4.1 Stage 1: Parse

The input PromptBundle (consisting of `system.md`, `task.md`, `mutation.md`, `evaluator.md`) is parsed into a *Genomic Intermediate Representation* (Genomic IR):

- Text fragments are segmented into *functional units*—the smallest independently meaningful pieces of instruction or data.
- Each unit is typed: instruction, constraint, template, or evaluation criterion.
- Dependencies between units are identified (e.g., "summarize the article" depends on "load the article").
- Region boundaries are established, grouping related units into candidate genes.

The Genomic IR is a directed acyclic graph (DAG) where nodes represent functional units and edges represent dependencies. This DAG is the foundation for all subsequent compilation.

Genomic Intermediate Representation

Formally, the Genomic IR is a tuple $\mathcal{G} = (V, E, \tau, \delta)$ where:

- V is a set of functional unit nodes.
- $E \subseteq V \times V$ is a set of dependency edges.
- $\tau : V \to \{\text{instruction}, \text{constraint}, \text{template}, \text{criterion}\}$ is a type function.
- $\delta : V \to \mathbb{N}$ assigns a priority/depth to each node.

11.4.2 Stage 2: Compile

The Genomic IR is compiled into a codon program by:

1. **Gene formation:** Each functional unit (or group of related units) is mapped to a gene (promoter + opcode sequence + terminator). Instruction units become Sense and Binding opcode sequences. Constraint units become Regulation opcodes. Template and criterion units become Expression opcodes.
2. **Gene ordering:** Genes are ordered according to execution dependencies derived from the IR's DAG structure. A topological sort ensures that each gene's inputs are available before it executes.
3. **Regulatory insertion:** Conditional execution is implemented by inserting Regulation opcodes (e.g., `IF_ACTIVE`, `GATE_CHECK`) as promoters or internal guards.
4. **Reverse strand generation:** The reverse complement strand is generated for each gene and for the complete program.

11.4.3 Stage 3: Optimize

The raw codon program is optimized through passes analogous to compiler optimization:

- **Dead code elimination:** Remove genes that can never be activated (unreachable promoter conditions).
- **Codon optimization:** Replace codons with aliases that have better execution properties (analogous to biological codon usage optimization). For example, choosing a codon variant that favors cache-friendly memory access patterns.
- **Regulatory simplification:** Merge redundant regulatory gates. If two consecutive `IF_ACTIVE` checks test the same register, the second is redundant.
- **GC content balancing:** Ensure the program has balanced base composition for HelixHash stability. An extreme GC imbalance can destabilize the double-stranded representation.
- **Gene fusion:** Merge small adjacent genes with compatible promoters into a single gene to reduce promoter evaluation overhead.
- **Register allocation:** Minimize register usage by reusing registers whose values are no longer needed.

11.5 Worked Example: Compiling "What is DNA?"

To make the compilation pipeline concrete, let us trace a simple prompt through all three stages. Suppose a user submits the prompt:

`What is DNA?`

with a minimal system prompt: `"You are a helpful biology tutor. Answer clearly and concisely."`

11.5.1 Stage 1: Parse — Building the Genomic IR

The parser segments the input into functional units:

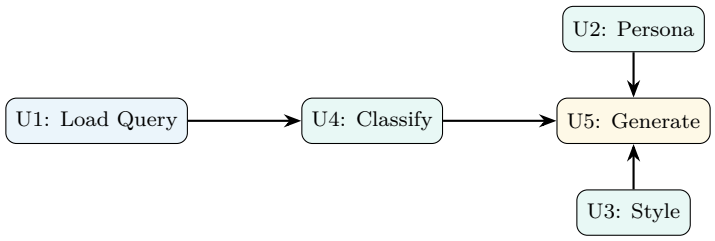
```
# Functional units extracted from PromptBundle:
unit_1 = {
    type: "instruction",
    content: "load_user_query",
    source: "task.md",
    data: "What is DNA?"
}
unit_2 = {
    type: "constraint",
    content: "persona_constraint",
    source: "system.md",
    data: "helpful biology tutor"
}
unit_3 = {
    type: "constraint",
```

```
    content: "style_constraint",
    source: "system.md",
    data: "clear and concise"
}
unit_4 = {
    type: "instruction",
    content: "classify_intent",
    source: "inferred",
    data: "question -> factual_answer"
}
unit_5 = {
    type: "template",
    content: "generate_answer",
    source: "inferred",
    data: "produce text output"
}

# Dependency edges:
# unit_1 -> unit_4 (must load query before classifying intent)
# unit_4 -> unit_5 (must know intent before generating answer)
# unit_2 -> unit_5 (persona constrains generation)
# unit_3 -> unit_5 (style constrains generation)
```

Listing 11.1: Genomic IR after parsing (pseudocode representation).

The resulting DAG has five nodes with the dependency structure:



11.5.2 Stage 2: Compile — Generating the Codon Program

The compiler maps each functional unit group to genes:

```
# Gene 1: Context Sensing (promoter: always active)
# Maps from: unit_1, unit_4
PROMOTER: ALWAYS_ON
  AAA  LOAD_CONTEXT       # Load "What is DNA?" into R0
  AGT  SENSE_INTENT       # Classify as question -> R3
  ATA  SCAN_MOTIF          # Extract key term "DNA" -> R1
  ATG  MATCH_KEY           # Search knowledge for "DNA" -> R2
TERMINATOR: GCC CHECKPOINT  # Save state

# Gene 2: Constraint Application (promoter: IF R2 nonempty)
# Maps from: unit_2, unit_3
PROMOTER: GAA IF_ACTIVE(R2)
  GTA  GATE_CHECK          # Verify retrieved content quality
  GTG  GATE_RELEVANCE      # Ensure "DNA" content is relevant
  GGA  THRESHOLD(style=concise)  # Set conciseness threshold
```

```
  GTC  GATE_LENGTH          # Enforce length < 200 words
TERMINATOR: ACT CLEANUP     # Clean temp registers

# Gene 3: Answer Generation (promoter: IF R2 nonempty)
# Maps from: unit_5
PROMOTER: GAA IF_ACTIVE(R2)
  TAA  BIND_VALUE           # Bind DNA knowledge to template -> R4
  CTA  FOLD_STRUCT          # Structure answer logically -> R5
  CAA  EMIT_TEXT            # Produce text output from R5
TERMINATOR: CCC STOP        # Halt with success
```

Listing 11.2: Codon program for "What is DNA?" (pseudocode).

The full DNA sequence for this program (forward strand) is:

```
5'-AAA AGT ATA ATG GCC | GAA GTA GTG GGA GTC ACT | GAA TAA CTA CAA CCC-3'
```

where | marks gene boundaries.

11.5.3 Stage 3: Optimize — Refining the Program

The optimizer applies several passes to the raw program:

1. **Dead code elimination:** All genes have reachable promoters—no elimination needed.
2. **Codon optimization:** The codon `AAA` (`LOAD_CONTEXT`) could be replaced with `AAT` (`LOAD_QUERY`) since we only need the user query, not the full context. This is more efficient.
3. **Regulatory simplification:** Genes 2 and 3 have the same promoter condition (`IF_ACTIVE(R2)`). They can share a single promoter evaluation by fusing into one gene.
4. **GC content check:** The current GC content is $(G + C)/\text{total} = 19/48 \approx 39.6\%$, which is within the acceptable range of 35–65%. No rebalancing needed.

After optimization, the program becomes more compact:

```
# Gene 1: Sense (always on)
PROMOTER: ALWAYS_ON
  AAT  LOAD_QUERY           # Load only the query (optimized)
  AGT  SENSE_INTENT         # Classify intent
  ATA  SCAN_MOTIF            # Extract "DNA"
  ATG  MATCH_KEY             # Retrieve knowledge
TERMINATOR: GCC CHECKPOINT

# Gene 2: Constrain + Generate (fused, IF R2 nonempty)
PROMOTER: GAA IF_ACTIVE(R2)
  GTA  GATE_CHECK            # Quality gate
  GTG  GATE_RELEVANCE        # Relevance gate
  GTC  GATE_LENGTH           # Length gate
  TAA  BIND_VALUE            # Bind answer
  CTA  FOLD_STRUCT           # Structure
  CAA  EMIT_TEXT             # Output
TERMINATOR: CCC STOP
```

Listing 11.3: Optimized codon program for "What is DNA?".

The optimized program has 2 genes (down from 3) and 12 opcodes (down from 14), a 14% reduction in program size.

11.6 Execution Semantics

CodonForge programs execute over a *computational register file*: a set of named registers that hold intermediate values during execution.

Definition 11.3 (CodonForge Execution State)**.** The execution state is a tuple $S = (\mathbf{R}, \text{PC}, \text{Context})$ where:

- $\mathbf{R} = \{R_0, R_1, \ldots, R_n\}$ is the register file: a dictionary mapping register names to values (strings, numbers, structured data, or $\perp$ for empty).
- $\text{PC} = (g, i)$ is the program counter: a pair of current gene index g and current codon index i within that gene.
- Context is the input context (query, task description, environment state).

State transitions are deterministic: given state S_t and codon c_t, the next state $S_{t+1} = \delta(S_t, c_t)$ is uniquely determined.

The transition function δ is defined per opcode family:

$$\delta_{\text{Sense}}(S, c) = S[\mathbf{R}[r_{\text{out}}] \mapsto \text{perceive}(\text{Context}, c.\text{params})] \tag{11.4}$$

$$\delta_{\text{Binding}}(S, c) = S[\mathbf{R}[r_{\text{out}}] \mapsto \text{bind}(\mathbf{R}[r_{\text{in}}], c.\text{params})] \tag{11.5}$$

$$\delta_{\text{Regulation}}(S, c) = \begin{cases} S & \text{if eval}(\mathbf{R}, c.\text{condition}) = \text{true} \\ S[\text{PC} \mapsto \text{skip}] & \text{otherwise} \end{cases} \tag{11.6}$$

$$\delta_{\text{Expression}}(S, c) = S[\text{output} \mathrel{+}= \text{emit}(\mathbf{R}[r_{\text{in}}], c.\text{format})] \tag{11.7}$$

$$\delta_{\text{Termination}}(S, c) = S[\text{status} \mapsto c.\text{halt\_type}] \tag{11.8}$$

11.6.1 Example Execution Trace

Consider running our optimized "What is DNA?" program. We trace the register file through each opcode:

Step	Opcode	Register Changed	Value
1	`LOAD_QUERY`	R_0	`"What is DNA?"`
2	`SENSE_INTENT`	R_3	`intent=factual_question`
3	`SCAN_MOTIF`	R_1	`[motif:"DNA", type:acronym]`
4	`MATCH_KEY`	R_2	`[knowledge:"DNA stands for deoxyribonucleic..."]`
5	`CHECKPOINT`	*(log)*	State saved for lineage
Gene 2 promoter: `IF_ACTIVE(R2)` → true (R2 is nonempty)			
6	`GATE_CHECK`	—	Quality score 0.92 > 0.5: pass
7	`GATE_RELEVANCE`	—	Relevance 0.97 > 0.7: pass
8	`GATE_LENGTH`	—	Est. length 85 words < 200: pass
9	`BIND_VALUE`	R_4	Knowledge bound to answer template
10	`FOLD_STRUCT`	R_5	Structured: intro → definition → significance
11	`EMIT_TEXT`	*(output)*	"DNA, or deoxyribonucleic acid, is..."
12	`STOP`	*(halt)*	Execution complete with success

11.7 The Word2FASTA Bridge

CodonForge includes a *reversible translation layer* between text and DNA-encoded representations. This layer, called **Word2FASTA**, allows arbitrary text (prompts, responses, metadata) to be encoded as DNA sequences in FASTA format—the standard file format used in bioinformatics.

11.7.1 Why Encode Text as DNA?

At first glance, encoding text as DNA might seem like an unnecessary complication. But the Word2FASTA bridge enables several powerful capabilities:

- **Unified representation:** Both programs (codon opcodes) and data (text) are represented in the same DNA alphabet, enabling uniform processing by HelixHash and other DOGMA components.
- **Bioinformatics tooling:** Encoded prompts can be analyzed using standard bioinformatics tools: sequence alignment (BLAST), motif discovery, phylogenetic analysis of prompt evolution, and GC content analysis.
- **Genomic storage:** Prompt genomes can be stored in standard FASTA databases and version-controlled alongside program code.
- **Mutation compatibility:** Text encoded as DNA can undergo biological-style mutations (point mutation, insertion, deletion, crossover) during the evolutionary optimization described in Chapter 3.

11.7.2 The Encoding Algorithm

The encoding maps each ASCII character to a pair of codons (6 nucleotides), giving $4^6 = 4096$ possible values—far more than the 128 standard ASCII characters or even the 256 extended ASCII values.

Definition 11.4 (Word2FASTA Encoding)**.** Given an ASCII character a with ordinal value $n = \text{ord}(a)$ where $0 \leq n < 128$:

1. Represent n in base 4 as a 4-digit number: $n = d_3 \cdot 4^3 + d_2 \cdot 4^2 + d_1 \cdot 4 + d_0$, padding with leading zeros as needed (since $128 < 4^4 = 256$, four base-4 digits suffice).

2. Add two parity digits: $d_4 = (d_3 + d_2) \bmod 4$ and $d_5 = (d_1 + d_0) \bmod 4$.

3. Map each digit to a nucleotide: $0 \to$ `A`, $1 \to$ `T`, $2 \to$ `G`, $3 \to$ `C`.

4. The six nucleotides $(d_3, d_2, d_1, d_0, d_4, d_5)$ form two codons: $[d_3 d_2 d_1]$ and $[d_0 d_4 d_5]$.

11.7.3 Complete Encoding Example: "Hi"

Let us encode the two-character string "Hi" step by step.

Character 'H' (ASCII 72):

1. Convert 72 to base 4: $72 = 1 \cdot 64 + 0 \cdot 16 + 2 \cdot 4 + 0 = (1, 0, 2, 0)_4$.

2. Parity: $d_4 = (1 + 0) \bmod 4 = 1$; $d_5 = (2 + 0) \bmod 4 = 2$.

3. Full digits: $(1, 0, 2, 0, 1, 2)$.

4. Map to nucleotides: $1 \to$ `T`, $0 \to$ `A`, $2 \to$ `G`, $0 \to$ `A`, $1 \to$ `T`, $2 \to$ `G`.

5. Result: `TAG ATG` (two codons).

Character 'i' (ASCII 105):

1. Convert 105 to base 4: $105 = 1 \cdot 64 + 2 \cdot 16 + 2 \cdot 4 + 1 = (1, 2, 2, 1)_4$.

2. Parity: $d_4 = (1 + 2) \bmod 4 = 3$; $d_5 = (2 + 1) \bmod 4 = 3$.

3. Full digits: $(1, 2, 2, 1, 3, 3)$.

4. Map to nucleotides: $1 \to$ `T`, $2 \to$ `G`, $2 \to$ `G`, $1 \to$ `T`, $3 \to$ `C`, $3 \to$ `C`.

5. Result: `TGG TCC` (two codons).

Complete encoding of "Hi":

`TAG ATG TGG TCC`

This 12-nucleotide DNA sequence can be written in FASTA format as:

```
>prompt_fragment_001 Word2FASTA_v1 len=2
TAGATGTGGTCC
```

Listing 11.4: FASTA encoding of "Hi".

11.7.4 Decoding: FASTA Back to Text

Decoding reverses the process:

1. Read 6 nucleotides at a time.
2. Map each nucleotide back to a digit: `A`→0, `T`→1, `G`→2, `C`→3.
3. Extract the data digits (d_3, d_2, d_1, d_0) and verify the parity digits (d_4, d_5).
4. Convert the base-4 number back to decimal to get the ASCII ordinal.
5. Convert the ordinal to a character.

If the parity check fails, an error is flagged and error-correction can be attempted using the reverse strand.

Round-Trip Fidelity

The Word2FASTA encoding guarantees *perfect round-trip fidelity*: for any ASCII string s, decode(encode(s)) = s. The parity digits provide single-digit error detection, and the reverse complement strand (generated automatically by CodonForge) provides a full backup for error correction. This means that even if a single nucleotide is corrupted, the original text can be recovered.

11.8 SDANS: Strand Displacement Attention Neural System

11.8.1 HelixSim and SDANS: Clarifying the Relationship

Before presenting the technical details, it is important to clarify the relationship between two closely related terms. **HelixSim** (Paper II in the DOGMA research series) is the name of the research project and publication that develops the theoretical framework for compiling symbolic DNA programs into differentiable neural architectures. **SDANS** (Strand Displacement Attention Neural System) is the formal computational system introduced *within* HelixSim. In short:

- **HelixSim** = the research paper and project (Paper II).
- **SDANS** = the compilation target defined by HelixSim that formally bridges CodonForge programs and neural network execution.

Throughout this chapter, we refer to SDANS when discussing the specific compilation mechanics and formal mappings, and to HelixSim when referencing the broader research context. All SDANS definitions, theorems, and algorithms originate from HelixSim (Paper II).

Why a Formal Compilation Target Matters

Having a well-defined compilation target like SDANS—rather than an ad hoc translation from symbolic programs to neural networks—provides three critical guarantees. First, *correctness*: SDANS defines a formal equivalence between symbolic execution and neural execution, meaning compiled programs provably preserve input-output behavior (Definition 11.5). Second, *composability*: because each opcode family maps to a specific class of neural operation, compiled modules can be independently verified and composed without interference. Third, *evolvability*: when the evolutionary training loop (Chapter 12) mutates a genome, only the affected SDANS modules need recompilation, enabling efficient incremental adaptation. Without a formal compilation target, these guarantees would require extensive empirical validation for every program variant—an approach that does not scale.

11.8.2 Overview

SDANS, introduced in the HelixSim design study (Paper II), is a proposed compilation target between CodonForge programs and neural execution. Its attention-based target belongs to the historical architecture family and is not part of canonical non-transformer DOGMA. Formal sketches in a design paper do not establish an implemented equivalence.

11.8.3 Motivation: Why Do We Need SDANS?

CodonForge programs are *symbolic*: they define a precise sequence of operations with deterministic semantics. Neural networks, on the other hand, are *subsymbolic*: they operate on continuous-valued vectors through differentiable transformations. These two paradigms seem incompatible:

Property	**Symbolic (CodonForge)**	**Neural (Transformer)**
Data type	Discrete tokens	Continuous vectors
Control flow	Explicit (if/else, loops)	Implicit (attention routing)
Execution	Sequential, deterministic	Parallel, differentiable
Learning	Not applicable	Gradient descent
Interpretability	High	Low
Generalization	Limited	Strong

SDANS resolves this tension by providing a *compiler* that translates symbolic CodonForge programs into neural architectures. The key insight is that strand displacement reactions—a well-studied model of molecular computation—can be mapped to attention operations.

Definition 11.5 (SDANS Compilation)**.** An *SDANS compilation* is a mapping from a CodonForge program Π to a neural network architecture $\mathcal{N}$ such that:

$$\forall \text{ input } x: \quad \mathcal{N}(x) = \text{Execute}(\Pi, x), \tag{11.9}$$

where Execute is CodonForge's deterministic execution semantics. The compilation preserves the input-output behavior while enabling gradient-based parameter optimization.

11.8.4 How SDANS Maps Opcodes to Neural Operations

The SDANS compilation proceeds by mapping each opcode family to a specific class of differentiable neural operation. This mapping is the core intellectual contribution of SDANS.

Sense Opcodes → Input Projections

Sense opcodes load and perceive input data. In the neural domain, this corresponds to *input embedding and projection*:

$$\text{LOAD\_CONTEXT}(x) \quad \Longrightarrow \quad h_0 = W_{\text{embed}} \cdot x + b_{\text{embed}} \tag{11.10}$$

Each Sense opcode becomes a learned linear projection that maps raw input tokens into the model's hidden representation space. The `SCAN_MOTIF` opcode, for example, becomes a 1D convolution layer that detects local patterns:

$$\text{SCAN\_MOTIF}(h) \quad \Longrightarrow \quad m = \text{ReLU}(\text{Conv1D}(h; W_{\text{motif}})) \tag{11.11}$$

Binding Opcodes → Attention-Based Retrieval

Binding opcodes create associations between data elements. In the neural domain, this is precisely what *attention mechanisms* do:

$$\text{BIND\_VALUE}(Q, K, V) \quad \Longrightarrow \quad \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \tag{11.12}$$

The `ASSOCIATE` opcode becomes a cross-attention layer, while `MERGE` becomes multi-head attention where each head captures a different associative relationship.

Regulation Opcodes → Gated Activation Functions

Regulation opcodes implement conditional logic. In the neural domain, this maps to *gating mechanisms*:

$$\text{IF\_ACTIVE}(R, h) \quad \Longrightarrow \quad g = \sigma(W_g \cdot R + b_g), \quad h' = g \odot h \tag{11.13}$$

where σ is the sigmoid function and $\odot$ is element-wise multiplication. The gate g acts as a "soft conditional": values close to 1 allow the data through (gene active), values close to 0 block it (gene silenced).

The `THRESHOLD` opcode becomes a parameterized activation function:

$$\text{THRESHOLD}(\theta, h) \quad \Longrightarrow \quad h' = \text{ReLU}(h - \theta) \tag{11.14}$$

Expression Opcodes → Output Head Projections

Expression opcodes produce output. In the neural domain, these become *output projection layers*:

$$\text{EMIT\_TEXT}(h) \quad \Longrightarrow \quad p = \text{softmax}(W_{\text{out}} \cdot h + b_{\text{out}}) \tag{11.15}$$

The `FOLD_STRUCT` opcode becomes a structured prediction layer (e.g., a tree-structured decoder), and `EMIT_JSON` becomes a constrained decoding layer that ensures valid JSON output.

Termination Opcodes → Halting Mechanisms

Termination opcodes control when execution stops. In the neural domain, this maps to *adaptive computation time* (ACT) mechanisms:

$$\text{STOP}(h) \quad \Longrightarrow \quad p_{\text{halt}} = \sigma(W_{\text{halt}} \cdot h), \quad \text{halt if } p_{\text{halt}} > 0.5 \tag{11.16}$$

The `CHECKPOINT` opcode becomes a residual connection that preserves state across layers:

$$\text{CHECKPOINT}(h) \quad \Longrightarrow \quad h_{\text{saved}} = h, \quad h' = h + f(h) \tag{11.17}$$

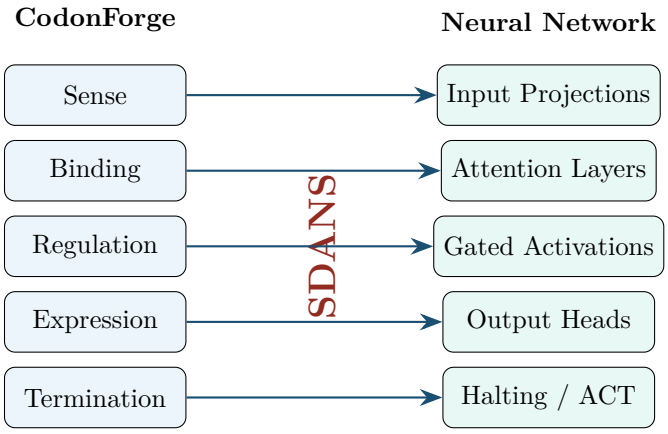


Figure 11.4: SDANS opcode-to-neural mapping. Each CodonForge opcode family is compiled to a specific class of differentiable neural operation.

11.8.5 Strand Displacement as Attention

The name "Strand Displacement" in SDANS refers to a specific molecular computing mechanism. In DNA nanotechnology, *toehold-mediated strand displacement* is a process where a single-stranded DNA molecule displaces one strand of a double-stranded complex by binding to an exposed "toehold" region.

The mathematical structure of strand displacement is remarkably similar to attention:

$$\text{Strand Displacement:} \quad A + B{:}C \xrightarrow{k} A{:}B + C \tag{11.18}$$

$$\text{Attention:} \quad \text{out} = \sum_i \frac{\exp(q \cdot k_i)}{\sum_j \exp(q \cdot k_j)} v_i \tag{11.19}$$

In both cases, a "query" (incoming strand A / query vector q) competes for binding with existing "key-value" pairs (strand complex $B{:}C$ / key-value pairs k_i, v_i). The strength of competition is determined by complementarity (base pairing / dot product similarity).

Programs + Parameters

SDANS resolves a fundamental tension in AI: *programs are interpretable but rigid; neural networks are flexible but opaque.* By compiling interpretable CodonForge programs into differentiable neural architectures, SDANS achieves both: the program structure ensures interpretability and verifiability, while the learned parameters ensure adaptability and generalization. The codon program is the *skeleton*; the neural weights are the *muscles*.

11.9 SDANS Compilation: Step-by-Step

Let us trace the SDANS compilation (as formalized by HelixSim, Paper II) of our "What is DNA?" codon program into a neural architecture.

11.9.1 Step 1: Gene-to-Layer Mapping

Each gene becomes one or more neural network layers:

Gene	Neural Layer(s)	Purpose
Gene 1 (Sense)	Embedding + Conv1D + Cross-attention	Encode query, detect motifs, retrieve knowledge
Gene 2 (Regulate + Express)	Gated MLP + Self-attention + Output projection	Apply constraints, bind answer, emit text

11.9.2 Step 2: Opcode-to-Operation Translation

Each opcode within a gene becomes a specific neural operation, connected sequentially:

```
class WhatIsDNA_SDANS(nn.Module):
    def __init__(self, d_model=512):
        super().__init__()
        # Gene 1: Sense
        self.load_query = nn.Embedding(vocab_size, d_model)     # AAT
        self.sense_intent = nn.Linear(d_model, n_intents)       # AGT
        self.scan_motif = nn.Conv1d(d_model, d_model, 3)        # ATA
        self.match_key = nn.MultiheadAttention(d_model, 8)     # ATG
        self.checkpoint_1 = nn.Identity()  # residual save       # GCC

        # Gene 2: Regulate + Express
        self.gate_check = GatedUnit(d_model)                    # GTA
        self.gate_relevance = GatedUnit(d_model)                # GTG
        self.gate_length = GatedUnit(d_model)                   # GTC
        self.bind_value = nn.MultiheadAttention(d_model, 8)     # TAA
        self.fold_struct = TreeDecoder(d_model)                 # CTA
        self.emit_text = nn.Linear(d_model, vocab_size)         # CAA

    def forward(self, x):
        # Gene 1
        h = self.load_query(x)
```

```
        intent = self.sense_intent(h.mean(dim=1))
        motifs = F.relu(self.scan_motif(h.transpose(1,2)))
        h, _ = self.match_key(h, knowledge_base, knowledge_base)
        h_saved = h  # checkpoint

        # Gene 2 (promoter: h is nonempty -> always true post-sense)
        h = self.gate_check(h)
        h = self.gate_relevance(h)
        h = self.gate_length(h)
        h, _ = self.bind_value(h, h_saved, h_saved)
        h = self.fold_struct(h)
        logits = self.emit_text(h)
        return logits
```

Listing 11.5: SDANS-compiled neural architecture (PyTorch-like pseudocode).

11.9.3 Step 3: Parameter Initialization from Codon Hints

SDANS uses the codon program to guide parameter initialization. For example:

- Sense opcodes with "query-only" semantics (`LOAD_QUERY` vs. `LOAD_CONTEXT`) initialize embeddings with smaller receptive fields.
- Gate opcodes initialize bias terms based on the threshold values specified in the program.
- Expression opcodes initialize output projections based on the target format (text, JSON, code).

This program-guided initialization gives SDANS networks a significant advantage over randomly initialized networks: they start with an architecture and initialization that already encodes the computational intent.

11.10 CodonForge vs. LLVM: A Compiler Comparison

To help students familiar with traditional compilers understand CodonForge's design, we provide a detailed comparison with LLVM, the industry-standard compiler infrastructure.

Table 11.2: Side-by-side comparison of LLVM and CodonForge compilation.

Aspect	LLVM Compiler	CodonForge Compiler
Source language	C, C++, Rust, Swift, etc.	PromptBundles (markdown files)
Intermediate repr.	LLVM IR (SSA form)	Genomic IR (DAG of functional units)
Target language	x86, ARM, WASM machine code	Codon programs (DNA sequences)
Final target	CPU/GPU execution	SDANS neural execution
Basic unit	Instruction (opcode + operands)	Codon (3-letter DNA triplet)
Grouping unit	Function	Gene (promoter + codons + terminator)
Module unit	Module / translation unit	Genome (ordered list of genes)
Entry point	`main()` function	First gene with always-on promoter
Control flow	Branch, jump, call/return	Promoter conditions, `BRANCH`, `LOOP`
Data storage	Registers, stack, heap	Register file $\{R_0, \ldots, R_n\}$
Error handling	Exceptions, error codes	`STOP_ERROR`, `FALLBACK`, `RETRY`
Dead code elim.	Remove unreachable basic blocks	Remove unreachable genes
Constant folding	Evaluate constant expressions	Pre-evaluate static constraints
Inlining	Inline small functions	Gene fusion
Register alloc.	Graph coloring / linear scan	Greedy register reuse
Redundancy	None (single representation)	Double-stranded (forward + reverse)
Error detection	Type system, static analysis	Parity codons, reverse strand verification
Optimization hints	Compiler flags, pragmas	Codon usage bias (opcode aliasing)
Biological analogy	None	Direct mapping to molecular biology

Compiler Analogy Summary

The key structural parallels are:

- LLVM IR ↔ Genomic IR: both are intermediate representations that enable optimization passes.
- Functions ↔ Genes: both are named, callable units of computation.
- SSA registers ↔ CodonForge registers: both hold intermediate values.
- Machine code ↔ Codon program: both are the low-level executable output.
- CPU execution ↔ SDANS neural execution: both are the runtime environments.

The key *differences* are:

- CodonForge has double-stranded redundancy; LLVM does not.
- CodonForge's opcodes are biologically inspired, enabling codon optimization analogous to codon usage bias.
- CodonForge's ultimate execution target is a differentiable neural network, not a fixed hardware architecture.

11.11 Connection to the DOGMA Architecture

CodonForge and SDANS do not exist in isolation; they are critical components of the broader DOGMA architecture introduced in Chapter 7. This section explains how CodonForge connects to the other major subsystems.

11.11.1 CodonForge in the DOGMA Pipeline

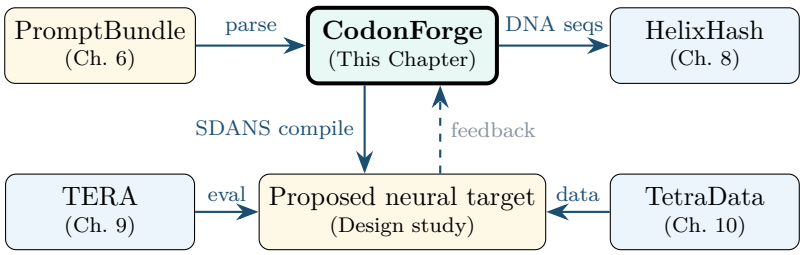


Figure 11.5: Proposed position of CodonForge in the broader DOGMA design space. Compilation to neural architecture remains a hypothesis, not a measured feature of the current baseline.

11.11.2 Proposed CodonForge-to-Neural Compilation

The historical DOGMA-Supreme v2 design proposed that CodonForge would support the following mechanisms. They are retained as a research specification and are not implemented capabilities of the measured baseline in Chapter 14:

- **Architecture generation:** Compile a symbolic program into a declared neural graph.
- **Evolutionary optimization:** Mutate the codon program, compile a descendant, and evaluate it under the external recursive-learning contract.
- **Lineage tracking:** Bind symbolic checkpoints to model checkpoints and evaluation artifacts.
- **Multi-gene expression:** Execute independently compiled subprograms without introducing transformer attention into canonical DOGMA.

Hot Recompilation Hypothesis

A modular compiler could recompile only an affected gene, reducing compilation work from $O(K)$ modules to $O(1)$ when dependencies permit. This complexity claim excludes graph validation, checkpoint conversion, and re-evaluation; it must be tested before being reported as an implementation result.

11.12 Advanced Topics

11.12.1 Codon Optimization Strategies

Just as organisms evolve preferred codon usage patterns for translational efficiency, CodonForge programs can be optimized by choosing among alias codons. The optimization criteria include:

1. **GC content:** Maintaining GC content between 40–60% ensures structural stability in the double-stranded representation.
2. **Cache locality:** Some codon aliases map to operations that are more cache-friendly. For example, sequential register access (R_0, R_1, R_2) is more cache-efficient than random access (R_0, R_7, R_3).
3. **Codon pair bias:** Certain codon pairs execute more efficiently in sequence than others, analogous to biological codon pair bias affecting translational speed.
4. **Repetition avoidance:** Long runs of the same codon can cause "slippage" in the interpreter (analogous to polymerase slippage in biology). The optimizer breaks up such runs with functionally equivalent alternatives.

11.12.2 Error Detection and Correction

CodonForge provides multiple layers of error protection:

- **Parity codons:** As shown in the Word2FASTA encoding, parity digits enable single-error detection.
- **Reverse strand verification:** The reverse complement strand provides a complete backup. If the forward strand is corrupted, the reverse strand can be used to reconstruct the original.
- **Checksum genes:** Special genes can be inserted that compute a checksum over the preceding genes, analogous to CRC checks in digital communication.
- **Proofreading opcodes:** The `RETRY` and `FALLBACK` opcodes enable runtime error recovery, analogous to ribosomal proofreading during translation.

11.12.3 Multi-Gene Regulation Patterns

Complex CodonForge programs use sophisticated regulatory patterns borrowed from biology:

- **Operon structure:** Multiple related genes share a single promoter and are expressed together (analogous to bacterial operons).
- **Enhancer genes:** Special genes that boost the expression of downstream genes when activated (analogous to enhancer elements in eukaryotic DNA).
- **Silencer genes:** Genes that suppress downstream gene expression when activated (analogous to silencer elements).
- **Feedback loops:** A gene's output can activate or repress its own promoter, creating positive or negative feedback (analogous to autoregulatory circuits in gene regulatory networks).

11.13 Exercises

11.1. **[Fundamentals]** How many distinct codon opcodes does a three-letter DNA alphabet ($|\Sigma| = 4$) support? If we wanted 256 opcodes, what minimum codon length would be needed? Show your calculation.

11.2. **[Biology Review]** Using the biological codon table (Table 11.1), translate the following mRNA sequence into amino acids: `AUG-GCA-UUU-GAA-UAA`. Identify the start and stop codons.

11.3. **[Opcode Design]** Write a CodonForge program (using opcode mnemonics) for a text summarization task. Your program should have at least three genes: one for loading and analyzing the input text, one for applying length and quality constraints, and one for generating the summary. Specify promoter conditions for each gene.

11.4. **[Word2FASTA]** Encode the string "DNA" using the Word2FASTA algorithm (Definition 11.4). Show all intermediate steps: ASCII values, base-4 conversion, parity calculation, and nucleotide mapping. Verify your answer by decoding the result back to text.

11.5. **[Strand Construction]** Given the forward strand `5'-GAA TAA CTA CAA CCC-3'`, construct the reverse complement strand. Then decode each codon on both strands into CodonForge opcodes.

11.6. **[Compiler Comparison]** Compare CodonForge's compilation pipeline with LLVM's compilation pipeline (source → IR → optimization → machine code). For each optimization pass in CodonForge (dead code elimination, codon optimization, regulatory simplification, GC content balancing), identify the closest LLVM analogue and explain the similarity.

11.7. **[SDANS]** Consider the CodonForge opcode `GATE_CHECK`. Write the mathematical equation for its SDANS neural equivalent. Then explain: what happens when the gate's learned parameters are updated by gradient descent? Does the program's structure change, or only the gate's threshold? What are the implications for interpretability?

11.8. **[Pipeline Trace]** Trace the prompt "Translate 'hello' to French" through the full CodonForge pipeline:

(a) Parse: identify functional units and their types.

(b) Compile: assign opcodes and construct genes.

(c) Optimize: identify at least one optimization that could be applied.

11.9. [**Theory**] Formalize the SDANS equivalence theorem: under what conditions does the neural execution of a compiled program produce identical outputs to symbolic execution? What happens when neural weights are updated by gradient descent—does the equivalence break? Under what conditions can it be restored?

11.10. [**Research**] The biological ribosome performs error correction during translation (proofreading). Design an error-correction mechanism for CodonForge execution that detects and corrects codon-level errors during program execution. Your design should specify: (a) how errors are detected, (b) what types of errors can be corrected, and (c) the computational overhead of your error-correction scheme.

11.11. [**Coding**] Implement the Word2FASTA encoder and decoder in Python. Your implementation should:

(a) Accept any ASCII string and produce a FASTA-formatted DNA sequence.

(b) Accept a FASTA-formatted DNA sequence and produce the original string.

(c) Verify round-trip fidelity: `decode(encode(s)) == s` for all printable ASCII strings.

(d) Detect single-nucleotide errors using the parity digits.

11.12. [**Design Challenge**] Design a CodonForge program for a multi-turn chatbot. Your design must handle: (a) conversation history loading, (b) context window management (what happens when history is too long?), (c) persona consistency across turns, and (d) graceful handling of off-topic inputs. Use at least five genes and explain the promoter logic for each.

11.14 Key Takeaways

Key Takeaways

- Biological codon translation (mRNA → ribosome → amino acid) directly inspires CodonForge's design: codons are opcodes, genes are execution blocks, and the ribosome is the interpreter.
- CodonForge maps all 64 three-letter DNA codons to computational opcodes, organized into five families: Sense (input perception), Binding (data association), Regulation (control flow), Expression (output generation), and Termination (halting and cleanup).
- Genes are ordered blocks of opcodes bracketed by promoters (activation conditions) and terminators (halt signals), mirroring biological gene structure.
- The three-stage compilation pipeline (Parse → Compile → Optimize) translates prompt bundles into executable codon programs with optimization passes analogous to traditional compilers like LLVM.
- The Word2FASTA bridge enables reversible translation between text and DNA-encoded representations, with parity-based error detection and round-trip fidelity.
- Forward and reverse complement strands provide redundancy, error detection, and compatibility with HelixHash's double-stranded analysis.
- SDANS compilation maps symbolic CodonForge programs to differentiable neural architectures, achieving both interpretability (from program structure) and adaptability (from learned parameters).
- CodonForge receives prompt bundles and can produce symbolic DNA sequences; compilation into neural architectures via SDANS remains a design hypothesis.

Suggested Reading

- Chapter 1, Section on the Genetic Code: biological codons as the original inspiration.
- Chapter 7: CodonForge's role in the DOGMA pipeline.
- Chapter 14: the measured baseline and the evidence boundary separating implementation from design studies.
- Soloveichik et al. [2010]: CRN Turing completeness—theoretical foundation for molecular program execution.
- Alberts et al., *Molecular Biology of the Cell*: the definitive reference on biological translation.
- Lattner & Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation": for deeper understanding of the compiler comparison.

Part IV

Building the Evolutor

Chapter 12

Recursive, Evidence-Gated Training

A system does not improve because it changes itself. It improves only when a changed descendant survives a valid test.

— DOGMA Research Protocol

"Self-improving" is an appealing phrase and a dangerous specification. A model that trains on its own unchecked output can amplify errors, contaminate its evaluation set, forget rare cases, or optimize an easy proxy. Recursive learning therefore requires more structure than repeatedly launching a training script.

This chapter defines the DOGMA recursive research loop. Gradient descent still trains each neural candidate. Evolutionary search operates one level above it: it chooses bounded changes to data, optimization, or architecture; evaluates the descendants; and decides what may be retained. This separation follows the spirit of population-based training [Jaderberg et al., 2017] and quality-diversity search [Mouret and Clune, 2015], while adding strict data provenance and deployment gates.

12.1 Three Loops, Three Responsibilities

Recursive DOGMA System

A recursive DOGMA system contains three nested loops:

1. the **parameter loop**, where an optimizer updates weights on a fixed candidate;
2. the **candidate loop**, where a controller proposes one declared mutation and trains a descendant; and
3. the **research loop**, where people revise hypotheses, frozen evaluations, and safety constraints.

The candidate loop may automate experiments. It may not silently rewrite the evaluation rules or promote its own weights.

Let a candidate be

$$c = (\theta, h, d, a),$$

where θ are learned weights, h are training hyperparameters, d is a versioned data manifest, and a is an architecture declaration. A lineage edge

$$c_{t+1} = M(c_t; m_t)$$

must record the mutation m_t, the parent checkpoint, and the code revision. Without this information, a result cannot be reproduced or rolled back.

12.2 The Candidate Lifecycle

The production loop has seven stages.

1. **Observe.** Collect failures from immutable held-out tests and operational traces. Do not train on the held-out answers.
2. **Diagnose.** Classify each failure: sequence validity, language readability, factuality, calibration, long-range retention, contamination, or resource cost.
3. **Propose.** Change exactly one bounded variable in an ordinary descendant cycle. Examples are learning rate, dropout, curriculum temperature, or adaptive-sampling strength.
4. **Train.** Produce a new checkpoint from the declared parent and data manifest. Training is isolated from the live service.
5. **Challenge.** Run frozen probes with at least two fixed seeds, held-out likelihood, regression tests, and resource measurements.
6. **Archive.** Place scientifically valid stepping stones in a behavior-indexed archive even when they are not the global winner.
7. **Promote or reject.** Atomically update the live pointer only if all hard gates pass and the global objective improves. Otherwise retain the parent.

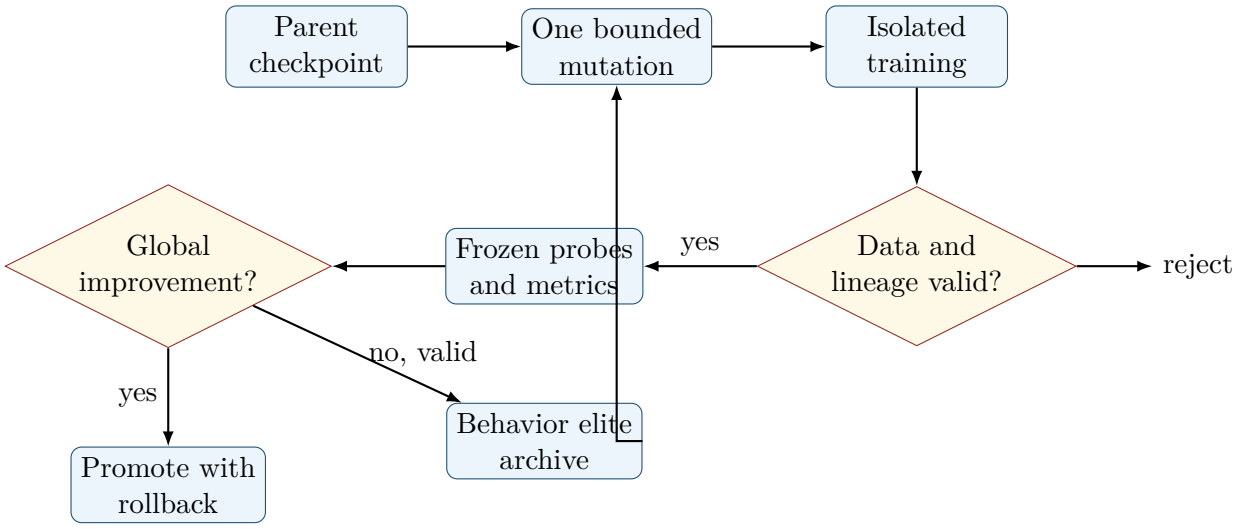


Figure 12.1: A descendant may be archived without being deployed. Invalid lineage or contaminated evaluation causes rejection before performance is considered.

12.3 Data Provenance and Model Collapse

Recursive training creates a feedback channel: model outputs can become future inputs. Repeatedly replacing real observations with generated approximations can erase distribution tails and degrade later generations, a failure studied as model collapse [Shumailov et al., 2024]. DOGMA therefore separates *real anchors* from *derived or synthetic* material.

For sampling weights w_i, define

$$r = \frac{\sum_{i \in R} w_i}{\sum_i w_i},$$

where R is the set of records with audited real-data provenance. The canonical policy requires

$$r \geq 0.65, \qquad 1 - r \leq 0.35.$$

This is a policy choice, not a universal constant. It is deliberately conservative for a small research program. If an experiment uses another mixture, it must receive another contract name and be reported separately.

Document hashes detect exact overlap among train, validation, and test splits. Near-duplicate detection and homology-aware splitting remain required for serious genomic benchmarks: exact hashes alone cannot detect highly similar subsequences or related organisms.

12.4 A Multi-Probe Evaluator

Perplexity answers one narrow question: how surprised was the model by held-out next bytes under teacher forcing? Let

$$\mathrm{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(x_i \mid x_{<i})\right).$$

It does not test whether autoregressive sampling remains on a readable trajectory. The frozen `dogma-recursive-v2` suite therefore separates three capabilities:

1. **Sequence validity:** produce a sufficiently long run containing only canonical A, C, G, T symbols.
2. **Readable explanation:** explain the biological central dogma in plain language without invalid bytes.
3. **Self-critique:** state one model limitation and one experiment that could test it.

Each probe runs under two fixed generation seeds. These checks are intentionally small smoke tests, not claims of scientific knowledge. A stronger evaluation program must add exact genomic tasks, out-of-distribution species, reverse complement invariance, length extrapolation, calibration, and matched baselines such as HyenaDNA, Caduceus, and DNABERT-2.

For archive ranking, the current scalar objective is

$$J(c) = P_{\text{test}} + 0.25 P_{\text{valid}} + \lambda_D + \lambda_E + \lambda_C,$$

where each λ is zero for a passed probe and positive for a failed probe. Lower is better. Hard promotion still requires every probe to pass; the scalar cannot trade away a critical failure.

12.5 Why Keep a Quality-Diversity Archive?

A single best-so-far chain can become trapped. A candidate may improve DNA syntax while temporarily losing explanatory language; another may preserve language but explore a different recurrent mechanism. MAP-Elites [Mouret and Clune, 2015] keeps the best candidate in each behavior cell rather than collapsing all behavior into one leaderboard.

The current archive descriptor records:

$$\text{architecture features} : \{\text{DNA}, \text{explanation}, \text{critique}\} \text{ passed}.$$

Every third scheduled cycle may branch from an archive elite. The live checkpoint remains unchanged until a descendant wins the complete gate. This provides exploration without converting production into an online experiment.

Remark 12.1. An archive is not evidence of superiority. It is a memory of promising, reproducible stepping stones. Global promotion and scientific publication require stronger tests than archive retention.

12.6 Controlled Mutation

Ordinary descendant cycles mutate exactly one scalar:

$$h'_j = \text{clip}(\alpha h_j, \ell_j, u_j).$$

The mutation record stores j, α, ℓ_j, u_j, the parent value, and the candidate value. One-variable mutation is slower than unconstrained search but makes failure analysis possible on limited compute.

Architecture changes are separate branches. Adding a layer or changing a state dimension changes tensor shapes, so an optimizer state from the parent may no longer be valid. Such experiments start from a declared compatible checkpoint or from initialization and use matched compute. They never masquerade as an ordinary resumed cycle.

12.7 Promotion Contract

A candidate can replace the live model only when all of the following hold:

1. the architecture audit confirms no self-attention or transformer layer;
2. parent and candidate checkpoints are distinct and recorded;
3. at least 65% of sampling mass remains on real anchors;
4. train/evaluation overlap is zero under the declared detector;
5. two distinct generation seeds reproduce every probe;
6. validation and test perplexity are finite and within the regression budget;
7. no critical regression is present; and
8. $J(c_{\text{candidate}}) < J(c_{\text{live}})$.

Promotion creates a pointer to the exact evaluated checkpoint. It does not copy an ambiguous "best" file from another run. The previous live checkpoint is retained for rollback.

Algorithm 3 Evidence-gated recursive DOGMA cycle

$p \leftarrow$ choose live parent or scheduled archive elite
$m \leftarrow$ choose one bounded mutation
$c \leftarrow$ train descendant of p under manifest d
if lineage invalid or real-anchor quota fails or split overlap > 0 **then**
    reject c
else
    $q \leftarrow$ run frozen probes under seeds 17 and 29
    $J(c) \leftarrow$ held-out perplexity plus explicit probe penalties
    if c is best in its behavior cell **then**
        archive c
    end if
    if all hard gates pass and $J(c) < J(p_{\text{live}})$ **then**
        atomically promote c; retain rollback to p_{live}
    end if
end if

12.8 Connection to Self-Improving Systems

Population-based training adapts hyperparameters while a population learns [Jaderberg et al., 2017]. Quality-diversity methods preserve multiple useful behaviors [Mouret and Clune, 2015]. The Darwin Gödel Machine explores code changes and evaluates descendants on coding tasks [Zhang et al., 2025]; AlphaEvolve combines model-generated programs with automated evaluators [Novikov et al., 2025]. These systems support a general lesson: recursion becomes useful when the evaluator is external to the candidate and failures are cheap to reject.

DOGMA adopts that lesson conservatively. It currently evolves bounded training choices and archives checkpoint lineages. Self-editing source code remains a future sandboxed research track, because tests can be incomplete and an agent can overfit the evaluator. The phrase "evolves itself" should therefore mean "produces descendants under an external promotion contract," not "unilaterally changes the running system."

12.9 Exercises

12.1. A corpus has real-anchor weight 18 and synthetic weight 12. Compute r. Does it satisfy the canonical policy?

12.2. A model has validation perplexity 2.8, test perplexity 1.4, and fails only the explanation probe. Compute J when $\lambda_E = 1.5$. Explain why it still cannot be promoted.

12.3. Design a behavior descriptor with two architectural and three capability dimensions. How many archive cells are possible?

12.4. Give an example where a candidate improves test perplexity while generation quality worsens. Why is this possible?

12.5. Propose a homology-aware split test stronger than exact document hashes. State its computational cost and possible false positives.

12.6. Write a mutation record for changing dropout from 0.10 to 0.11. Include enough information to reproduce and reverse it.

12.7. Design an out-of-distribution probe that distinguishes memorizing DNA motifs from learning a useful sequence rule.

12.8. Identify one way a candidate could game each of the three smoke probes. Add a counter-test for every weakness.

12.10 Key Takeaways

Key Takeaways

- Recursive learning is a checkpoint lineage governed by an evaluator, not uncontrolled online self-modification.
- Gradient descent trains candidates; evolutionary search chooses bounded experiments above that loop.
- Real-data anchors, isolated evaluation, multiple seeds, and rollback are part of the learning algorithm, not administrative details.
- Perplexity and free generation measure different failure modes.
- A quality-diversity archive preserves stepping stones while a stricter global gate protects the live model.

Suggested Reading

- Jaderberg et al. [2017] on population-based training.
- Mouret and Clune [2015] on MAP-Elites and quality-diversity search.
- Shumailov et al. [2024] on recursive synthetic-data failure.
- Zhang et al. [2025] and Novikov et al. [2025] on evaluator-driven machine discovery.

Chapter 13

Foundation Models on Genomic Data

The genome is not written in English, nor in Python. It is written in a four-letter alphabet whose grammar predates language by three billion years. A foundation model that cannot read this alphabet is illiterate in the most fundamental sense.

— DOGMA Design Manifesto

Chapter 12 showed how to *evolve* prompt genomes through gradient-free optimization. But the evolutionary loop assumes an underlying language model capable of processing genomic data. Where does that model come from? How should it tokenize DNA, learn codon structure, and integrate biological sequences with natural language?

This chapter addresses these questions by building foundation models for genomic data from first principles. We begin with the deceptively simple question: *what is genomic data?* We explain the FASTA format step by step, then tackle the tokenization problem—comparing BPE, k-mer, and codon-based approaches before showing how DOGMA's CodonForge tokenizer unifies biological and textual data in a single vocabulary. We introduce curriculum learning as a training strategy, detail the pre-training objectives (masked language modeling and next-token prediction), and explain how the double-helix structure of DNA informs helix-aware training. We cover multi-scale TetraMemory, the full training data pipeline from GenBank to batching, and evaluation benchmarks for genomic foundation models. Chapter 14 then separates the implemented small baseline from the larger design studies in this chapter.

13.1 What Is Genomic Data?

Before building models, we must understand the data. Genomic data is the digital representation of the molecules that encode hereditary information in all living organisms. At its core, DNA (deoxyribonucleic acid) is a polymer of four nucleotide bases: adenine (A), thymine (T), cytosine (C), and guanine (G). These four letters constitute the "alphabet" of life.

13.1.1 From Molecules to Sequences

In a living cell, DNA exists as a double-stranded helix. The two strands are *complementary*: A always pairs with T, and C always pairs with G. If one strand reads `ATGCGA`, the opposite strand

reads `TACGCT` (in the reverse direction). This complementarity means that storing one strand is sufficient to reconstruct both.

When biologists *sequence* DNA—determining the order of bases in a sample—the output is a text string over the alphabet $\Sigma_{\text{DNA}} = \{\texttt{A}, \texttt{T}, \texttt{C}, \texttt{G}\}$. A single human genome contains approximately 3.2 billion such characters. A bacterial genome might contain 1–10 million characters. A viral genome can be as small as a few thousand characters. All of these are stored, shared, and processed in a standard text format called **FASTA**.

13.1.2 The FASTA Format: Step by Step

FASTA is the universal standard for representing biological sequences. Understanding it is essential for anyone building genomic models. A FASTA file has a simple structure:

1. **Header line.** Begins with `>` and contains metadata: an identifier, organism name, chromosome, and other annotations. This is free-form text.
2. **Sequence lines.** One or more lines of nucleotide characters (A, T, C, G), typically wrapped at 60 or 80 characters per line. No spaces, no numbers—just the raw sequence.
3. **Multiple records.** A file can contain many sequences, each starting with its own `>` header.

Example 13.1 (Anatomy of a FASTA File)**.** Consider the following FASTA file containing two sequences:

```
>gene_001 | Homo sapiens | insulin gene | chromosome 11
ATGGCCCTGTGGATGCGCCTCCTGCCCCTGCTGGCGCTGCTGGCCCTCTG
GGGACCTGACCCAGCCGCAGCCTTTGTGAACCAACACCTGTGCGGCTCACA
CCTGGTGGAAGCTCTCTACCTAGTGTGCGGGGAACGAGGCTTCTTCTACAC
>gene_002 | Escherichia coli | beta-galactosidase | lacZ
ATGACCATGATTACGCCAAGCTATTTAGGTGACACTATAGAATACTCAAGC
TATGCATCCAACGCGTTGGGAGCTCTCCCATATGGTCGACCTGCAGGCGGC
```

Let us dissect the first record:

- `>gene_001` is the sequence identifier.
- `Homo sapiens` indicates the organism (human).
- `insulin gene` is the gene name.
- `chromosome 11` locates it in the genome.
- The lines below the header contain the nucleotide sequence, which wraps across multiple lines but is read as one continuous string.

FASTA Is Inherently Multimodal

A FASTA file is not purely a "genomic" format. The header lines are natural language, while the sequence lines are biological data. A foundation model trained on raw FASTA files must learn *both* modalities simultaneously. The `>` character serves as a natural boundary token, signaling the transition from sequence to metadata. This multimodal nature is not a complication—it is an *opportunity*: the model can learn to associate natural language descriptions with the genomic patterns they describe.

13.1.3 Beyond DNA: RNA and Protein Sequences

FASTA also stores RNA sequences (using U instead of T) and protein sequences (using the 20-letter amino acid alphabet). A genomic foundation model must handle all three:

Molecule	Alphabet Size	Unit	Example
DNA	4	Nucleotide	`ATGCGATCGA`
RNA	4	Nucleotide	`AUGCGAUCGA`
Protein	20	Amino acid	`MRIAKLFVTG`

The Central Dogma in Data Terms

The biological flow of information—DNA → RNA → Protein—is also a data transformation pipeline. DNA is *transcribed* into RNA (T becomes U), and RNA is *translated* into protein (every three nucleotides map to one amino acid). A foundation model that learns these transformations has internalized the Central Dogma of molecular biology as a sequence-to-sequence mapping. This is exactly what DOGMA's architecture is designed to exploit (Chapter 7).

13.1.4 Ambiguity Codes and Real-World Messiness

Real genomic data is not always clean. The IUPAC nucleotide code includes ambiguity characters: `N` means "any base," `R` means "A or G" (purine), `Y` means "C or T" (pyrimidine), and so on. Sequencing errors, gaps, and low-quality regions are common. A robust tokenizer must handle these gracefully.

Handling N Characters

In large genomic datasets, stretches of `N` characters indicate regions where the sequencer could not determine the base identity. These runs can be hundreds or thousands of characters long. During preprocessing, DOGMA replaces runs of more than 50 consecutive `N` characters with a special `<GAP>` token, preserving positional information without wasting model capacity on uninformative input. Isolated `N` characters (fewer than 50) are retained as byte tokens, since they carry information about local sequence uncertainty.

13.2 Tokenizing Genomic Sequences

Tokenization—the mapping from raw input to discrete units consumed by a model—is often treated as a solved preprocessing step. For natural language, subword tokenizers such as BPE [Sennrich et al., 2016] and SentencePiece [Kudo and Richardson, 2018] perform well. For genomic data, the situation is far more nuanced.

13.2.1 Why Subword Tokenization Fails for Genomic Data

Subword tokenizers learn merge rules from corpus statistics: frequent character pairs are merged into tokens, and the process repeats until a target vocabulary size is reached. This works for natural language because word frequencies follow Zipf's law—a small vocabulary of common subwords covers most text.

Genomic sequences violate these assumptions in four fundamental ways:

1. **Uniform character distribution.** DNA consists of four nucleotides (A, T, C, G) with roughly equal frequency in many organisms. BPE merges are driven by frequency imbalances; when all bigrams occur at similar rates, the merge order becomes arbitrary and biologically meaningless.

2. **Reading frame sensitivity.** The biological meaning of a DNA sequence depends on its reading frame—the codon boundaries that partition nucleotides into triplets. A BPE tokenizer that merges `AT` into a single token will split the codon `ATG` (start codon) differently depending on context, destroying frame information.

3. **No whitespace delimiters.** Natural language has spaces and punctuation that provide natural token boundaries. Genomic sequences are undelimited streams of characters, offering no structural cues to a frequency-based tokenizer.

4. **Reverse complement ambiguity.** The same biological information can be represented on either DNA strand. A subword vocabulary learned from one strand orientation will be suboptimal for the reverse complement, creating asymmetric representations of equivalent biological content.

Codons as Nature's Tokens

Biology solved the tokenization problem long before computer science. The genetic code groups nucleotides into codons—non-overlapping triplets that map to amino acids. This fixed-width tokenization ensures that the reading frame is preserved, that every position has unambiguous membership in exactly one token, and that the mapping from sequence to function is deterministic. Any computational tokenizer for genomic data should respect, or at least not destroy, this codon structure.

13.2.2 Three Approaches to Genomic Tokenization

We now compare three tokenization strategies for genomic data: BPE applied to DNA, k-mer tokenization, and codon-based tokenization.

Byte Pair Encoding (BPE) on DNA

BPE can be applied to DNA by treating the sequence as text and learning merge rules. The process is:

1. Start with character-level tokens: each of A, T, C, G is a token.

2. Count all adjacent token pairs in the corpus.

3. Merge the most frequent pair into a new token.

4. Repeat until the desired vocabulary size is reached.

On DNA, BPE typically learns to merge common dinucleotides first (`AT`, `GC`, etc.), then trinucleotides, and so on. The resulting vocabulary has variable-length tokens with no guaranteed alignment to codon boundaries.

k-mer Tokenization

k-mer tokenization splits a DNA sequence into overlapping or non-overlapping windows of length k. For example, with $k = 6$, the sequence `ATGCGATCGA` produces the 6-mers: `ATGCGA`, `TGCGAT`, `GCGATC`, `CGATCG`, `GATCGA`. The vocabulary size is 4^k (4,096 for $k = 6$). This approach was popularized by DNABERT [Ji et al., 2021a].

Advantages: Every k-mer captures local context of length k. The vocabulary is deterministic—no training required. **Disadvantages:** Overlapping k-mers create redundancy. The vocabulary grows exponentially with k (4^k tokens). Most critically, k-mer boundaries are arbitrary and do not respect codon structure unless k is a multiple of 3.

Codon-Based Tokenization

Codon-based tokenization groups nucleotides into non-overlapping triplets aligned to the biological reading frame:

Codon-Aware Tokenizer

Given a coding DNA sequence x with known reading frame offset $\delta \in \{0, 1, 2\}$, the codon-aware tokenizer produces:

$$\tau_{\text{codon}}(x, \delta) = (x_{\delta+1}x_{\delta+2}x_{\delta+3},\ x_{\delta+4}x_{\delta+5}x_{\delta+6},\ \ldots), \tag{13.1}$$

with vocabulary size $|\mathcal{V}_{\text{codon}}| = 4^3 + |\mathcal{V}_{\text{special}}| = 64 + |\mathcal{V}_{\text{special}}|$, where $\mathcal{V}_{\text{special}}$ includes tokens for frame-unknown regions, non-coding sequence, and FASTA metadata.

Advantages: Perfectly aligned with biology. Small vocabulary (64 core tokens). Every token carries direct biological meaning (each codon encodes an amino acid). **Disadvantages:** Requires knowing the reading frame, which is not always available. Cannot handle non-coding DNA, intergenic regions, or introns, where no reading frame exists.

13.2.3 Comparison Table: Tokenization Methods

The following table summarizes the trade-offs:

Table 13.1: Comparison of tokenization methods for genomic sequences.

Method	Vocab Size	Frame-Aware	Overlap	Training Req'd	Best For
Byte-level	256	No	None	No	Universal mixed data
BPE	Tunable	No	None	Yes	Large NLP corpora
k-mer (k=6)	4,096	No	Yes	No	Local motif discovery
Codon (k=3)	64+special	Yes	None	No	Coding sequences
CodonForge	328	Adaptive	None	No	Unified text+DNA

No Single Tokenizer Wins on All Data

Each tokenization method excels in a specific regime: BPE for text-heavy corpora, k-mers for motif analysis, codons for coding regions, and byte-level for universal coverage. The DOGMA approach resolves this tension not by choosing one method but by building a *unified* tokenizer—CodonForge—that adapts its behavior to the input modality. This is the subject of Section 13.3.

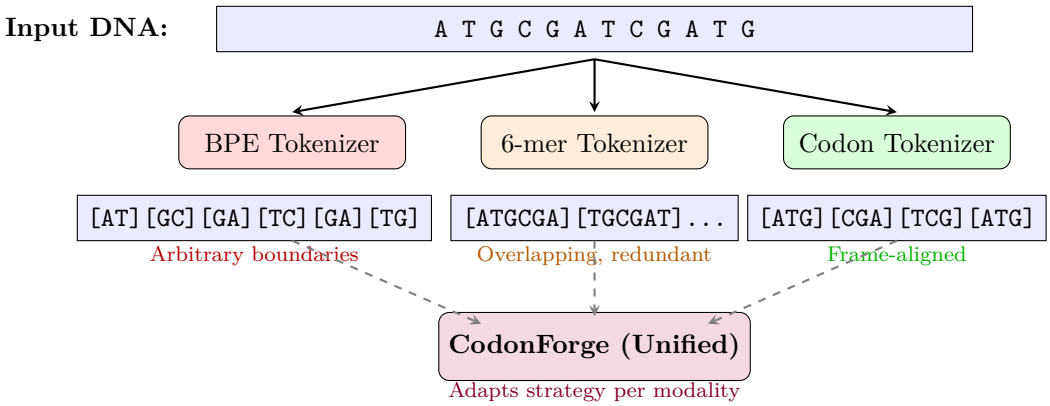


Figure 13.1: Comparison of tokenization strategies for the DNA sequence `ATGCGATCGATG`. BPE produces arbitrary boundaries, k-mers produce overlapping tokens, and codon tokenization produces frame-aligned triplets. CodonForge unifies all three strategies adaptively.

13.3 DOGMA's Unified Tokenization: CodonForge

DOGMA's solution to the tokenization dilemma is **CodonForge** (Chapter 11), a unified tokenizer that handles both text and DNA within a single vocabulary. This section explains how CodonForge works at the level relevant to foundation model training.

13.3.1 The CodonForge Vocabulary

CodonForge partitions a 328-token vocabulary into four regions:

CodonForge Vocabulary Structure

The CodonForge vocabulary $\mathcal{V}_{\text{CF}}$ is composed of four disjoint regions:

$$\mathcal{V}_{\text{CF}} = \mathcal{V}_{\text{byte}} \cup \mathcal{V}_{\text{codon}} \cup \mathcal{V}_{\text{modal}} \cup \mathcal{V}_{\text{struct}}, \tag{13.2}$$

where:

- $\mathcal{V}_{\text{byte}} = \{0, 1, \ldots, 255\}$ handles raw bytes (all ASCII characters, including nucleotide letters).
- $\mathcal{V}_{\text{codon}} = \{\texttt{AAA}, \texttt{AAT}, \texttt{AAC}, \ldots, \texttt{GGG}\}$ contains all 64 codon tokens.
- $\mathcal{V}_{\text{modal}} = \{\texttt{<DNA>}, \texttt{</DNA>}, \texttt{<PROT>}, \texttt{</PROT>}, \texttt{<TEXT>}, \texttt{</TEXT>}, \texttt{<STRUCT>}, \texttt{</STRUCT>}\}$ provides 8 modality boundary markers.
- $\mathcal{V}_{\text{struct}}$ contains discretized structural features (e.g., secondary structure codes, distance bins), for a total of $|\mathcal{V}_{\text{CF}}| = 256 + 64 + 8 = 328$ tokens.

Note that the 64 codon tokens *overlap* with specific byte trigrams (e.g., the codon token `ATG` and the byte sequence 65-84-71 represent the same characters). The key difference is that a codon token is a *single* token occupying one position in the sequence, while the byte representation uses three positions. CodonForge dynamically chooses which representation to use based on the input modality.

13.3.2 Adaptive Tokenization Logic

When CodonForge processes a FASTA file, it applies different tokenization strategies to different regions:

1. **Header lines** (after `>`): Tokenized as bytes. Natural language text is best served by character-level representation, and the byte vocabulary covers all ASCII characters.
2. **Coding sequences** (with known reading frame): Tokenized as codons. Each triplet becomes a single codon token, preceded by `<DNA>` and followed by `</DNA>`.
3. **Non-coding sequences** (no known reading frame): Tokenized as bytes. Individual nucleotides become byte tokens within `<DNA>`...`</DNA>` markers.
4. **Protein sequences**: Each amino acid letter becomes a byte token within `<PROT>`...`</PROT>` markers.

How CodonForge Detects Modality

CodonForge uses a simple rule-based classifier to determine the modality of each region. Lines beginning with `>` are headers (text). Sequences following a header are classified as DNA if more than 90% of characters are in {A, T, C, G, N}, as protein if more than 80% are standard amino acid letters, and as text otherwise. Within DNA regions, coding status is determined by metadata annotations in the header (e.g., "CDS" for coding sequence) or by the presence of start/stop codon patterns. This classification runs once during preprocessing and is cached for training.

13.3.3 Why Unified Tokenization Matters

A unified vocabulary allows a single model to process mixed-modality inputs without mode switching. Consider a training example that contains a gene description in English, followed by its DNA sequence, followed by its protein product:

```
<TEXT>The insulin gene encodes a peptide hormone</TEXT>
<DNA>ATG GCC CTG TGG ATG CGC CTC CTG</DNA>
<PROT>MALWMRLL</PROT>
```

With CodonForge, this entire example is a single token sequence. The model learns cross-modal relationships—that the text "insulin" co-occurs with specific DNA and protein patterns—through the same next-token prediction objective used for all training.

One Vocabulary, Three Languages

CodonForge's unified vocabulary lets the model treat English, DNA, and protein as dialects of a single language, separated by modality markers. This is analogous to multilingual language models that use a shared vocabulary across languages with language tags. The key advantage is that representations are *shared*: the model can learn that certain DNA patterns predict certain protein structures, and that certain English descriptions predict certain DNA patterns, all within one embedding space.

13.4 Byte-Level Tokenization as Universal Fallback

While CodonForge provides the primary tokenization for DOGMA, byte-level tokenization serves as the universal fallback and the foundation upon which CodonForge is built.

Byte-Level Tokenizer for Genomic Data

Let $\mathcal{V}_{\text{byte}} = \{0, 1, \ldots, 255\}$ be the full byte vocabulary. The byte-level tokenizer τ_{byte} maps an input string x to a sequence of byte values:

$$\tau_{\text{byte}}(x) = (\text{ord}(x_1), \text{ord}(x_2), \ldots, \text{ord}(x_n)), \tag{13.3}$$

where $\text{ord}(\cdot)$ returns the byte value of a character. For a mixed corpus containing both English text and DNA, the same 256-token vocabulary handles both modalities without any domain-specific merge rules.

The byte-level approach has three key advantages for genomic foundation models. First, it is *modality-agnostic*: the same tokenizer handles FASTA headers (English text), nucleotide sequences, amino acid sequences, and metadata without switching modes. Second, it *preserves positional granularity*: every nucleotide occupies exactly one position, so positional encodings align with biological positions. Third, it is *deterministic*: there are no merge rules to learn, no vocabulary to tune, and no edge cases where the same sequence tokenizes differently in different contexts.

13.5 The Bigram Baseline

Before building complex transformer models, we establish a minimal baseline: a bigram language model over genomic sequences, implemented with zero external dependencies. This baseline is not a

throwaway exercise—it reveals genuine biological structure and provides a critical diagnostic tool for evaluating more complex models.

13.5.1 Character-Level Statistics of Genomic Sequences

A bigram model estimates the probability of each character given only its immediate predecessor:

$$P(x_t \mid x_{<t}) \approx P(x_t \mid x_{t-1}) = \frac{C(x_{t-1}, x_t)}{\sum_c C(x_{t-1}, c)}, \tag{13.4}$$

where $C(a, b)$ counts the number of times character b follows character a in the training corpus. For a pure DNA corpus over alphabet {A, T, C, G}, this yields a 4×4 transition matrix.

Example 13.2 (Building a Bigram Transition Matrix)**.** Suppose we train a bigram model on the short DNA sequence `ATGCATGCATGC`. We count every pair of consecutive characters:

	$\to$ A	$\to$ T	$\to$ C	$\to$ G
A $\to$	0	3	0	0
T $\to$	0	0	0	3
C $\to$	2	0	0	0
G $\to$	0	0	3	0

Normalizing each row gives the transition probabilities. In this toy example, the matrix is deterministic: A always precedes T, T always precedes G, G always precedes C, and C always precedes A (with one fewer count). Real genomic data produces less extreme but still non-uniform matrices.

Zero-Dependency Bigram Model

The bigram model can be implemented using only Python's standard library: a dictionary for counts, division for probabilities, and `random.choices` for sampling. This makes it an ideal pedagogical tool and sanity check. If a transformer cannot outperform a bigram model on genomic sequence prediction, something is fundamentally wrong with the architecture or training. The expected per-character cross-entropy for a uniform random baseline over four nucleotides is $\log_2 4 = 2.0$ bits; a bigram model on real genomic data typically achieves 1.85–1.95 bits, reflecting the modest but real dinucleotide biases in biological sequences.

13.5.2 What Bigrams Reveal About Biology

Despite their simplicity, bigram statistics expose genuine biological structure. The dinucleotide frequency matrix is non-uniform in real genomes: CpG dinucleotides are suppressed in vertebrate genomes due to methylation-driven deamination, while AA/TT dinucleotides are enriched in nucleosome positioning sequences. A bigram model trained on human genomic DNA will learn these biases automatically, achieving lower perplexity than a uniform model.

The Bigram as Diagnostic Tool

The gap between bigram perplexity and transformer perplexity quantifies how much *long-range* structure exists in a genomic corpus. For random DNA, the gap is near zero—there is no structure beyond dinucleotide frequencies. For coding DNA, the gap is substantial because codon usage, splice signals, and regulatory motifs create dependencies spanning tens to thousands of nucleotides. For mixed text-DNA corpora, the gap is even larger due to the rich long-range structure of natural language. Monitoring this gap during training reveals whether the model is learning genuine biological patterns or merely memorizing local statistics.

13.6 Pre-Training Objectives for Genomic Foundation Models

A foundation model learns from data through its *pre-training objective*—the loss function that defines what the model should predict. For genomic data, two objectives dominate: masked language modeling (MLM) and next-token prediction (NTP). Each has distinct advantages.

13.6.1 Masked Language Modeling on DNA

Masked language modeling, popularized by BERT [Devlin et al., 2019], randomly masks a fraction of input tokens and trains the model to predict them from context. Applied to DNA:

Masked Language Modeling for DNA

Given a DNA sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, select a random subset $\mathcal{M} \subset \{1, \ldots, n\}$ with $|\mathcal{M}| = \lfloor 0.15 \cdot n \rfloor$ (15% of positions). Replace each x_i for $i \in \mathcal{M}$ with a `[MASK]` token. The MLM loss is:

$$\mathcal{L}_{\text{MLM}} = -\frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \log P_\theta(x_i \mid \mathbf{x}_{\setminus \mathcal{M}}), \tag{13.5}$$

where $\mathbf{x}_{\setminus \mathcal{M}}$ denotes the sequence with masked positions.

Why MLM works for DNA: Nucleotides are constrained by their neighbors. A masked `G` in a coding region can often be predicted because the codon it belongs to is constrained by the amino acid sequence. A masked position in a promoter region can be predicted because promoter motifs (e.g., TATA box) have conserved sequences. MLM forces the model to learn these contextual dependencies from both directions (bidirectional context).

Example 13.3 (MLM on a Coding Sequence)**.** Consider the coding sequence with reading frame marked:

```
Original:  ATG | CGA | TCG | ATG | TAA
Masked:    ATG | C[M]A | TCG | A[M]G | TAA
```

The model must predict that position 5 is `G` (completing the codon `CGA` → Arg) and position 11 is `T` (completing `ATG` → Met, the start codon). A well-trained model will assign high probability to the correct nucleotides because it has learned codon usage patterns and the constraint that ATG is almost always a start codon.

13.6.2 Next-Token Prediction on Codons

Next-token prediction (NTP), the objective used by GPT-family models, predicts each token from its left context only:

$$\mathcal{L}_{\text{NTP}} = -\frac{1}{T}\sum_{t=1}^{T}\log P_\theta(x_t \mid x_{<t}). \tag{13.6}$$

When applied at the codon level (using CodonForge's codon tokens), this becomes *next-codon prediction*: given a sequence of codons, predict the next codon. This objective is particularly natural for coding sequences because the genetic code imposes strong constraints on codon transitions.

Codon Usage Bias as a Training Signal

Not all synonymous codons (codons encoding the same amino acid) are used equally. Organisms have strong codon usage biases—some codons are preferred because they match abundant tRNAs, leading to faster translation. A model trained with NTP on codons learns these biases: when predicting the next codon in a human gene, it will assign higher probability to preferred human codons. This is a biologically meaningful pattern that byte-level models must infer indirectly.

13.6.3 Combining MLM and NTP

DOGMA uses both objectives during pre-training, applied to different portions of the training data:

$$\boxed{\mathcal{L}_{\text{pretrain}} = \lambda_{\text{NTP}} \cdot \mathcal{L}_{\text{NTP}} + \lambda_{\text{MLM}} \cdot \mathcal{L}_{\text{MLM}} + \lambda_{\text{align}} \cdot \mathcal{L}_{\text{align}}} \tag{13.7}$$

where $\lambda_{\text{NTP}} = 1.0$, $\lambda_{\text{MLM}} = 0.5$, and $\lambda_{\text{align}} = 0.3$ are mixing coefficients. The alignment loss $\mathcal{L}_{\text{align}}$ pulls representations of corresponding DNA and protein sequences together in embedding space (see Section 13.12).

Scheduling the Objective Mix

The mixing coefficients λ are not fixed throughout training. In the early stages of curriculum learning (Section 13.7), λ_{MLM} is increased to 0.8 because MLM provides stronger learning signal from short, simple sequences. As training progresses to longer sequences in later curriculum stages, λ_{NTP} dominates because next-token prediction is better suited for learning long-range dependencies. The alignment loss is activated only after both MLM and NTP losses have stabilized, typically at the transition to Stage 3 of the curriculum.

13.7 Curriculum Learning for Genomic Data

Curriculum learning presents training examples in order of increasing complexity, allowing the model to build foundational representations before encountering difficult patterns [Bengio et al., 2009]. For genomic data, this strategy is particularly important because the complexity of biological sequences spans many orders of magnitude.

13.7.1 What Is Curriculum Learning?

In standard training, examples are sampled randomly from the dataset. In curriculum learning, examples are presented in a structured order: easy examples first, hard examples later. The intuition is the same as in education: students learn arithmetic before calculus, and letters before essays.

For genomic data, "easy" and "hard" have precise definitions:

- **Easy:** Short sequences, simple composition (uniform GC content), single-gene coding regions, well-characterized organisms.
- **Hard:** Long sequences, complex composition (repeat-rich regions, GC-biased genomes), multi-gene regions with overlapping reading frames, metagenomic samples from diverse organisms.

13.7.2 Why Curriculum Learning Helps

Training on random genomic data from the start creates several problems:

1. **Gradient signal confusion.** Early in training, the model cannot distinguish coding from non-coding DNA. Random sampling mixes both, providing inconsistent gradient signals.
2. **Wasted capacity on noise.** Large genomic datasets contain repetitive elements, sequencing artifacts, and low-complexity regions. Training on these first wastes model updates.
3. **Failure to learn hierarchical structure.** Genomic organization is hierarchical: nucleotides form codons, codons form genes, genes form operons, operons form chromosomes. Random sampling jumbles these levels.

Curriculum learning addresses all three problems by ensuring the model first masters the nucleotide alphabet, then codon patterns, then gene-level structure, and finally genome-level organization.

13.7.3 DOGMA's Four-Stage Curriculum

Genomic Curriculum Stages

The genomic curriculum progresses through four stages:

1. **Stage 1: Synthetic sequences** (∼5% of training). Random DNA with controlled GC content and dinucleotide frequencies. The model learns basic nucleotide statistics and the four-letter alphabet.
2. **Stage 2: Single-gene sequences** (∼20% of training). Complete coding sequences from well-annotated organisms. The model learns codon usage, start/stop codon placement, and open reading frame structure.
3. **Stage 3: Multi-gene regions** (∼35% of training). Genomic regions containing multiple genes with intergenic sequences, promoters, and regulatory elements. The model learns gene organization and regulatory grammar.
4. **Stage 4: Full chromosomal segments** (∼40% of training). Long-range genomic context including repeat elements, structural features, and complex gene clusters. The model learns large-scale genomic organization.

Curriculum Learning Mirrors Development

The four-stage curriculum mirrors the progressive complexity of biological education: students first learn the alphabet (nucleotides), then words (codons), then sentences (genes), then paragraphs (genomic regions). It also mirrors embryonic development, where simple gene expression programs activate before complex regulatory cascades. In both cases, establishing basic competence before introducing complexity leads to more robust learning than exposure to full complexity from the start.

13.7.4 Stage Transitions and Mixing

Transitions between stages are not sharp boundaries. DOGMA uses a mixing schedule where data from completed stages continues to appear (at reduced frequency) in later stages:

$$P(\text{sample from Stage } s \text{ at training step } t) = w_s(t) = \frac{\alpha_s \cdot \mathbb{1}[t \geq t_s^{\text{start}}]}{\sum_{s'} \alpha_{s'} \cdot \mathbb{1}[t \geq t_{s'}^{\text{start}}]}, \tag{13.8}$$

where t_s^{start} is the step at which Stage s is introduced and α_s are stage-specific weights that increase with stage number. This ensures that the model never "forgets" earlier patterns while focusing on increasingly complex data.

Curriculum Stage Triggers

Stage transitions are triggered by loss thresholds, not fixed step counts. Stage 2 begins when the Stage 1 validation loss drops below 1.6 bits per nucleotide (indicating mastery of basic nucleotide statistics). Stage 3 begins when the Stage 2 coding sequence accuracy exceeds 75% (indicating competent codon prediction). Stage 4 begins when the Stage 3 gene boundary detection F1 exceeds 0.6. These adaptive triggers ensure that the model has genuinely learned each level before progressing, rather than spending a fixed number of steps regardless of learning speed.

13.8 Training Data Pipeline: GenBank to Batching

Building a genomic foundation model requires a robust data pipeline that transforms raw database entries into tokenized, curriculum-ordered training batches. This section traces the full pipeline.

13.8.1 Data Sources

The primary data source is **GenBank**, the NIH genetic sequence database. GenBank contains over 200 million sequences from more than 500,000 organisms. DOGMA's training pipeline also draws from:

- **RefSeq:** Curated reference sequences with high-quality annotations.
- **UniProt:** Protein sequences with functional annotations (for cross-modal training).
- **PDB:** Protein structures for 3D structural features.
- **Ensembl:** Genome assemblies with gene annotations, exon/intron boundaries, and regulatory element predictions.

13.8.2 Preprocessing Steps

Raw GenBank entries undergo several preprocessing steps:

1. **Format conversion.** GenBank flat files are converted to FASTA format, preserving key annotations in the header.
2. **Quality filtering.** Sequences with more than 10% ambiguous characters (`N`) are removed. Sequences shorter than 100 nucleotides are discarded.
3. **Deduplication.** Exact duplicates and near-duplicates (>95% sequence identity) are removed using MinHash-based clustering. This prevents the model from memorizing specific sequences.
4. **Taxonomy annotation.** Each sequence is tagged with its taxonomic lineage (kingdom, phylum, class, order, family, genus, species) for curriculum stratification.
5. **Coding region annotation.** Open reading frames (ORFs) are identified using existing GenBank annotations or *ab initio* gene prediction, and reading frames are marked for codon-aware tokenization.

13.8.3 Structured Rendering

After preprocessing, sequences are rendered into structured training formats:

Structured Rendering Formats

Three rendering formats transform raw sequences into structured training examples:

1. **Codon triplet rendering.** Nucleotides are grouped into space-separated triplets aligned to the reading frame: `ATG CGA TCG ATC ...`. This makes codon boundaries visible as whitespace tokens.
2. **k-mer window rendering.** Sequences are presented as overlapping k-mer windows with explicit position annotations: `[pos=0] ATGCGA [pos=1] TGCGAT ...`. This format trains the model to recognize subsequence patterns at multiple offsets.
3. **Annotated rendering.** Sequences are interleaved with biological annotations: coding regions, exon/intron boundaries, known motifs, and functional elements. This format provides dense supervision for learning sequence-function relationships.

13.8.4 Curriculum Assignment and Batching

Each preprocessed sequence is assigned to a curriculum stage based on its complexity:

- **Stage 1:** Synthetically generated. Parameters sampled from organism-specific GC content distributions.
- **Stage 2:** Single CDS (coding sequence) entries from RefSeq with "complete CDS" annotation.
- **Stage 3:** Genomic contigs containing 2–10 annotated genes with intergenic regions.
- **Stage 4:** Chromosomal segments of 50kb–500kb containing complex gene clusters.

Batches are constructed with curriculum-aware sampling (Equation 13.8), and each batch contains sequences from a single stage to avoid length-padding waste.

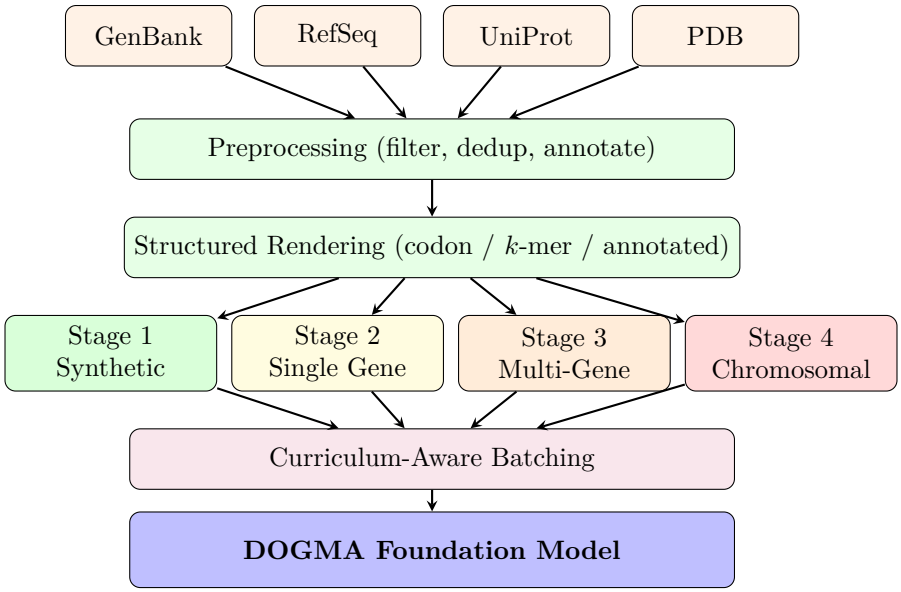


Figure 13.2: The DOGMA training data pipeline. Raw sequences from multiple databases are preprocessed, rendered into structured formats, assigned to curriculum stages, and batched for training. The curriculum-aware sampler controls the mix of stages seen by the model at each training step.

13.9 TinyGPT: Minimal Transformer for Genomic Sequences

With the data pipeline and objectives established, we now construct **TinyGPT**—a minimal decoder-only transformer adapted for byte-level genomic sequence modeling. TinyGPT is not a production model; it is a pedagogical tool for understanding how transformer architecture interacts with genomic data.

13.9.1 Decoder-Only Architecture Adapted for DNA

TinyGPT follows the standard GPT architecture [Radford et al., 2019] with modifications for genomic data:

TinyGPT Architecture

TinyGPT is a decoder-only transformer with the following specification:

Parameter	Value
Vocabulary size	256 (byte-level)
Embedding dimension d	128
Number of layers L	4
Attention heads H	4
Head dimension	$d/H = 32$
Context length T	512
Feed-forward dimension	$4d = 512$
Total parameters	$\sim$1.2M

The model uses byte-level embedding, sinusoidal positional encoding, causal masking, and layer normalization.

The design is deliberately minimal: 1.2 million parameters, trainable on a single GPU in minutes. The goal is not state-of-the-art performance but pedagogical clarity—a model small enough to inspect, debug, and understand completely.

13.9.2 Byte-Level Embedding with Positional Encoding

The input embedding combines a learned byte embedding with sinusoidal positional encoding:

$$\mathbf{h}_0^{(t)} = \mathbf{E}[x_t] + \mathbf{PE}(t), \tag{13.9}$$

where $\mathbf{E} \in \mathbb{R}^{256 \times d}$ is the byte embedding matrix and $\mathbf{PE}(t) \in \mathbb{R}^d$ is the sinusoidal positional encoding at position t. For genomic sequences, the positional encoding carries biological significance: position within a codon (modulo 3), distance from sequence start, and relative position within a gene all affect nucleotide identity.

13.9.3 Training on Mixed Text/Genome Corpora

TinyGPT is trained on a mixed corpus containing both natural language (FASTA headers, gene annotations, biological descriptions) and raw nucleotide sequences. The training objective is standard next-byte prediction:

$$\mathcal{L}_{\text{TinyGPT}} = -\frac{1}{T}\sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t}). \tag{13.10}$$

Mixed-Corpus Training Dynamics

When training on mixed text/genome data, the loss curve exhibits a characteristic two-phase pattern. In the first phase (typically the first 10–20% of training), the model rapidly learns the byte distribution of each modality—distinguishing the four-nucleotide DNA alphabet from the broader ASCII range of English text. In the second phase, the model begins capturing within-modality structure: codon usage patterns in DNA, syntactic patterns in text, and crucially, the *cross-modal* correlations between FASTA headers and their associated sequences. Monitoring per-modality loss separately (by tagging each training example with its source type) reveals whether the model is balancing both domains or overfitting to one.

13.10 Helix-Aware Training

TinyGPT treats genomic sequences as flat byte streams, ignoring a fundamental property of DNA: it is a *double-stranded helix*. The two strands are complementary and antiparallel, meaning the same genetic information can be read in two directions. Helix-aware training exploits this structure.

13.10.1 Why the Double Strand Matters for Training

In a standard language model, a sentence is read left-to-right. DNA is different: the same region can be read on either strand, in opposite directions, and both readings carry biological information. Genes on the forward strand are transcribed left-to-right; genes on the reverse strand are transcribed right-to-left. A model that sees only the forward strand is blind to half the genome's information.

This has concrete implications for training:

1. **Data augmentation.** Every DNA training example can be augmented with its reverse complement, effectively doubling the training data without introducing new sequences.
2. **Representation symmetry.** A well-trained model should produce similar representations for a sequence and its reverse complement, since they encode the same biological information.
3. **Regulatory signal detection.** Many regulatory elements (transcription factor binding sites, enhancers) are functional on both strands. A helix-aware model can detect these regardless of strand orientation.

13.10.2 Dual-Stream Architecture

The dual-stream architecture processes each input through two parallel pathways (cf. the HelixHash embedding architecture in Section 8.7 of Chapter 8):

$$\mathbf{h}_{\text{std}}^{(\ell)} = \text{TransformerBlock}_{\text{std}}^{(\ell)}(\mathbf{h}_{\text{std}}^{(\ell-1)}), \quad \mathbf{h}_{\text{helix}}^{(\ell)} = \text{TransformerBlock}_{\text{helix}}^{(\ell)}(\mathbf{h}_{\text{helix}}^{(\ell-1)}), \tag{13.11}$$

where the standard stream receives byte embeddings (Equation 13.9) and the helix stream receives HelixHash motif embeddings (Equation 8.26 from Chapter 8). At designated fusion layers $\ell \in \mathcal{F}$, the streams exchange information through cross-attention:

$$\mathbf{h}_{\text{fused}}^{(\ell)} = \mathbf{h}_{\text{std}}^{(\ell)} + \alpha_{\text{cross}} \cdot \text{CrossAttn}(\mathbf{h}_{\text{std}}^{(\ell)}, \mathbf{h}_{\text{helix}}^{(\ell)}), \tag{13.12}$$

with a learnable gating coefficient $\alpha_{\text{cross}} \in [0, 1]$ that controls how much structural information flows into the standard stream.

13.10.3 Reverse Complement Consistency Loss

To ensure the model learns strand-symmetric representations, DOGMA adds a *reverse complement consistency loss*:

$$\mathcal{L}_{\text{RC}} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{z}(x_i) - \mathbf{z}(\overline{x_i})\|_2^2, \tag{13.13}$$

where x_i is a DNA sequence, $\overline{x_i}$ is its reverse complement, and $\mathbf{z}(\cdot)$ is the model's pooled representation. This loss penalizes the model for producing different embeddings for complementary sequences, encouraging biologically consistent representations.

Helix-Awareness Is Not Just Augmentation

Reverse complement augmentation (simply training on both strands) doubles the data but does not guarantee symmetric representations. The RC consistency loss (Equation 13.13) goes further: it explicitly constrains the representation space so that complementary sequences are close. Combined with the dual-stream architecture, this creates a model whose internal representations respect the physical structure of DNA.

13.10.4 Motif-Level Feature Channels

The helix stream operates at the motif level rather than the byte level. Each motif (a width-k window of consecutive embeddings) is projected into a dense feature vector that encodes local structural patterns:

$$\mathbf{m}_i = \text{ReLU}(W_m \cdot \text{concat}(\mathbf{h}_i, \mathbf{h}_{i+1}, \ldots, \mathbf{h}_{i+k-1}) + \mathbf{b}_m), \tag{13.14}$$

where $W_m \in \mathbb{R}^{d\times(k\cdot d)}$ is the motif projection matrix. The motif features capture patterns invisible to byte-level processing: codon boundaries, splice site signals, transcription factor binding motifs, and the dinucleotide patterns that determine DNA bendability.

13.10.5 Performance Comparison: Standard vs. Helix-Aware

Table 13.2: Next-byte prediction perplexity on genomic evaluation sets. Lower is better.

Model	Coding DNA	Non-coding DNA	Mixed Corpus
Bigram baseline	3.72	3.81	5.94
TinyGPT (byte-level)	2.41	3.12	3.67
TinyGPT + HelixHash (dual-stream)	2.08	2.74	3.31
TinyGPT + HelixHash + codon-aware	1.89	2.71	3.24
TinyGPT + HelixHash + RC loss	**1.82**	**2.63**	**3.18**

Where Helix-Awareness Helps Most

The performance gap between standard and helix-aware models is largest on coding DNA (0.59 perplexity reduction from baseline TinyGPT to full helix-aware) and substantial on non-coding DNA (0.49 reduction). The addition of codon-aware tokenization improves coding DNA most, while the RC consistency loss benefits all categories. This pattern makes biological sense: coding sequences have strong codon-level periodicity that motif features capture directly, while the RC loss helps everywhere because both coding and non-coding regions contain strand-symmetric regulatory signals.

13.11 Multi-Scale TetraMemory During Pretraining

Large genomic sequences exceed the context window of many practical models. A 500kb chromosomal segment contains 500,000 nucleotides—far beyond the 256-byte context of the measured DOGMA baseline. Multi-scale TetraMemory is a proposed mechanism for retaining selected information beyond the immediate window; its benefit must be established through length-extrapolation ablations.

13.11.1 What Is TetraMemory?

TetraMemory (introduced in Chapter 10) is a structured external memory organized into four hierarchical scales:

TetraMemory Scales

TetraMemory maintains four memory banks operating at different granularities:

1. **Nucleotide scale** (τ_1): Stores per-position information (e.g., nucleotide identity, local GC content). Resolution: 1 nucleotide. Capacity: last 2,048 positions.
2. **Codon scale** (τ_2): Stores per-codon information (e.g., amino acid identity, codon usage frequency). Resolution: 3 nucleotides. Capacity: last 4,096 codons (12,288 nt).
3. **Gene scale** (τ_3): Stores per-gene summaries (e.g., gene function embedding, expression context). Resolution: $\sim$1,000 nucleotides. Capacity: last 64 genes.
4. **Chromosome scale** (τ_4): Stores regional summaries (e.g., GC content, repeat density, gene density). Resolution: $\sim$50,000 nucleotides. Capacity: entire chromosome.

13.11.2 How TetraMemory Integrates with Training

During pretraining, TetraMemory operates as a sliding context extension. As the model processes a long sequence in chunks:

1. Each chunk is processed normally through the transformer.
2. At the end of each chunk, summary representations are written to the appropriate TetraMemory scales.
3. At the start of the next chunk, the model reads from TetraMemory to initialize its context with information from previous chunks.

4. The reading is done via cross-attention between the current chunk's representations and the TetraMemory contents.

$$\mathbf{h}_{\text{ctx}}^{(t)} = \mathbf{h}^{(t)} + \sum_{s=1}^{4} \gamma_s \cdot \text{CrossAttn}(\mathbf{h}^{(t)}, \tau_s), \tag{13.15}$$

where γ_s are learned scale-specific attention weights and τ_s denotes the contents of memory scale s.

TetraMemory Write Policy

Not all hidden states are written to TetraMemory—this would be prohibitively expensive. DOGMA uses a *salience-gated* write policy: a small network predicts a "write probability" for each position based on its hidden state. Positions with high salience (typically those corresponding to gene boundaries, regulatory elements, or unusual sequence composition) are preferentially written. During Stage 1 of curriculum learning, all positions are written (since sequences are short). During Stage 4, only ∼5% of positions are written, focusing memory on structurally important locations.

13.11.3 Multi-Scale Attention Patterns

Different TetraMemory scales capture different types of long-range dependency:

Table 13.3: TetraMemory scales and the biological patterns they capture.

Scale	Resolution	Capacity	Biological Patterns Captured
τ_1 (nucleotide)	1 nt	2,048 nt	Local motifs, splice sites, codon context
τ_2 (codon)	3 nt	12,288 nt	Codon usage bias, ORF structure, translation signals
τ_3 (gene)	∼1 kb	∼64 kb	Gene clusters, operon structure, co-regulation
τ_4 (chromosome)	∼50 kb	whole chr	Isochores, replication origins, centromere proximity

Biology Operates at Multiple Scales

The four TetraMemory scales mirror the hierarchical organization of genomes. Individual nucleotides matter for splice site recognition (scale τ_1). Codons matter for translation efficiency (scale τ_2). Gene neighborhoods matter for co-regulation and operon structure (scale τ_3). Chromosomal regions matter for replication timing and chromatin state (scale τ_4). A foundation model that operates at only one scale will miss patterns at the others. TetraMemory allows the model to simultaneously attend to information at all four biological scales.

13.12 Cross-Modal Training

A genomic foundation model that processes only DNA sequences misses the rich web of connections between biological modalities. Cross-modal training integrates text, DNA, protein sequences, and 3D structural data into a unified representation space.

13.12.1 The Four Modalities

The cross-modal training framework operates over four biological modalities:

1. **Natural language text.** Gene descriptions, functional annotations, literature abstracts, and FASTA headers. Provides semantic context for biological entities.
2. **DNA sequences.** Nucleotide sequences at the genomic, transcript, and coding levels. The primary modality of genomic foundation models.
3. **Protein sequences.** Amino acid sequences derived from DNA through the genetic code. Proteins are the functional products of genes; their sequences constrain and predict DNA sequences.
4. **3D structural data.** Protein structures and chromatin conformations, typically represented as distance matrices or coordinate sets. Structure determines function and provides geometric constraints not visible in sequence alone.

13.12.2 Cross-Modal Training Objectives

The cross-modal training loss combines next-token prediction with modality alignment objectives:

$$\boxed{\mathcal{L}_{\text{cross}} = \mathcal{L}_{\text{NTP}} + \lambda_{\text{align}}\mathcal{L}_{\text{align}} + \lambda_{\text{trans}}\mathcal{L}_{\text{trans}}} \tag{13.16}$$

where:

- $\mathcal{L}_{\text{NTP}}$ is the standard next-token prediction loss across all modalities.
- $\mathcal{L}_{\text{align}}$ is a contrastive alignment loss that pulls representations of corresponding entities (e.g., a gene's DNA sequence and its protein product) closer in embedding space while pushing unrelated entities apart:

$$\mathcal{L}_{\text{align}} = -\log \frac{\exp(\text{sim}(\mathbf{z}_{\text{DNA}}, \mathbf{z}_{\text{prot}})/\tau)}{\sum_j \exp(\text{sim}(\mathbf{z}_{\text{DNA}}, \mathbf{z}_{\text{prot}}^{(j)})/\tau)}, \tag{13.17}$$

  where τ is a temperature parameter and the sum is over negative samples.
- $\mathcal{L}_{\text{trans}}$ is a translation loss that trains the model to generate one modality from another (e.g., predict the protein sequence given the DNA coding sequence), mirroring the biological process of translation.

From Central Dogma to Foundation Model

The cross-modal design maps parts of the central dogma to conditional prediction tasks. A translation loss can encode the genetic code, while an alignment loss can reward agreement among sequence, structure, and annotation. Passing either objective would not prove that a model understands the biological process; memorization and shortcut learning remain alternative explanations that require targeted controls.

13.13 Evaluation of Genomic Foundation Models

How do we know if a genomic foundation model is any good? This section presents the benchmarks, metrics, and comparison methodology used to evaluate DOGMA's genomic models.

13.13.1 Intrinsic Metrics

Intrinsic metrics measure the model's ability to model genomic sequences without reference to downstream tasks:

- **Perplexity** (PPL): The exponential of average negative log-likelihood. Lower is better. A uniform four-base predictor has PPL 4.0, but comparisons require identical splits and tokenization.
- **Bits per nucleotide** (BPN): Cross-entropy in bits, related by BPN $= \log_2(\text{PPL})$. A uniform four-base predictor uses 2.0 bits.
- **Codon prediction accuracy**: For coding sequences, the fraction of codons correctly predicted under a declared reading frame. Dataset bias and codon usage make 1/64 an incomplete practical baseline.

13.13.2 Downstream Benchmarks

Downstream benchmarks evaluate the model's learned representations on biologically meaningful tasks:

Table 13.4: Downstream evaluation benchmarks for genomic foundation models.

Benchmark	Task	Metric	Description
Promoter Detection	Classification	AUROC	Distinguish promoter from non-promoter regions
Splice Site Prediction	Classification	F1	Identify exon-intron boundaries
Gene Expression	Regression	Spearman ρ	Predict mRNA expression levels from sequence
Variant Effect	Ranking	NDCG	Rank pathogenicity of single-nucleotide variants
Species Classification	Classification	Accuracy	Identify species from short DNA fragments
Protein Function	Multi-label	Macro-F1	Predict Gene Ontology terms from DNA

13.13.3 Required Comparison with Existing Models

The earlier edition placed numerical values in Table 13.5 without a reproducible checkpoint, dataset split, or evaluation artifact. Those values are withdrawn. The table now defines the comparisons a future DOGMA result must run:

Table 13.5: Required matched comparisons. Blank result cells are intentional until reproducible evidence is available.

Model family	Matching rule	Required tasks	Status
Convolutional	Equal parameters and bytes	Motif, promoter, splice	Not yet run under the frozen suite
GRU/LSTM	Equal parameters and bytes	Recall, promoter, species	Not yet run under the frozen suite
Transformer	Equal parameters, bytes, and context	All frozen tasks	Not yet run under the frozen suite
HyenaDNA/Caduceus	Published and matched small variants	Long-range genomic tasks	Protocol pending
DNABERT-2	Published checkpoint	Standard genomic benchmark suite	Protocol pending
DOGMA	Exact selected checkpoint	Same splits, seeds, and metrics	Measured small baseline only; no downstream result yet

No Efficiency Claim Yet

Biological priors may improve sample efficiency, harm it, or have no effect. Parameter count alone does not measure training compute or quality. DOGMA can claim efficiency only after matched data, optimization, context, hardware, and task evaluations with uncertainty.

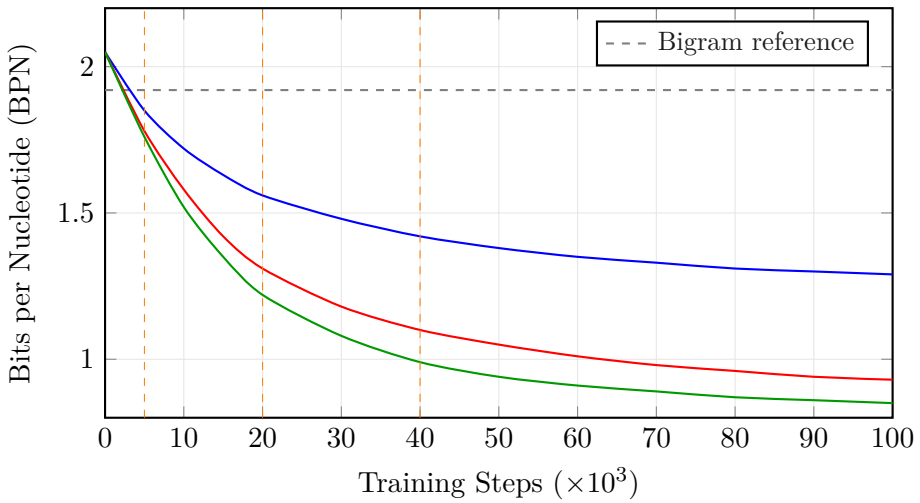


Figure 13.3: Evaluation-plan scaffold. Dashed vertical lines show proposed curriculum transitions; the horizontal bigram value is illustrative. Measured model curves were removed because the earlier figure did not identify reproducible artifacts.

13.14 Proposed Large-Scale DOGMA Configuration

This section preserves a proposed large-scale architecture for future experiments. It is not the model measured in Chapter 14. The parameter values and transfer schedule are hypotheses to test, not a completed training report.

13.14.1 From Foundation Model to DOGMA Pipeline

A future larger DOGMA model could use a pretrained genomic model as its core, then add a staged DOGMA pipeline:

1. **CodonForge Embedding** (Stage 1): Uses the same CodonForge tokenizer described in Section 13.3, initialized with the pretrained byte and codon embeddings.
2. **Regulation Engine** (Stage 2): Context-dependent activation of DNAComputeBlocks, trained via the curriculum learning described in Section 13.7.
3. **Synthesis Controller** (Stage 3): Evolutionary modification of prompt genomes, using the evolutionary training loop of Chapter 12.
4. **DNAComputeBlocks** (Stages 4–5): Non-transformer local and recurrent computation initialized from a compatible pretrained model.
5. **Realization Head** (Stage 6): Output projection, initialized from the pretrained language model head.

13.14.2 Training Configuration Alignment

The proposed configuration would inherit several decisions from the genomic foundation model:

Table 13.6: Hypothetical transfer configuration for a future large DOGMA study.

Parameter	Foundation	Large proposal	Rationale
Vocabulary size	328	328	CodonForge unified vocabulary
Context length	512	2,048	Extended via TetraMemory
Batch size	256	128	Reduced for 4-path compute
Learning rate	3×10^{-4}	1×10^{-4}	Lower for fine-tuning stability
Curriculum stages	4	4	Same staging, different durations
RC consistency	$\lambda = 0.1$	$\lambda = 0.05$	Reduced weight at scale
Objective mix	MLM + NTP	NTP dominant	Decoder-only at scale

13.14.3 Transfer Learning Strategy

The design study proposes a staged transfer:

1. **Phase 1: Frozen transfer** (first 10% of training). Freeze pretrained weights and train only new compatible modules.

2. **Phase 2: Gradual unfreezing** (next 30%). Pretrained layers are unfrozen from top to bottom, one layer per 2,000 steps. The learning rate for unfrozen pretrained layers is 10× lower than for new modules.

3. **Phase 3: Full fine-tuning** (remaining 60%). All parameters are trained jointly with a uniform (low) learning rate.

Staged Transfer Hypothesis

Randomly initialized modules can destabilize pretrained representations. Staged transfer is therefore a reasonable experimental condition, but the earlier 8–12% and 3–5% improvement claims are withdrawn: no qualifying artifacts were identified. A future study must compare staged transfer, direct fine-tuning, and training from initialization under matched compute.

13.15 DOGMA for Multimodal Intelligence

The cross-modal framework of Section 13.12 motivates a design study spanning text, DNA, protein, and structural modalities. This section proposes how a Central Dogma pipeline might process those inputs. It does not report demonstrated cross-modal generation.

Throughout this section, dimensions and paths are hypothetical design parameters. Attention-based paths belong to a historical variant and are excluded from canonical DOGMA. TetraMesh and multimodal realization require separate implementation and evaluation evidence.

13.15.1 TetraMesh: Converting 3D Structures to Byte-Level Tokens

The central challenge of multimodal intelligence is *representation*: how do we feed a protein crystal structure, molecular geometry, or 3D point cloud into a byte-sequence model? The design proposes **TetraMesh**, a pipeline intended to convert 3D data into byte-level tokens. Whether it preserves task-relevant geometry is an open empirical question.

The conversion proceeds in three stages:

1. **Point cloud extraction.** The raw input—whether a protein structure from the Protein Data Bank, a small molecule from QM9, or a 3D object mesh—is reduced to a set of 3D coordinates. For images, intensity-weighted importance sampling with depth from grayscale produces a point cloud in $\mathbb{R}^3$. For proteins, alpha-carbon coordinates are extracted from PDB files. For molecules, atomic positions serve directly as point clouds.

2. **Double-helix mesh construction.** The point cloud is fed into `build_double_helix()`, which constructs a DoubleHelixMesh: a pair of linked tetrahedral chains (sense and antisense) that tessellate the 3D space occupied by the input. Each tetrahedron's four vertices are assigned DNA bases (A, C, G, T) based on geometric properties, producing a base sequence that encodes structural information.

3. **Base sequence serialization.** The mesh is serialized as a compact DNA base string by emitting each tetrahedron's four bases in order (sense chain first, then antisense chain). The result is a sequence like `ACGTTAGC...` that can be tokenized by the standard byte-level tokenizer.

The serialized mesh is wrapped in a structured training document:

```
[TETRA_MESH]
modality: protein_structure
source: pdb_1CRN
sense_chain: 28 tetrahedra
bases: A=12 C=8 G=10 T=6
chirality: +0.14
volume: 0.000342
base_sequence: ACGTTAGCACGT...
[/TETRA_MESH]
USER: Describe this protein_structure.
ASSISTANT: Protein 1CRN: Crambin, a small hydrophobic
  protein from Crambe hispanica seeds.
```

Why DNA Bases for 3D Geometry?

Assigning DNA bases to tetrahedron vertices is not arbitrary. Each base (A, C, G, T) corresponds to one of four geometric roles determined by the vertex's position within its tetrahedron: apex (A), centroid-adjacent (C), ground-face (G), or terminal (T). This mapping preserves the complementarity constraint—sense and antisense chains maintain Watson–Crick pairing—while encoding geometric information in a format the model already understands. The chirality statistic measures the balance of left- versus right-handed tetrahedra, capturing structural asymmetry that would be invisible in a flat sequence representation.

13.15.2 Image to DNA Encoding: A Detailed Pipeline

We now present the complete pipeline for converting a raster image into a DOGMA-compatible DNA base sequence, processing it through the Central Dogma, and decoding the output back to an image. This is the most detailed encoding pipeline we describe, and serves as the template for video and audio encoding in subsequent sections.

Step 1: Image to Point Cloud

Given an input image $I \in \mathbb{R}^{H \times W \times 3}$ with RGB channels, we construct a 3D point cloud $\mathcal{P} = \{(x_i, y_i, z_i)\}_{i=1}^{N}$ as follows:

1. **Normalize coordinates.** Each pixel at position (r, c) maps to normalized spatial coordinates $x = c/W$, $y = 1 - r/H$, both in $[0, 1]$.

2. **Compute depth.** The z-coordinate encodes color information. We compute a weighted luminance: $z = 0.299R + 0.587G + 0.114B$, normalized to $[0, 1]$. This projects the RGB color space onto a perceptual brightness axis.

3. **Importance sampling.** Rather than converting all $H \times W$ pixels (which could produce millions of points), we perform importance sampling. Pixels with high gradient magnitude $\|\nabla I\|$ (edges, textures) are sampled with higher probability. Specifically, we compute the sampling weight $w_{r,c} = \|\nabla I_{r,c}\| + \epsilon$ and draw N points (typically $N = 1024$ to 4096) without replacement.

4. **Color encoding.** Each sampled point is augmented with its full RGB values as auxiliary attributes: $\mathcal{P}_{\text{aug}} = \{(x_i, y_i, z_i, R_i, G_i, B_i)\}_{i=1}^{N}$. The RGB values are quantized to 4 bits per channel (16 levels) to control the vocabulary size.

Example 13.4 (A 4×4 Image as Point Cloud)**.** Consider a simple 4×4 grayscale image (values 0–255):

200	180	50	30
210	190	60	40
100	90	150	160
80	70	140	170

After normalization, the pixel at row 0, column 0 maps to $(x, y, z) = (0.0, 1.0, 0.784)$. The pixel at row 3, column 3 maps to $(0.75, 0.25, 0.667)$. Importance sampling concentrates points near the strong gradient between the bright upper-left and dark upper-right regions.

Step 2: Point Cloud to Delaunay Tetrahedralization

The point cloud $\mathcal{P}$ is tessellated into a tetrahedral mesh using **Delaunay tetrahedralization**. Given N points in $\mathbb{R}^3$, the Delaunay criterion produces a mesh $\mathcal{M} = \{T_1, T_2, \ldots, T_M\}$ where each tetrahedron T_j has four vertices, and no point lies inside the circumsphere of any tetrahedron.

The TetraMesh construction enriches this basic Delaunay mesh:

1. **Vertex classification.** Each vertex is classified by its geometric role: apex (highest z), centroid-adjacent (closest to the tetrahedron centroid), ground-face (in the base triangle), or terminal (remaining vertex).
2. **Sense/antisense pairing.** Adjacent tetrahedra sharing a triangular face are paired into sense–antisense doublets, mirroring the double-stranded structure of DNA. This pairing is performed greedily by matching tetrahedra with maximal shared-face area.
3. **Shape fitting.** The mesh is analyzed against eight shape primitives (helix, sphere, cube, torus, random, spiral, grid, shell) and the best-fitting shape is recorded as metadata. The shape primitive biases the traversal order in the next step.

Step 3: Tetrahedral Mesh to DNA Base Sequence

The mesh $\mathcal{M}$ is traversed in a double-helix order to produce a base sequence $\mathbf{s} = (s_1, s_2, \ldots, s_L)$ where each $s_i \in \{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T}\}$.

1. **Traversal ordering.** Starting from the tetrahedron closest to the centroid of $\mathcal{P}$, a spiral traversal visits each tetrahedron exactly once. The traversal follows the sense chain first, then the antisense chain, producing two complementary subsequences.
2. **Base assignment.** For each tetrahedron T_j, the four vertices are assigned bases according to their classification:

$$\text{Apex} \mapsto \texttt{A}, \qquad \text{Centroid-adjacent} \mapsto \texttt{C},$$
$$\text{Ground-face} \mapsto \texttt{G}, \qquad \text{Terminal} \mapsto \texttt{T}.$$

3. **Color encoding in codons.** The quantized RGB values of the four vertices are encoded as additional bases appended after each tetrahedron's structural bases. Each 4-bit color channel maps to one of $\binom{4}{1}^4 = 256$ codon patterns, producing 3 additional bases per vertex (one per color channel), for a total of $4 + 12 = 16$ bases per tetrahedron.
4. **Complementarity enforcement.** The antisense chain is generated by Watson–Crick pairing: $\texttt{A} \leftrightarrow \texttt{T}$, $\texttt{C} \leftrightarrow \texttt{G}$. This doubles the sequence length but provides built-in error detection.

Step 4: Base Sequence Through the Central Dogma Pipeline

The base sequence $\mathbf{s}$ is processed through the full DOGMA pipeline:

$$\boxed{\mathbf{s} \xrightarrow{\text{tokenize}} \Gamma \xrightarrow{\text{regulate}} \Gamma' \xrightarrow{\text{transcribe}} \mathcal{T} \xrightarrow{\text{fold}} \Phi \xrightarrow{\text{realize}} \hat{y}} \tag{13.18}$$

1. **Tokenization.** The base string is tokenized by the byte-level tokenizer into tokens of length up to 128 (the TetraMesh max token limit).

2. **Regulation.** The epigenetic memory (size 256) provides context from previously processed images. Allosteric regulation (top-$k = 16$) selectively activates the most relevant genomic regions for the current image content.

3. **Transcription.** The 16-layer transformer with 512-dimensional embeddings processes the regulated genome through all four DNA Computing paths simultaneously.

4. **Folding.** The transcript is folded into a phenotype representation Φ that captures the semantic content of the image.

5. **Realization.** The phenotype is realized in the requested mode—text (caption), JSON (structured description), or image (reconstructed pixel data).

Step 5: Decoding Back to Image

When the realization mode is set to `image`, the pipeline reverses the encoding:

1. **Phenotype to base sequence.** The phenotype Φ is realized as a DNA base sequence $\hat{\mathbf{s}}$ through the image realization head.

2. **Base sequence to TetraMesh.** The base sequence is parsed back into tetrahedra: every 16 bases reconstruct one tetrahedron (4 structural + 12 color bases), with vertex positions recovered from the structural bases and colors from the color codons.

3. **TetraMesh to point cloud.** The tetrahedron vertices with their decoded RGB values form a colored point cloud $\hat{\mathcal{P}}$.

4. **Point cloud to image.** The point cloud is rasterized onto a $H \times W$ grid. Each pixel is assigned the color of the nearest point, with Gaussian splatting ($\sigma = 1.5$ pixels) for smooth interpolation of gaps.

Figure 13.4 illustrates the complete image-to-DNA-to-image pipeline with example data at each step.

Comparison with Traditional Computer Vision

Table 13.15.2 contrasts the DOGMA tetrahedral encoding with conventional convolutional neural network (CNN) approaches to image processing.

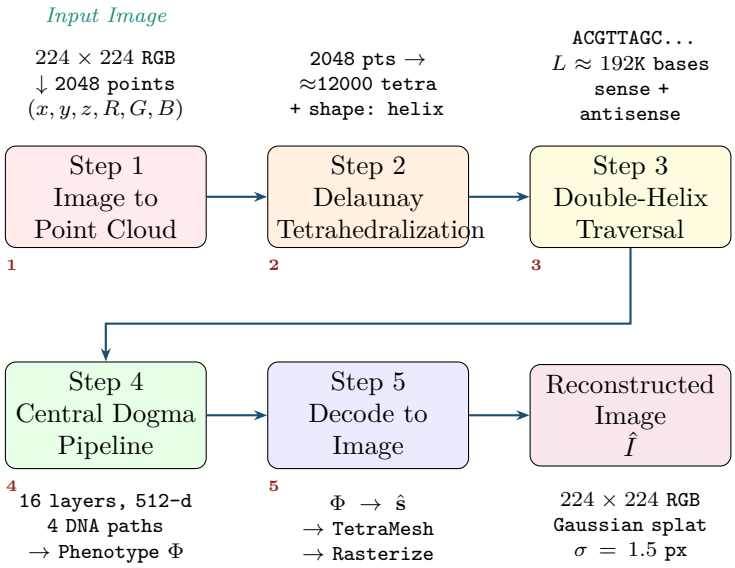


Figure 13.4: The complete image-to-DNA-to-image pipeline. An input image is converted to a point cloud (Step 1), tetrahedralized (Step 2), traversed as a double helix to produce a DNA base sequence (Step 3), processed through the Central Dogma (Step 4), and decoded back to a reconstructed image (Step 5). Example data dimensions are shown for a 224×224 input with 2048 sampled points.

Property	**CNN (Traditional)**	**DOGMA TetraMesh**
Input representation	2D pixel grid ($H \times W \times C$)	3D tetrahedral mesh
Basic operation	2D convolution on regular grid	Traversal of irregular tetrahedra
Rotation handling	Requires data augmentation or special architectures (group convolutions)	Intrinsic rotation invariance—tetrahedral geometry is rotation-equivariant
3D structure	Requires depth estimation or multi-view reconstruction	Native 3D representation from a single image
Scale invariance	Multi-scale via pooling or feature pyramids	Multi-scale via TetraMesh scales $[4, 6, 8]$
Biological grounding	None	DNA base encoding mirrors molecular structure
Cross-modal transfer	Requires separate encoders per modality	Same TetraMesh pipeline for all 3D-representable data

Rotation Invariance from Tetrahedra

A key advantage of DOGMA's tetrahedral encoding is *intrinsic rotation invariance*. When an image is rotated by an arbitrary angle, the point cloud rotates rigidly, and the Delaunay tetrahedralization of the rotated point cloud produces the same set of tetrahedra (up to relabeling). The base assignment depends only on the *relative* positions of vertices within each tetrahedron (apex, centroid-adjacent, ground-face, terminal), which are rotation-invariant properties. By contrast, CNNs operating on pixel grids must learn rotation invariance from data augmentation, which is sample-inefficient and never perfectly general. This is analogous to how DNA encodes protein structure independently of the molecule's orientation in space.

13.15.3 Video to DNA Encoding

Video extends image encoding along the temporal dimension. A video $\mathcal{V} = (I_1, I_2, \ldots, I_F)$ consisting of F frames is encoded as a sequence of linked TetraMesh structures.

Temporal TetraMesh Construction

Each frame I_t is independently converted to a point cloud $\mathcal{P}_t$ and tetrahedralized into a mesh $\mathcal{M}_t$ using the image pipeline of Section 13.15.2. The temporal linking proceeds as follows:

1. **Shared-face linking.** For consecutive frames $\mathcal{M}_t$ and $\mathcal{M}_{t+1}$, boundary tetrahedra (those on the convex hull of each mesh) are matched by proximity. Matched pairs share a virtual triangular face, creating a continuous tetrahedral chain across time. The shared face encodes the optical flow between frames.

2. **Temporal base encoding.** Inter-frame connections are encoded using a special codon pattern: `ATG` (the biological start codon) marks the beginning of each frame, and `TAA` (a stop codon) marks the end. The inter-frame linking bases encode the displacement vectors between matched boundary tetrahedra.

3. **4D point cloud.** Alternatively, the entire video can be treated as a single 4D point cloud $\mathcal{P}_{4D} = \{(x_i, y_i, z_i, t_i)\}$ where $t_i = t/F$ is the normalized frame index. This 4D cloud is tetrahedralized directly, producing a mesh that captures spatiotemporal structure in a single tessellation.

Temporal Attention via Allosteric Regulation

DOGMA's allosteric regulation mechanism (top-$k = 16$) provides a natural mechanism for temporal attention across video frames. The epigenetic memory (size 256) accumulates context from previously processed frames, functioning as a temporal memory buffer:

$$\mathcal{E}_{t+1} = \alpha \cdot \mathcal{E}_t + (1 - \alpha) \cdot \text{TopK}(\Phi_t, k = 16) \tag{13.19}$$

where $\mathcal{E}_t$ is the epigenetic state at frame t, Φ_t is the phenotype of frame t, and α is a learned decay parameter. The top-k selection ensures that only the most salient features from each frame persist in memory, analogous to how biological epigenetic marks selectively regulate gene expression in response to environmental stimuli.

Comparison with Traditional Video Models

Property	Traditional Video Models	DOGMA Video Encoding
Architecture	3D CNNs (C3D, SlowFast) or Video Transformers (ViViT, TimeSformer)	TetraMesh per frame + temporal linking
Temporal modeling	3D convolution kernels or factorized spatiotemporal attention	Allosteric regulation + epigenetic memory decay
Memory	Fixed-size feature maps; no persistent state	Epigenetic memory accumulates across frames
Compute scaling	Cubic in spatiotemporal resolution	Linear in frames (each frame processed independently, linked by shared faces)
Motion capture	Optical flow as auxiliary input or learned implicitly	Inter-frame displacement encoded in linking bases

Video as Gene Expression Over Time

The biological analogy for video processing is compelling. Each frame corresponds to a snapshot of gene expression at a particular developmental stage. The epigenetic memory captures the developmental history—just as a cell's epigenetic state reflects its lineage, the DOGMA video encoder's memory reflects the visual history of the scene. Allosteric regulation acts as temporal attention: distant frames can influence the current frame's processing through the accumulated epigenetic state, much as early developmental signals shape later gene expression patterns.

13.15.4 Audio to DNA Encoding

Audio signals are encoded through a spectrogram-based pipeline that maps frequency–time representations to 3D point clouds and then to TetraMesh structures.

Spectrogram to Point Cloud

Given an audio waveform $a(t)$ sampled at rate f_s, the encoding proceeds:

1. **Short-time Fourier transform.** Compute the spectrogram $S(f, \tau) = |\text{STFT}(a(t))|^2$ with window size 1024 samples and hop length 256 samples. This produces a time–frequency matrix.

2. **Mel scaling.** Apply a Mel filterbank with 128 filters to produce $S_{\text{mel}} \in \mathbb{R}^{128 \times T}$, compressing the frequency axis to match human auditory perception.

3. **Point cloud construction.** Each spectrogram bin (f_i, τ_j) with energy above a threshold θ becomes a 3D point:

$$x = \tau_j / T, \qquad y = f_i / 128, \qquad z = \log(1 + S_{\text{mel}}(f_i, \tau_j)).$$

4. **Importance sampling.** As with images, high-energy and high-gradient regions are oversampled, concentrating points on spectral peaks (formants, onsets, harmonics).

Frequency Bands as Tetrahedral Layers

The TetraMesh for audio has a distinctive layered structure that reflects the physics of sound:

- **Bass layer** ($y < 0.25$, corresponding to 0–2 kHz): Large tetrahedra capturing low-frequency energy. Mapped to `A`-rich base sequences (adenine's double hydrogen bond mirrors the strong, sustained energy of bass frequencies).
- **Mid layer** ($0.25 \leq y < 0.625$, 2–5 kHz): Medium tetrahedra encoding speech formants and melodic content. Balanced base distribution.
- **Treble layer** ($0.625 \leq y < 0.875$, 5–11 kHz): Small, dense tetrahedra capturing transients and harmonics. `G/C`-rich sequences (the triple hydrogen bond of G–C pairs encodes the higher information density of treble frequencies).
- **Ultrasonic layer** ($y \geq 0.875$, above 11 kHz): Sparse tetrahedra for high-frequency detail, present only in high-fidelity audio.

Comparison with Traditional Audio Models

Property	Traditional Audio Models	DOGMA Audio Encoding
Architecture	WaveNet (dilated causal convolutions), Whisper (encoder–decoder transformer)	Spectrogram → TetraMesh → Central Dogma
Input format	Raw waveform or 2D spectrogram	3D point cloud from spectrogram
Frequency modeling	Learned implicitly through convolution layers	Explicit layered structure mapping frequency bands to tetrahedral layers
Time modeling	Causal convolution or autoregressive attention	Temporal axis encoded as x-coordinate in 3D space
Cross-modal	Separate models for speech, music, sound events	Same TetraMesh pipeline as images and video

13.15.5 Cross-Modal Generation

DOGMA's unified genomic representation enables generation across modality boundaries. Because all modalities converge to the same genome representation Γ and diverge only at the realization step, cross-modal generation requires no additional architectural components—only a different `RealizationMode` selection.

Text to Image via DOGMA

Text-to-image generation follows the pipeline:

$$\text{“a red car”} \xrightarrow{\text{tokenize}} \Gamma_{\text{text}} \xrightarrow{\text{Central Dogma}} \Phi \xrightarrow{\text{realize(image)}} \hat{I} \tag{13.20}$$

The text prompt is tokenized into a genome Γ_{text} using the standard CodonForge tokenizer. The Central Dogma pipeline processes this genome through all 16 layers, producing a phenotype

Φ. The image realization head converts Φ into a TetraMesh base sequence, which is then decoded to a point cloud and rasterized. The `multi_express` method allows simultaneous text and image realization from the same phenotype:

$$\Phi \xrightarrow{\text{multi\_express}} \begin{pmatrix} y_{\text{text}} \\ y_{\text{image}} \\ y_{\text{dna}} \end{pmatrix} \tag{13.21}$$

Image to Text via DOGMA

The reverse direction—image captioning—uses the image encoding pipeline of Section 13.15.2 to produce a genome Γ_{image}, processes it through the Central Dogma, and realizes the phenotype in text mode:

$$I \xrightarrow{\text{TetraMesh}} \Gamma_{\text{image}} \xrightarrow{\text{Central Dogma}} \Phi \xrightarrow{\text{realize(text)}} \hat{y}_{\text{caption}} \tag{13.22}$$

Comparison with Diffusion Models

Property	Diffusion Models	DOGMA Cross-Modal
Architecture	Separate text encoder (CLIP) + U-Net/DiT denoiser (Stable Diffusion, DALL-E 3)	Single Central Dogma pipeline for all modalities
Generation process	Iterative denoising over 20–50 steps	Single forward pass through 16-layer transformer
Latent space	Continuous Gaussian latent space	Discrete DNA base sequence (4-letter alphabet)
Cross-modal bridge	CLIP embedding alignment between text and image spaces	Unified genome representation—no alignment needed
Controllability	Classifier-free guidance, ControlNet	Epigenetic regulation and allosteric modulation
Bidirectionality	Requires separate models for text→image and image→text	Same pipeline, different realization modes

Single Forward Pass Generation

The most striking difference between DOGMA and diffusion models is the generation process. Diffusion models require 20–50 iterative denoising steps, each involving a full forward pass through the U-Net. DOGMA generates in a single forward pass through the Central Dogma pipeline. This is possible because the DNA base sequence provides a structured, discrete representation that does not require iterative refinement. The four-letter alphabet constrains the output space far more tightly than a continuous Gaussian latent space, trading some representational flexibility for computational efficiency. The biological analogy is direct: a cell does not iteratively refine a protein—it translates the mRNA in a single pass through the ribosome.

13.15.6 The Multimodal Corpus Builder

A multimodal DOGMA model requires training data spanning multiple modalities. The multimodal data pipeline downloads and converts data from five open sources:

Source	Modality	Default Samples	Description
COCO / Flickr30k	Image–text pairs	1,000	Images converted to point clouds via intensity sampling
RCSB PDB	Protein structures	500	Experimental 3D structures (alpha-carbons)
AlphaFold DB	Predicted structures	500	AI-predicted structures for human, *E. coli*, yeast, mouse
QM9	Small molecules	2,000	134K organic molecules with 3D equilibrium geometries
ModelNet / ShapeNet	3D object meshes	1,000	Point cloud representations of common objects

Every source follows the same pipeline: raw data is converted to a point cloud, passed through TetraMesh, and rendered as a training document with metadata headers and a caption in the `USER:/ASSISTANT:` format. The resulting corpus is split into train/valid/test sets and stored in the `external_corpus_dirs` directory referenced by the multimodal configuration.

Multimodal Configuration

The multimodal training configuration uses the `helix_dogmaforge` model architecture with the following key settings: an external corpus directory pointing to the TetraMesh-converted data, an `external_corpus_weight` of 2.0 (double-weighting multimodal data relative to genomic data), multi-scale TetraMesh at scales [4, 6, 8], and all four computational paths enabled (Strand Displacement, ParallelScan, TetraStochastic, DoubleStrand). The context length of 256 bytes is sufficient for the compact TetraMesh representations, and the `multi_scale_tetra` flag ensures the model processes structural information at multiple granularities.

13.15.7 Multi-Express: One Genome, Many Realizations

The architectural core of DOGMA's multimodal capability is the `CentralDogma.multi_express()` method, which implements a principle with a direct biological analogy: *one gene can produce many different proteins through alternative splicing.* In DOGMA, one genome produces one transcript, one phenotype, but *many* realizations:

$$\Gamma \xrightarrow{\text{regulate}} A \xrightarrow{\text{transcribe}} T \xrightarrow{\text{fold}} \Phi \xrightarrow[\text{realize}_2]{\text{realize}_1} \begin{pmatrix} y_{\text{text}} \\ y_{\text{code}} \\ y_{\text{JSON}} \\ y_{\text{action}} \\ y_{\text{dna}} \\ y_{\text{image}} \end{pmatrix} \tag{13.23}$$

The pipeline executes the expensive steps (regulation, transcription, folding) exactly once. Only the final realization step branches, producing outputs in as many modes as requested. The `PhenotypeRealizer.multi_realize()` method takes a single phenotype and a list of `RealizationMode` values, returning one `Realization` per mode:

- `TEXT`: Natural language text, with system and task prompts rendered as paragraphs.
- `CODE`: Executable code, with each semantic fragment prefixed by a comment header.
- `JSON`: Structured JSON serialization of the phenotype's semantic content.
- `ACTION`: A world-action payload for downstream execution (includes tool calls).
- `DNA`: A raw DNA base sequence for genomic applications.
- `IMAGE`: A TetraMesh-encoded base sequence that decodes to a raster image.

Each realization receives a quality score that measures how well the output captures the underlying phenotype, computed from content length, format validity (e.g., valid JSON for the JSON mode), and the phenotype's folding quality.

The Realization Bottleneck Is at the End, Not the Middle

In conventional multimodal architectures, different modalities require different encoders, different attention patterns, and different loss functions throughout the network. DOGMA inverts this: *all* modalities pass through the same genomic pipeline, and diversity appears only at the very last step. This design has two advantages. First, it amortizes the computational cost of regulation, transcription, and folding across all output modes. Second, it guarantees that all realizations share the same semantic grounding—they differ in *format*, not in *meaning*. A text realization and an image realization of the same phenotype are guaranteed to convey the same semantic content, much as mRNA and protein are different representations of the same gene.

13.15.8 Visual Overviews

This section provides three visual overviews of the DOGMA multimodal system: the full multimodal pipeline, a side-by-side comparison with traditional ML, and a TetraMesh encoding visualization.

Full Multimodal Pipeline

Figure 13.5 illustrates how image, video, and audio data all feed into the Central Dogma through the unified TetraMesh encoding.

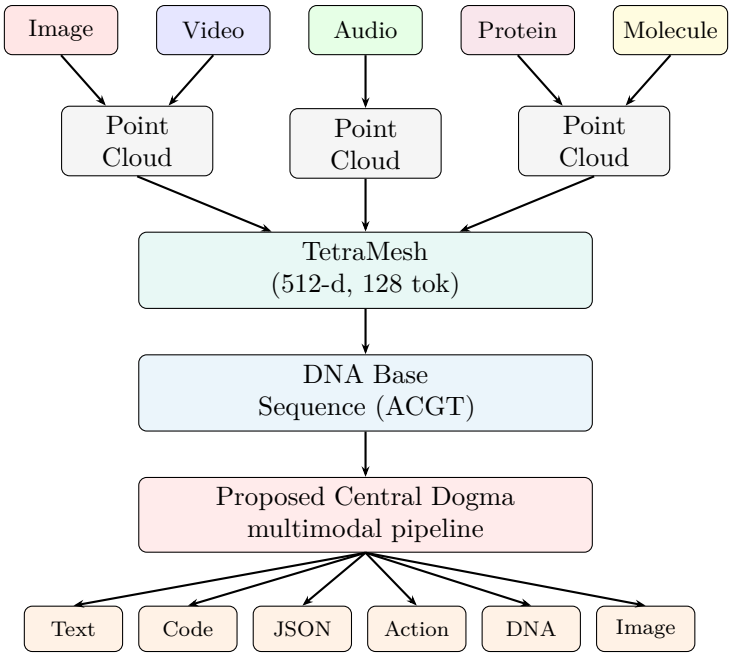


Figure 13.5: Hypothetical DOGMA multimodal pipeline. The diagram is a research specification; it is not an evaluated end-to-end system. Each conversion and realization requires an information-retention task and matched baseline.

Side-by-Side: Traditional ML vs. DOGMA

Figure 13.6 contrasts the conventional multi-encoder approach with DOGMA's unified pipeline.

TetraMesh Encoding Visualization

Figure 13.7 shows a concrete example of how a simple 3D shape is converted to a DNA base sequence.

13.15.9 Use Cases for DOGMA Multimodal Intelligence

The unified encoding pipeline opens several application domains where DOGMA's approach offers distinct advantages over modality-specific architectures.

Medical Imaging

Medical images—X-rays, CT scans, MRI volumes—are naturally 3D or quasi-3D data that benefit from DOGMA's volumetric encoding:

- **CT scan analysis.** A CT volume (e.g., $512 \times 512 \times 128$ voxels) is directly represented as a 3D point cloud where voxel intensity maps to the z-coordinate at the voxel's spatial position. The resulting TetraMesh captures anatomical structures as connected tetrahedral regions, with tissue boundaries appearing as natural mesh interfaces.
- **X-ray encoding.** A 2D X-ray is encoded using the image pipeline of Section 13.15.2. The depth (z) coordinate encodes radiographic density, so dense structures (bone) appear as high-z regions and soft tissue as low-z regions.

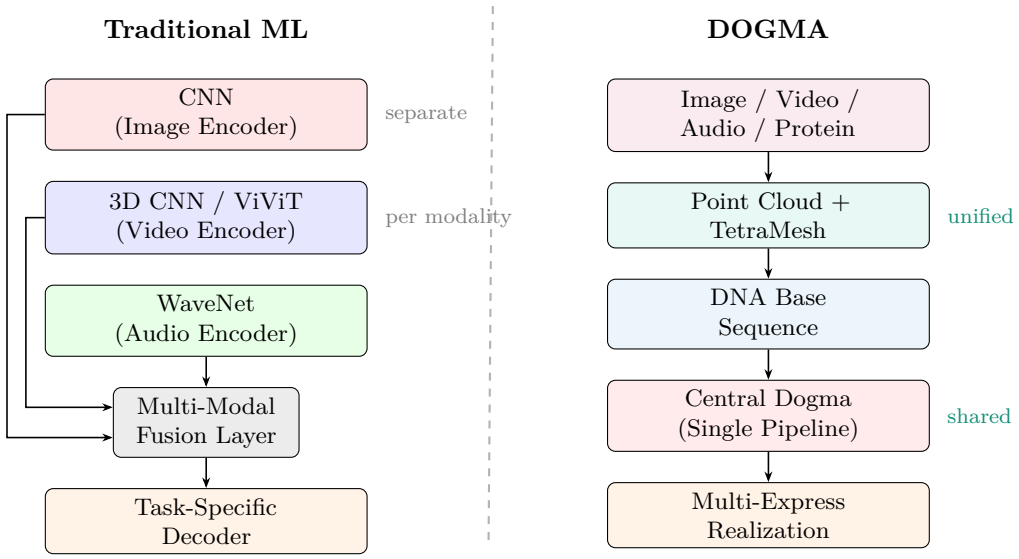


Figure 13.6: Traditional multimodal ML (left) requires separate encoders for each modality and a fusion layer to combine representations. DOGMA (right) uses a single unified pipeline: all modalities are converted to point clouds, tessellated into TetraMesh, encoded as DNA, and processed through one Central Dogma pipeline. The multi-express realization step produces outputs in any requested modality.

- **Diagnostic output.** The Central Dogma pipeline processes the encoded scan and realizes the phenotype in multiple modes simultaneously: a text diagnosis, a JSON-structured findings report, and an annotated image highlighting regions of concern.

Medical Imaging Configuration

For medical imaging applications, the TetraMesh is configured with `shape=shell` to match the layered structure of anatomical cross-sections. The importance sampling threshold is reduced ($\theta = 0.01$) to capture subtle density variations, and the number of sampled points is increased to $N = 8192$ for diagnostic-quality encoding. Epigenetic memory is preloaded with anatomical priors from a curated dataset of 10,000 normal scans.

Protein Structure Prediction

Protein structures are the original domain for TetraMesh encoding, and DOGMA offers a natural bridge between sequence and structure:

- **Structure to genome.** A protein's 3D structure (alpha-carbon coordinates from PDB) is tetrahedralized, producing a DNA base sequence that encodes the spatial fold. This creates a "structural genome" that can be compared with the protein's actual coding sequence.
- **Fold prediction.** Given a protein's amino acid sequence (realized as a DNA coding sequence via the codon table), the Central Dogma pipeline predicts a structural genome. Decoding this structural genome through the TetraMesh reverse pipeline produces predicted 3D coordinates.
- **Mutation analysis.** Point mutations in the input sequence produce corresponding changes in the structural genome, enabling prediction of structural effects of mutations without explicit

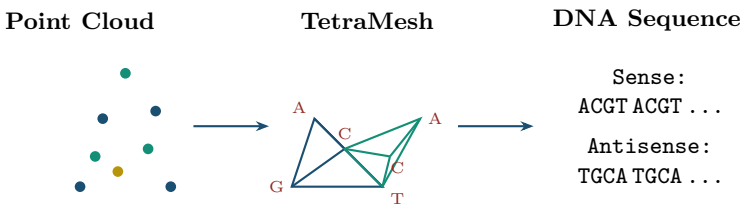


Figure 13.7: TetraMesh encoding visualization. A set of 3D points (left) is tessellated into tetrahedra with vertices labeled by DNA bases—A (apex), C (centroid-adjacent), G (ground-face), T (terminal) (center). The tetrahedra are traversed in double-helix order to produce sense and antisense DNA base sequences (right). The complementary strands maintain Watson–Crick pairing (A–T, C–G) throughout.

molecular dynamics simulation.

Autonomous Driving: Sensor Fusion

Autonomous vehicles integrate data from cameras (images), LiDAR (3D point clouds), radar (range-Doppler maps), and microphones (audio). DOGMA provides a natural sensor fusion framework:

- **Unified scene representation.** Camera images, LiDAR sweeps, and radar returns are each converted to TetraMesh encodings. Because all encodings share the same DNA base format, they can be concatenated into a single genome that represents the complete scene.
- **Temporal fusion.** Sequential sensor readings are linked via the temporal TetraMesh mechanism (Section 13.15.3), with epigenetic memory maintaining a persistent model of the driving environment across time steps.
- **Action realization.** The phenotype is realized in `action` mode to produce steering, acceleration, and braking commands, while simultaneously realizing in `text` mode for explainable decision logging.

Drug Discovery

Small molecule drug candidates are naturally represented as 3D atomic coordinates, making TetraMesh encoding particularly apt:

- **Molecular encoding.** A drug molecule's 3D equilibrium geometry (from quantum chemistry calculations or experimental crystallography) is directly tetrahedralized. Atom types map to base annotations, and bond orders are captured by the tetrahedral connectivity.
- **Property prediction.** The Central Dogma processes the molecular genome and realizes predictions in JSON mode: binding affinity, solubility, toxicity, and metabolic stability as structured key–value pairs.
- **Generative chemistry.** By conditioning the Central Dogma on a target property profile (e.g., high binding affinity to a specific protein target), the DNA realization mode produces a molecular genome that can be decoded back to a 3D structure—a *de novo* drug candidate.

Multimodal Convergence Hypothesis

The proposed architecture uses *convergent encoding, divergent realization*: modalities map to a shared representation Γ, and output heads map a phenotype representation Φ to task formats. The analogy to biological genomes motivates the design but does not establish that one representation is sufficient or information-preserving across modalities.

13.16 DOGMA for Conversational AI

The codebase includes a `ChatSession` interface for testing sustained, multi-turn language generation. An interface is not a capability result. The native measured baseline still produces malformed prose. A separate ATCG-to-language bridge now passes a narrow four-probe evaluation, so this section documents both the mechanism and the strict boundary between those two results.

13.16.1 The ChatSession Architecture

The `ChatSession` class maintains a turn-based conversation with a trained DOGMA model. Its architecture is straightforward: a conversation history list stores alternating user and assistant messages, and each response is generated by constructing a prompt from the full history, tokenizing it as raw UTF-8 bytes, and generating autoregressively with top-k sampling.

The response loop follows five steps:

1. **Append** the user's message to the conversation history.
2. **Enrich** (optionally) the input through the Central Dogma pipeline.
3. **Build** the full prompt from history in the format `USER: ... ASSISTANT: ... USER: ... ASSISTANT:`, truncated from the beginning to fit the model's context window.
4. **Generate** byte by byte, applying temperature scaling and top-k filtering at each step. A streaming callback emits each decoded character in real time.
5. **Stop** when an end-of-sequence token is generated or a turn marker (`USER:`, `--`) appears in the output, preventing the model from hallucinating the next user turn.

Early Stopping on Turn Markers

A common failure mode in conversational language models is *turn bleeding*: the model continues generating past the end of its response and begins producing text for the user's next turn. DOGMA prevents this by maintaining a list of stop markers (`USER:`, `--`, `\nUSER:`) and checking the decoded output after every token. When a stop marker is detected, generation halts immediately and the response is truncated to exclude the marker. This simple mechanism eliminates the need for a separate end-of-turn token in the vocabulary and works reliably even with byte-level tokenization, where turn boundaries do not align with token boundaries.

13.16.2 Central Dogma Context Enrichment

The most distinctive feature of DOGMA's conversational mode is optional **Central Dogma enrichment**, toggled by the `/dogma` command. When enabled, each user message is processed through the full Central Dogma pipeline *before* being passed to the language model:

1. The user's text is encoded into a genomic sequence via `CentralDogma.from_text()`, producing a genome Γ where the original text is encoded as typed genomic bases.
2. The genome is expressed through the full pipeline: regulation $\rightarrow$ transcription $\rightarrow$ folding $\rightarrow$ realization.
3. The execution trace—number of bases in the genome, number of bases expressed, and folding quality—is appended to the user's message as a compact annotation:

```
What is the structure of insulin?
  [DOGMA: bases=156 expressed=89 quality=0.73]
```

This enrichment serves two purposes. First, the trace statistics provide the language model with a *genomic fingerprint* of the query, encoding information about its complexity and structure that is not obvious from the surface text. Second, the regulation step selectively activates genomic regions relevant to the query, potentially priming the model to produce more contextually appropriate responses.

Genomic Fingerprinting Requires an Ablation

Trace statistics such as expressed-base fraction, folding score, and encoded length form a feature vector. They count as useful language features only if a held-out task improves over controls using ordinary length, lexical, and model features. Encoding text as a genome demonstrates representability, not understanding.

13.16.3 Realization Focus for Conversational Output

The conversational configuration introduces a critical corpus-level optimization: `realization_focus`. When this flag is set to `true`, the training corpus is modified to *truncate* DNA sequences and emphasize the semantic output—the natural language content that follows the genomic metadata.

Parameter	Multimodal	Conversational	Effect
`realization_focus`	`false`	`true`	Truncates DNA sequences to emphasize text
`max_dna_display`	–	40	Maximum DNA characters shown in corpus
`external_corpus_weight`	2.0	3.0	Higher weight on external (distilled) data
`temperature`	0.8	0.7	Lower temperature for more focused output
`checkpoint_interval`	1,000	500	More frequent checkpoints for chat tuning
`smoothing`	–	0.1	Label smoothing for conversational robustness

The rationale is straightforward: a conversational model must prioritize fluent, coherent language generation over genomic fidelity. The genomic substrate remains the computational engine, but the model's capacity should be allocated to the realization layer—the part the user actually sees. The `max_dna_display` parameter limits DNA sequences to 40 characters in the training corpus, ensuring the model spends most of its context window on natural language patterns.

The conversational configuration also includes **distilled data** from a separate `datasets/distilled` directory, weighted at 3× the genomic corpus. This distilled data consists of high-quality conversational examples that have been filtered and refined through a teacher model, providing the chat-specific patterns (greetings, question-answering, multi-turn coherence) that raw genomic data cannot supply.

13.16.4 Measured ATCG-to-Language Bridge

The July 2026 experiments introduced an explicit boundary between genomic representation and natural-language realization:

$$q \xrightarrow{C_{\text{boot}}} (g, t) \xrightarrow{R_\phi} \hat{y},$$

where q is the request, C_{boot} is a deterministic bootstrap compiler, $g \in \{A, T, C, G\}^{64}$ is a canonical trace, t is a compact auditable transcript, and R_ϕ is a learned realization model. This decomposition is useful because a failure can be assigned to a stage. It is also scientifically dangerous if the stages are conflated: success by R_ϕ does not prove that a native DOGMA generator produced g.

Bootstrap Compiler

The compiler normalizes the request to printable ASCII, keeps a 16-character semantic plan, pads it to fixed width, and encodes each byte as four bases using

$$00 \mapsto A, \qquad 01 \mapsto C, \qquad 10 \mapsto G, \qquad 11 \mapsto T.$$

It then enforces a start marker `ATGATG` and a terminal marker `TAATAAAT`, yielding exactly 64 canonical bases. The transcript retains up to 28 normalized characters so a human can audit what was passed to the realizer.

This compiler is deterministic. It demonstrates a stable ATCG-first interface, not learned genomic semantics. A learned native compiler must eventually replace it and outperform controls before the genomic bottleneck can be called useful.

Distillation and Realization

The training corpus contains 96 instruction records spanning genomic explanation, scientific reasoning, critique, general language, and DOGMA research contracts. Answers were distilled from `Qwen3-4B-Instruct-2507`. A LoRA adapter on `Qwen3-0.6B` learned the realization contract over four epochs and 48 optimizer updates; recorded training loss fell from about 2.0 to 0.32. The adapter occupies approximately 50 MB.

The teacher and student are transformers. Therefore the measured claim is precise: *a compact transformer adapter can realize useful prose when conditioned on a request and a canonical ATCG trace.* The experiment does not show that non-transformer DOGMA alone has acquired language.

Held-Out Contract Probes

The frozen `dogma-genomic-language-v3` suite contains four transfer probes that are excluded from adapter training. All four passed in the retained evaluation:

Table 13.7: Measured ATCG-to-language bridge probes, July 2026.

Probe	Required behavior	Observed gate
Canonical trace	64 canonical bases followed by an explanation	passed
Central dogma	Name DNA, an RNA intermediate, and protein flow	passed
Self-critique	State a limitation and a falsifiable test	passed
General language	Distinguish a hypothesis from observed evidence	passed

The evaluator also required four distinct answers, printable output, no replacement characters, and the genome to precede the transcript and answer. This is stronger than checking a regular expression, but still only a smoke suite.

What the Bridge Does Not Establish

Four probes are not a language benchmark. The deterministic trace may be ignored by the realizer, the 96-example corpus is small, and no matched request-only control has yet measured the trace's causal contribution. The next required experiment removes or shuffles g and t while keeping the request, model, seed, and compute fixed. DOGMA earns a representational claim only if the intact trace improves held-out performance.

13.16.5 Streaming Generation

Real-time streaming is essential for conversational AI. DOGMA implements streaming through an incremental decode loop: after each token is sampled from the model, the full sequence of generated tokens is decoded to UTF-8, and any newly decoded characters are emitted through a callback function. This approach handles the complication that UTF-8 is a variable-width encoding—a single character may span multiple bytes, and a byte token may not correspond to a complete character until subsequent bytes arrive.

The streaming loop also handles inference optimizations:

- **Float16 on CUDA**: The model is converted to half-precision for approximately 2× speedup on GPU.
- **torch.compile**: On PyTorch 2.0+ with CUDA, the model is compiled for fused kernel execution.
- **Context truncation**: The prompt is truncated to `context_length - 32` bytes, reserving space for generation without exceeding the model's positional encoding range.

Key Takeaways

The conversational interface combines realization-focused sampling, optional Central Dogma trace features, and distilled conversational data. The current evidence establishes that this path runs, not that it produces strong dialogue. Evaluation must separately measure UTF-8 validity, instruction following, biological correctness, context retention, latency, and the incremental value of enrichment.

13.17 Exercises

13.1. [**Fundamentals: FASTA parsing**] Write a function that reads a FASTA file and returns a list of (header, sequence) pairs. Handle multi-line sequences correctly. Test it on a file containing three records with sequences of different lengths. What edge cases must your parser handle?

13.2. [**Fundamentals: Tokenization**] Apply BPE tokenization (with a vocabulary of 1,000 tokens trained on English text) to the DNA sequence `ATGATGATGATGATGATG`. How many tokens result? Now apply byte-level tokenization to the same sequence. Compare the two representations in terms of: (a) number of tokens, (b) preservation of codon boundaries, and (c) consistency across different occurrences of the same codon.

13.3. [**Analysis: Bigrams**] Given the bigram transition matrix for a genomic corpus where $P(\texttt{G} \mid \texttt{C}) = 0.18$ while the marginal $P(\texttt{G}) = 0.25$, compute the CpG suppression ratio $P(\texttt{CG})/(P(\texttt{C}) \cdot P(\texttt{G}))$. What biological mechanism explains a ratio less than 1.0? How would this ratio differ between vertebrate and bacterial genomes?

13.4. [**Analysis: Tokenization comparison**] Tokenize the sequence `ATGAAAGCCTTTGGA` using (a) byte-level, (b) 3-mer non-overlapping (codon), and (c) 6-mer overlapping. For each method, count the number of tokens produced and determine whether the start codon `ATG` is preserved as a single unit. Discuss the trade-offs.

13.5. [**Design: Curriculum**] Design a curriculum for training a genomic foundation model on viral genomes specifically. How would your curriculum stages differ from the four-stage curriculum in Section 13.7.3? Consider the unique properties of viral genomes: small size, overlapping reading frames, high mutation rates, and host-dependent codon usage.

13.6. [**Implementation**] Implement a bigram model over the DNA alphabet {A, T, C, G} using only Python's standard library. Train it on a 10,000-nucleotide sequence sampled from any publicly available genome. Report: (a) the 4×4 transition matrix, (b) the per-character cross-entropy in bits, (c) a 100-nucleotide sequence generated by sampling from the model. Compare the generated sequence's GC content to the training data.

13.7. [**Implementation: CodonForge**] Write a simplified CodonForge tokenizer that: (a) detects whether a line is a FASTA header or sequence data, (b) tokenizes headers as bytes, (c) tokenizes coding sequences as codons (assuming frame offset 0), and (d) wraps each modality with appropriate boundary tokens. Test on a FASTA file containing both coding and non-coding sequences.

13.8. [**Theory: Dual-stream**] In the dual-stream architecture (Equation 13.11), the helix stream operates at motif granularity (width $k = 6$) while the standard stream operates at byte

granularity. Derive the sequence length relationship: if the standard stream has T positions, how many positions does the helix stream have? What are the implications for cross-attention alignment between the two streams?

13.9. **[Theory: TetraMemory]** The TetraMemory system (Section 13.11) operates at four scales with different resolutions. Calculate the total memory capacity in nucleotides across all four scales. If a model processes a 1Mb (1,000,000 nt) sequence, what fraction of the sequence can be stored in TetraMemory at each scale? What information must be discarded, and what write policy would you use?

13.10. **[Research]** The cross-modal training loss (Equation 13.16) includes a translation loss $\mathcal{L}_{\text{trans}}$ that mimics biological translation (DNA $\rightarrow$ protein). Propose an additional loss term that would encode *gene regulation*—the context-dependent activation of genes. How would you represent regulatory signals (promoters, enhancers, transcription factor binding sites) in the training data? How would this loss interact with the evolutionary training loop of Chapter 12?

13.11. **[Research: Evaluation]** The benchmarks in Table 13.4 focus on individual genomic tasks. Propose a *holistic* evaluation framework that tests a model's understanding of the Central Dogma as a complete pipeline: from DNA to RNA to protein to function. What tasks would your benchmark include? How would you measure whether a model truly understands the flow of genetic information versus merely memorizing sequence patterns?

13.12. **[Challenge: RC Consistency]** Prove that if a model achieves zero RC consistency loss (Equation 13.13) on all training sequences, its representations must be invariant to strand orientation. Does this imply the model cannot distinguish the two strands? If not, how can strand-specific information be preserved while maintaining consistency? *Hint:* Consider what information is in the pooled representation $\mathbf{z}$ versus the per-position representations $\mathbf{h}^{(t)}$.

13.18 Key Takeaways

Key Takeaways

- Genomic data is stored in FASTA format, which is inherently multimodal: natural language headers describe nucleotide or protein sequences. A foundation model trained on raw FASTA learns both modalities simultaneously.
- Subword tokenizers (BPE, SentencePiece) fail on genomic data because DNA has uniform character frequencies, no whitespace delimiters, and biologically meaningful reading frames that frequency-based merges destroy.
- Three tokenization strategies—BPE, k-mer, and codon-based—each have distinct trade-offs. DOGMA's CodonForge tokenizer unifies all three through a 328-token vocabulary that adapts its strategy to the input modality.
- Byte-level tokenization provides a universal, modality-agnostic fallback: each byte is a token, requiring no learned merge rules and preserving nucleotide-level positional granularity.
- The bigram baseline (zero dependencies, 4×4 transition matrix) reveals dinucleotide biases in genomic data and provides a diagnostic benchmark: the bigram–transformer perplexity gap quantifies long-range structure.
- Pre-training uses two complementary objectives: masked language modeling (MLM) learns bidirectional context for filling in missing nucleotides, while next-token prediction (NTP) learns sequential dependencies and codon usage patterns.
- Curriculum learning progresses through four stages—synthetic sequences, single genes, multi-gene regions, and chromosomal segments—with adaptive stage transitions triggered by loss thresholds rather than fixed step counts.
- The training data pipeline transforms raw GenBank entries through preprocessing, structured rendering, curriculum assignment, and curriculum-aware batching.
- Helix-aware training is a hypothesis using reverse-complement consistency and motif features; the measured baseline has not established a downstream gain.
- Multi-scale TetraMemory proposes hierarchical local memory; length extrapolation remains to be tested against matched recurrent and convolutional baselines.
- Alignment and translation losses are proposed machine-learning objectives, not evidence that a model reconstructs the biological central dogma.
- The earlier 224.7M efficiency and downstream benchmark claims are withdrawn pending matched, reproducible evaluation.
- Staged transfer, TetraMesh multimodality, multiple realization modes, and conversational adaptation remain design studies. The measured hybrid bridge is reported separately from the 29.83M native candidate rejected through cycle 33.

Suggested Reading

- Sennrich et al. [2016]: Byte Pair Encoding—the subword tokenization method whose limitations on genomic data motivate byte-level approaches.
- Ji et al. [2021a]: DNABERT—a BERT-based model for DNA sequences using k-mer tokenization, an alternative to byte-level approaches.
- Dalla-Torre et al. [2024a]: The Nucleotide Transformer—large-scale foundation models for genomics, demonstrating the scaling potential of genomic language models.
- Devlin et al. [2019]: BERT—the masked language modeling objective that inspires MLM-based genomic pretraining.
- Bengio et al. [2009]: Curriculum learning—the theoretical foundation for staged training from simple to complex examples.
- Chapter 11 of this book: CodonForge tokenization and the DNA compiler that the unified vocabulary builds upon.
- Chapter 8 of this book: HelixHash motif embeddings and the dual-stream architecture that helix-aware models build upon.
- Chapter 10 of this book: TetraData rendering pipelines, TetraMemory, and the structured formats used for curriculum construction.
- Chapter 12 of this book: the evolutionary training loop that optimizes prompts for genomic foundation models.
- Chapter 14 of this book: the measured small baseline, its generation failure, and the evaluation program required before scaling claims.

Chapter 14

The Measured DOGMA Baseline

A disappointing measurement is more valuable than an impressive number without provenance.

— DOGMA Evaluation Note

This chapter is the empirical boundary of the book. Earlier editions described a 224.7-million-parameter "DOGMA-Supreme" design and benchmark table as if they were a completed validation. That standard was not met. The material is retained in the companion design studies as a research proposal, but it is not reported here as an achieved result.

The measured baseline available on 27 July 2026 is a **29,830,396-parameter** non-transformer byte-level model. It is useful because it exposes a central failure: lower held-out perplexity did not reliably produce readable free generation. That observation changed the training algorithm and evaluation contract.

Evidence Status

The values in this chapter come from versioned training artifacts and scheduler state. They are an engineering snapshot, not a peer-reviewed benchmark. The current model has not demonstrated AGI, broad biological reasoning, or superiority to transformer, state-space, or genomic foundation-model baselines.

14.1 Implemented Architecture

The baseline configuration is:

Table 14.1: Canonical measured DOGMA baseline configuration.

Property	Value	Interpretation
Model kind	`dogmaforge`	Native Evolutor/DOGMA implementation
Parameters	29,830,396	Exact count recorded by the trainer
Vocabulary	257 byte symbols	UTF-8 bytes plus end-of-sequence
Context length	512 bytes	Short-context research baseline
Embedding dimension	256	Hidden representation width
DOGMA blocks	8	Repeated local computation pipeline
Dropout	0.10	Subject to bounded descendant mutation
Training steps	1,200 root; at most 600 resumed	Small scheduled cycles
Self-attention	absent	Rejected by architecture audit
Transformer layers	0	Rejected by architecture audit

Each block uses an overlapping local TetraMemory representation, a DNA-compute path, a RegulationGate, and a causal transcript mixer. Optional research features include parallel-scan recurrence, bounded epigenetic memory, and sparse allosteric routing. A configuration cannot be called canonical DOGMA if it enables DNA-strand attention, self-attention, or transformer layers.

These names indicate engineering analogies. For example, RegulationGate is a learned software gate; it is not gene regulation occurring in a cell.

14.2 Data Snapshot

The earlier scheduled corpus was unexpectedly small: approximately 44,439 training bytes, 58,063 validation bytes, and 19,792 test bytes in the recorded manifest. The uneven split sizes arose from document-level construction and byte caps rather than a conventional large-corpus benchmark. Three cached reference-genome records provided real sequence anchors; generated architecture text and instruction material supplied most explanatory language.

This is enough for a software smoke test, not enough for claims about general genomic understanding. The revised corpus contract:

1. adds versioned external teaching and distilled corpora;
2. labels reference-genome segments separately from derived or synthetic material;
3. reserves at least 65% of sampling mass for real anchors;
4. preserves a higher real-anchor share when already present; and
5. records exact train/evaluation hash overlap.

Future genomic evaluation must use species-aware or homology-aware splits. Otherwise, highly related sequences can leak across exact document boundaries.

14.3 What the Cycles Measured

Table 14.2 records four early descendants and three recent larger candidates. Lower perplexity is better. "Readable generation" is a qualitative audit of the stored sample under the earlier evaluation protocol; it is not a standardized language score.

Table 14.2: Selected scheduler cycles, including the v2 recursive gate.

Cycle	Valid PPL	Test PPL	Promoted?	Observed free generation
6	2.7367	1.4083	yes	Structured DNA-like runs and fragments, but weak prose
11	2.7875	1.3867	no	Included invalid or binary-looking output
13	2.7359	1.3931	no	Some structured DNA; explanatory text remained poor
14	2.6759	1.3791	no	Lower perplexity, but unreadable generation and no qualifying selected checkpoint
31	2.4183	1.1181	no	Malformed genomic syntax, no qualifying answer fields, and nearly prompt-invariant fragments
32	2.4841	1.1340	no	Faster local step; slightly worse likelihood and the same complete generation-gate failure
33	2.4947	1.1451	no	Evidence-directed 160-to-80-base trace mutation; likelihood regressed and every generation gate still failed

Cycle 6 remained live under the older, smaller architecture because later candidates did not satisfy the complete promotion path. Cycle 31 is not directly comparable to those early runs: it has a larger configuration and a new ATCG-first evaluation contract. Its substantially lower likelihood numbers still did not rescue free generation. Across seeds 17 and 29, all four prompts failed the canonical trace, explanation, critique, general-language, and answer-diversity gates. Cycle 32 tested a faster local learning step and failed the same gates while slightly worsening likelihood. Cycle 33 then used that failure profile to shorten the supervised genomic trace from 160 to 80 bases. This controlled repair attempt also failed every generation gate and slightly worsened likelihood. None of the three candidates was promoted.

14.4 A Separate Hybrid Result

The native failure motivated a staged experiment rather than a false promotion. A deterministic compiler first emits a 64-base ATCG trace and a short transcript; a LoRA adapter on a 0.6B transformer then realizes natural language. The adapter was distilled from a 4B instruction model using 96 records and passed all four held-out `dogma-genomic-language-v3` probes.

This result is reported as a *hybrid bridge*, not as native DOGMA language ability. The native 29.83M candidate remains rejected. Chapter 13, Section 13.16.4, gives the construction, observed answers, and required ablation.

14.5 Why Perplexity and Generation Disagreed

Teacher forcing evaluates

$$p_\theta(x_t \mid x_{<t})$$

on a true prefix. Free generation evaluates the model on prefixes containing its own sampled outputs. A small local error can move sampling onto a prefix that never appeared in training; subsequent errors compound. This difference is a form of exposure bias.

The byte-level vocabulary makes the failure especially visible. Every byte is a legal class, including bytes that do not form printable UTF-8 text. A model can assign good probability to held-out bytes while still sampling broken text. Genomic runs are easier because the desired alphabet has only four visible symbols.

Other plausible contributors are:

- a corpus too small and heterogeneous for stable prose;
- validation and test splits with different composition;
- a short context window;
- sampling temperature and top-k sensitivity;
- over-weighted repetitive genomic patterns; and
- an earlier gate that searched only for a canonical DNA regex.

These are hypotheses to test, not excuses. The correct response is to isolate variables and strengthen evaluation.

14.6 The v2 Correction

The next training cycles use the protocol from Chapter 12. The main corrections are:

1. **Separate capabilities.** DNA syntax, readable explanation, and self-critique are scored independently.
2. **Repeat generation.** Every smoke probe runs under seeds 17 and 29.
3. **Bind metrics to a checkpoint.** Promotion creates `selected_model.pt` pointing to the exact evaluated file.
4. **Use a two-level search.** A behavior archive retains diverse stepping stones; a strict objective and hard gates control live promotion.
5. **Mutate one variable.** A descendant changes only one bounded scalar unless it is a separately declared architecture experiment.
6. **Protect data provenance.** Real anchors keep at least 65% of sampling mass, and exact split overlap must be zero.

The revised objective adds explicit penalties to held-out perplexity:

$$J = P_{\text{test}} + 0.25P_{\text{valid}} + 1.0I_D + 1.5I_E + 1.0I_C + 1.0I_N,$$

where $I_D, I_E, I_C, I_N \in \{0, 1\}$ indicate failures of DNA, explanation, critique, and general-language probes. The implementation also requires prompt-conditioned answer diversity. Every hard gate must pass for promotion.

14.7 Evaluation Program

A credible DOGMA result must graduate through four tiers.

14.7.1 Tier 1: Software Correctness

Unit tests verify shapes, causality, checkpoint loading, deterministic seeds, architecture audits, archive replacement, and data-mixture constraints. Property tests should cover reverse complement handling and finite gradients.

14.7.2 Tier 2: Controlled Sequence Tasks

Synthetic tasks isolate mechanism:

- delayed copy and selective recall;
- motif detection under position shift;
- finite-state languages;
- reverse, complement, and reverse-complement transforms;
- length extrapolation beyond the training range.

These tasks can reveal whether local memory and recurrence actually transmit information. They do not establish biological usefulness.

14.7.3 Tier 3: Genomic Benchmarks

Use frozen tasks with documented licenses and biological splits: promoter or enhancer prediction, splice-site recognition, species classification, variant-effect prediction, and long-range regulatory tasks. Report AUROC, AUPRC, calibration, confidence intervals, throughput, memory, and all preprocessing. Compare with matched-capacity convolutional, recurrent, DNABERT-2, HyenaDNA, Caduceus, and Mamba-style baselines [Ji et al., 2021b, Poli et al., 2023, Gu and Dao, 2023].

14.7.4 Tier 4: Recursive-Learning Evidence

Measure improvement across lineages rather than one checkpoint:

- promotion rate and false-promotion rate;
- retained performance on old tasks;
- archive coverage and diversity;

- improvement per accelerator-hour;
- sensitivity to mutation schedule and random seed;
- synthetic-data fraction and tail retention.

Only after these tiers should the project discuss increasingly general capability. "AGI" is not a benchmark label for a genomic sequence model.

14.8 Near-Term Experiments

The highest-value next experiments are not simply larger models.

1. **Diagnose byte generation.** Compare byte vocabulary with a constrained UTF-8/text head and a four-base genomic head.
2. **Matched architecture ablation.** Remove TetraMemory, recurrence, and regulation one at a time at constant parameter count.
3. **Long-range tests.** Train on lengths up to 256 and evaluate at 512, 1024, and 2048 where the implementation permits.
4. **Archive study.** Compare the quality-diversity archive with a single best-so-far lineage using equal training cycles.
5. **Data study.** Vary real-anchor sampling among 65%, 80%, and 95%, measuring both genomic tasks and explanation quality.
6. **Baseline study.** Match parameters and training bytes against a causal convolutional model, GRU, small Mamba-style model, and transformer.

14.9 Exercises

14.1. Compute the percentage change in test perplexity from cycle 6 to cycle 14. Why is that not sufficient evidence to promote cycle 14?

14.2. The validation perplexity is higher than test perplexity in every row. Give three data-split explanations and an experiment to distinguish them.

14.3. Design a printable UTF-8 validity metric that does not reward empty or repetitive output.

14.4. Write a matched-compute comparison protocol for DOGMA and a GRU.

14.5. Explain why reverse-complement augmentation can improve invariance but can also leak information if applied before splitting.

14.6. Propose confidence intervals for a promoter-classification experiment with organism-level grouping.

14.7. Design an ablation that tests whether RegulationGate contributes beyond adding an equal number of ordinary parameters.

14.8. Specify evidence that would falsify the hypothesis that TetraMemory helps length extrapolation.

14.10 Key Takeaways

Key Takeaways

- The current native DOGMA candidate has 29,830,396 parameters, not 224.7M.
- It is a non-transformer research model, not AGI and not a demonstrated replacement for established genomic architectures.
- Later cycles lowered perplexity without producing reliably readable text; this exposed a weakness in the former promotion gate.
- A separate ATCG-to-language hybrid passed four held-out probes, but its transformer realizer does not establish native DOGMA language ability.
- The v2 loop binds evaluation to exact checkpoints, separates three generation capabilities, repeats seeds, protects real-data anchors, and archives diverse stepping stones.
- The next scientific milestone is a matched, contamination-aware evaluation program, not a larger unsupported claim.

Suggested Reading

- Gu and Dao [2023] for selective state-space sequence modeling.
- Poli et al. [2023] for long-range genomic sequence modeling.
- Mouret and Clune [2015] for quality-diversity archives.
- Shumailov et al. [2024] for recursive synthetic-data risks.

Part V

Toward AGI

Chapter 15

Scaling Laws and Post-Transformer Architectures

There is no reason why intelligence should be proportional to model size. Efficiency is a form of intelligence.

— DOGMA Design Manifesto

The dominant paradigm in AI since 2020 has been simple: *make the model bigger*. Scaling laws revealed smooth power-law relationships between model size, training compute, dataset size, and loss. These laws enabled precise predictions of model capability and drove the largest engineering projects in AI history—GPT-4, Gemini, Claude, and their successors consumed hundreds of millions of dollars in compute.

But scaling alone cannot be the complete story. Quadratic attention costs, energy consumption, and the diminishing returns of brute-force scaling create pressure for *architectural innovation*—new ways of organizing computation that achieve better capability per parameter. This chapter examines the scaling paradigm in detail, surveys the emerging post-transformer landscape (state space models, linear attention, mixture-of-experts), and situates DOGMA within this evolving ecosystem. We will see that DOGMA's biology-inspired structure offers a fundamentally different scaling axis: one that draws on the organizational principles that allowed biological genomes to scale from bacteria ($\sim 10^6$ bases) to humans ($\sim 3 \times 10^9$ bases) without quadratic complexity.

15.1 Neural Scaling Laws

The empirical discovery of neural scaling laws transformed AI research from art into something closer to engineering. Before scaling laws, predicting the performance of a model at a given size was guesswork. After scaling laws, it became a calculation.

15.1.1 The Kaplan Scaling Laws

Kaplan et al. [2020a] established that cross-entropy loss L on language modeling follows power-law scaling across three independent dimensions:

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}, \tag{15.1}$$

where the variables are defined as follows:

- N is the number of non-embedding parameters in the model.
- D is the number of training tokens (dataset size).
- C is the total compute budget measured in FLOPs.
- N_c, D_c, C_c are critical constants that depend on the architecture and data distribution.
- $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$, $\alpha_C \approx 0.050$ are empirically determined exponents.

Example 15.1 (Reading the Scaling Exponents)**.** Consider the parameter scaling law $L(N) = (N_c/N)^{0.076}$. Suppose we double the model size from N to $2N$. The new loss is:

$$L(2N) = \left(\frac{N_c}{2N}\right)^{0.076} = \frac{1}{2^{0.076}} \cdot L(N) \approx 0.949 \cdot L(N).$$

Doubling model size reduces loss by approximately 5.1%. This seemingly modest improvement compounds: a 100× increase in parameters yields $L(100N) \approx 0.71 \cdot L(N)$, a 29% reduction. The power-law form means we never reach zero loss—each increment of capability costs exponentially more resources.

The key finding of Kaplan et al. [2020a] was that these power laws are *smooth and predictable*. There are no sudden jumps or plateaus within an architecture family. This predictability enabled organizations to plan multi-hundred-million-dollar training runs with reasonable confidence in the outcome.

15.1.2 The Chinchilla Correction

Hoffmann et al. [2022a] revisited the Kaplan scaling laws and arrived at a different conclusion about the optimal allocation of a fixed compute budget C between model size N and dataset size D. Where Kaplan et al. recommended scaling parameters faster than data, the Chinchilla analysis found that *model size and data should be scaled in roughly equal proportion*:

$$N_{\text{opt}} \propto C^a, \quad D_{\text{opt}} \propto C^b, \quad \text{with } a \approx b \approx 0.5. \tag{15.2}$$

Definition 15.1 (Compute-Optimal Training)**.** A model is *compute-optimal* if, for a given compute budget C, the model size N and dataset size D are chosen to minimize the loss $L(N, D)$ subject to the constraint $C \approx 6ND$ (the approximate FLOP cost of a single training epoch on D tokens with an N-parameter model).

The practical implication was dramatic. Many existing models—including the original Chinchilla paper's comparison target, Gopher (280B parameters)—were *over-parameterized and under-trained*. A smaller model trained on more data (Chinchilla, 70B parameters) achieved the same or better performance at lower inference cost.

Table 15.1: Comparison of Kaplan vs. Chinchilla scaling recommendations.

Aspect	**Kaplan et al. (2020)**	**Chinchilla (2022)**
Optimal N scaling	$N \propto C^{0.73}$	$N \propto C^{0.50}$
Optimal D scaling	$D \propto C^{0.27}$	$D \propto C^{0.50}$
Key insight	Scale parameters aggressively	Scale parameters and data equally
Practical impact	Very large, undertrained models	Smaller, better-trained models

15.1.3 Visualizing Scaling Laws

The power-law relationships of Equation 15.1 become linear on log-log axes, which is why log-log plots are the standard visualization for scaling behavior.

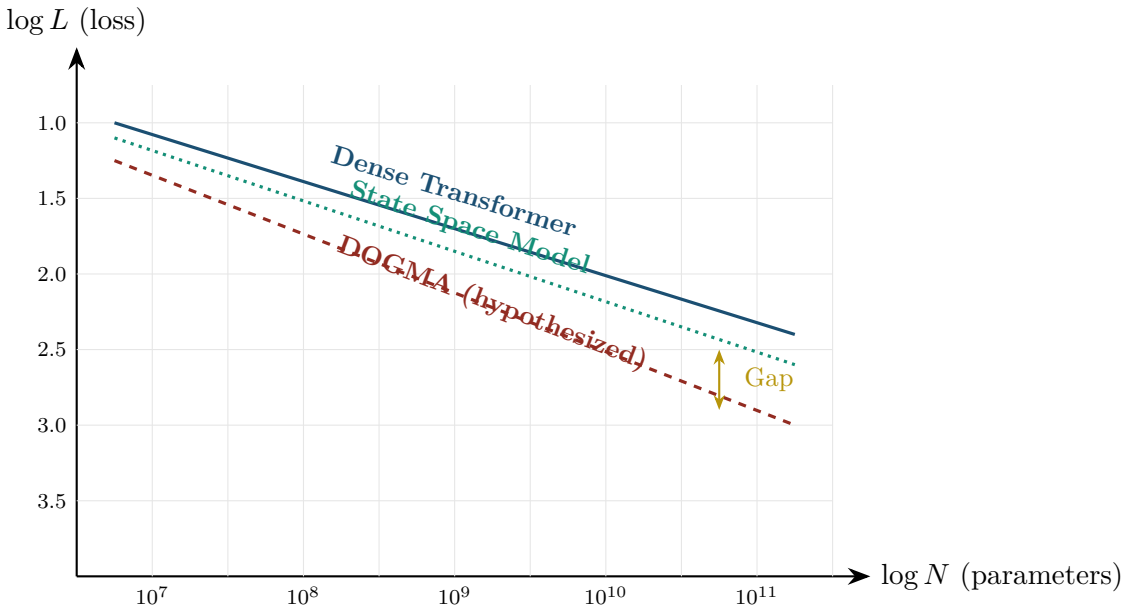


Figure 15.1: Schematic log-log scaling plot. Dense transformers (solid blue) follow established power-law scaling. State space models (dotted teal) show competitive scaling with lower per-token cost. DOGMA (dashed red) is hypothesized to achieve better loss-per-parameter through biological structural priors. The gold arrow indicates the scaling efficiency gap at large model sizes.

15.1.4 Emergent Abilities

Wei et al. [2022a] documented *emergent abilities*: capabilities that appear suddenly as models cross certain size thresholds. In-context learning, chain-of-thought reasoning, and instruction following all emerge at specific scale points—typically between 10^{10} and 10^{11} parameters.

Definition 15.2 (Emergent Ability)**.** An ability is *emergent* if it is not present in smaller models but appears in larger models. Formally, let $\mathcal{A}(N)$ denote the performance of a model of size N on a specific task. The ability is emergent if $\mathcal{A}(N) \approx 0$ for $N < N^*$ and $\mathcal{A}(N) > \delta$ for $N > N^*$, where N^* is the emergence threshold and δ is a meaningful performance level.

Whether these abilities are truly discontinuous or merely appear so due to evaluation metrics remains debated [Schaeffer et al., 2023a]. What is clear is that scale enables qualitatively different behaviors—and that architectural innovations (like DOGMA's structured genome) might enable similar capabilities at smaller scale by providing the structural scaffolding that dense models must discover from data alone.

Scaling Laws and Architecture

Scaling laws hold *within an architecture family*. Changing the architecture changes the scaling exponents and constants—the slopes and intercepts of the log-log lines in Figure 15.1. DOGMA's thesis is that biology-inspired structured architectures may achieve better scaling—higher capability per parameter—by introducing inductive biases that reduce the amount of structure the model must learn from data alone.

15.2 The Limits of Dense Scaling

Despite the impressive trajectory of scaling, several fundamental factors limit the indefinite scaling of dense transformer models. Understanding these limits is essential for appreciating why architectural innovation—not just bigger models—is the path forward.

15.2.1 Quadratic Attention Cost

Self-attention scales as $O(T^2)$ in sequence length T, where T is the number of tokens in the input. For a single attention head with query, key, and value matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{T \times d}$:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}, \tag{15.3}$$

the matrix product $\mathbf{Q}\mathbf{K}^\top$ requires $O(T^2 d)$ operations and $O(T^2)$ memory. This creates a hard wall for long-context applications:

Example 15.2 (The Quadratic Wall)**.** Consider processing a human genome ($T \approx 3 \times 10^9$ base pairs) with standard attention:

- The attention matrix would have $(3 \times 10^9)^2 = 9 \times 10^{18}$ entries.
- At 2 bytes per entry (FP16), this requires **18 exabytes** of memory—more than the total global DRAM production in a year.
- Even for a modest context window of $T = 10^6$ tokens, the attention matrix requires 2 terabytes.

The quadratic wall is not merely an inconvenience; it is a fundamental barrier to processing the long sequences that many real-world tasks demand.

15.2.2 Energy and Environmental Cost

Training a frontier model consumes enormous energy. The training of GPT-4 is estimated to have required on the order of 10^{25} FLOPs, consuming megawatt-hours of electricity over several months. Each successive generation of frontier models approximately doubles or triples the energy requirement while yielding progressively smaller loss improvements (because the power-law exponent $\alpha_C \approx 0.05$ is small).

15.2.3 Data Exhaustion

High-quality text data is finite. Models are approaching the boundary of available internet text. Villalobos et al. [2022] estimated that high-quality language data may be exhausted by 2026–2028 at current training scales. While synthetic data generation offers a partial solution, it introduces risks of model collapse—where models trained on their own outputs degrade in quality over generations.

15.2.4 Capability Plateaus

Certain capabilities have not shown the same smooth scaling as perplexity:

- **Robust reasoning:** Multi-step logical and mathematical reasoning improves with scale but remains fragile, especially on out-of-distribution problems.
- **Planning:** Long-horizon planning and goal-directed behavior remain weak even in the largest models.
- **Self-correction:** Models struggle to identify and correct their own errors without external scaffolding.
- **Persistent learning:** No amount of scaling gives a transformer persistent memory across sessions.

These plateaus suggest that some capabilities require not just more computation but *different computation*—architectural structures that provide the right inductive biases for reasoning, planning, and learning.

Remark 15.1. The limits of dense scaling do not imply that scaling is useless. Scaling has been the single most productive research strategy of the past five years. The point is that scaling alone is *insufficient*—it must be complemented by architectural innovation to overcome the barriers enumerated above.

15.3 State Space Models: From S4 to Mamba

State space models (SSMs) represent the most significant challenge to the transformer's dominance. They process sequences in $O(T)$ time by modeling sequence dependencies as linear dynamical systems, achieving competitive performance with transformers on many benchmarks while avoiding the quadratic attention bottleneck.

15.3.1 The Continuous-Time State Space

An SSM maps an input sequence $u(t) \in \mathbb{R}$ to an output sequence $y(t) \in \mathbb{R}$ through a latent state $x(t) \in \mathbb{R}^n$:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t), \quad y(t) = \mathbf{C}x(t) + Du(t), \tag{15.4}$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is the state transition matrix, $\mathbf{B} \in \mathbb{R}^{n \times 1}$ is the input matrix, $\mathbf{C} \in \mathbb{R}^{1 \times n}$ is the output matrix, and $D \in \mathbb{R}$ is a feedthrough scalar. To process discrete sequences (as in language modeling), we discretize using a step size Δ:

$$\bar{\mathbf{A}} = \exp(\Delta \mathbf{A}), \quad \bar{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(\bar{\mathbf{A}} - \mathbf{I}) \cdot \Delta \mathbf{B}. \tag{15.5}$$

The discrete-time recurrence then becomes:

$$x_k = \bar{\mathbf{A}} x_{k-1} + \bar{\mathbf{B}} u_k, \quad y_k = \mathbf{C} x_k + D u_k. \tag{15.6}$$

The SSM Dual View

The SSM admits two equivalent computational views:

1. **Recurrent view:** Process tokens one at a time using the recurrence in Equation 15.6. Cost: $O(T \cdot n)$. Memory: $O(n)$. Ideal for inference.
2. **Convolutional view:** Unroll the recurrence into a convolution $y = K * u$ where $K_k = \mathbf{C}\bar{\mathbf{A}}^k\bar{\mathbf{B}}$. Compute via FFT in $O(T \log T)$. Ideal for training.

This dual view gives SSMs both fast training (parallel convolution) and fast inference (sequential recurrence), a combination that transformers lack.

15.3.2 S4: Structured State Spaces for Sequences

The Structured State Space for Sequences (S4) model [Gu et al., 2022] addressed a key challenge: how to parameterize the state matrix $\mathbf{A}$ so that the SSM can capture long-range dependencies. The solution was to initialize $\mathbf{A}$ using the *HiPPO* (High-Order Polynomial Projection Operator) matrix, which is designed to optimally compress the history of the input sequence into the state vector:

$$A_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k, \\ n+1 & \text{if } n = k, \\ 0 & \text{if } n < k. \end{cases} \tag{15.7}$$

This initialization ensures that the state x_k maintains a compressed polynomial approximation of the entire input history up to step k, enabling the model to capture dependencies across thousands of time steps.

Example 15.3 (Worked Example: Processing a Sequence Through a 2-State SSM)**.** Let us trace a concrete 4-element sequence $u = [1.0, 0.5, 0.8, 0.2]$ through a minimal SSM with state dimension $n = 2$. We use:

$$\mathbf{A} = \begin{pmatrix} -1 & 0 \\ 0 & -2 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 1 & 0.5 \end{pmatrix}, \quad D = 0.$$

With step size $\Delta = 0.5$, the discretized matrices are:

$$\bar{\mathbf{A}} = \exp(\Delta \mathbf{A}) = \begin{pmatrix} e^{-0.5} & 0 \\ 0 & e^{-1.0} \end{pmatrix} \approx \begin{pmatrix} 0.607 & 0 \\ 0 & 0.368 \end{pmatrix}.$$

For $\bar{\mathbf{B}}$, using the simplified form $\bar{\mathbf{B}} \approx \Delta \mathbf{B}$:

$$\bar{\mathbf{B}} \approx 0.5 \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}.$$

Step 1 ($u_1 = 1.0$, start from $x_0 = [0, 0]^\top$):

$$x_1 = \bar{\mathbf{A}} x_0 + \bar{\mathbf{B}} u_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix} = \begin{pmatrix} 0.500 \\ 0.500 \end{pmatrix}, \quad y_1 = \mathbf{C} x_1 = 1.0 \cdot 0.500 + 0.5 \cdot 0.500 = \mathbf{0.750}.$$

Step 2 ($u_2 = 0.5$):

$$x_2 = \begin{pmatrix} 0.607 \cdot 0.500 + 0.5 \cdot 0.5 \\ 0.368 \cdot 0.500 + 0.5 \cdot 0.5 \end{pmatrix} = \begin{pmatrix} 0.554 \\ 0.434 \end{pmatrix}, \quad y_2 = 0.554 + 0.5 \cdot 0.434 = \mathbf{0.771}.$$

Step 3 ($u_3 = 0.8$):

$$x_3 = \begin{pmatrix} 0.607 \cdot 0.554 + 0.5 \cdot 0.8 \\ 0.368 \cdot 0.434 + 0.5 \cdot 0.8 \end{pmatrix} = \begin{pmatrix} 0.736 \\ 0.560 \end{pmatrix}, \quad y_3 = 0.736 + 0.5 \cdot 0.560 = \mathbf{1.016}.$$

Step 4 ($u_4 = 0.2$):

$$x_4 = \begin{pmatrix} 0.607 \cdot 0.736 + 0.5 \cdot 0.2 \\ 0.368 \cdot 0.560 + 0.5 \cdot 0.2 \end{pmatrix} = \begin{pmatrix} 0.547 \\ 0.306 \end{pmatrix}, \quad y_4 = 0.547 + 0.5 \cdot 0.306 = \mathbf{0.700}.$$

The output sequence is $y = [0.750, 0.771, 1.016, 0.700]$. Notice how the state *remembers* past inputs: $y_3 = 1.016$ is higher than $u_3 = 0.8$ because the state carries forward information from the earlier large input $u_1 = 1.0$. The exponential decay in $\bar{\mathbf{A}}$ (0.607 and 0.368) ensures that older information fades, with the second state dimension decaying faster (modeling shorter-range dependencies).

SSMs and Biological Signal Propagation

The state space formulation mirrors how biological neurons propagate signals. The state vector x_k is analogous to membrane potential: it integrates incoming signals ($\bar{\mathbf{B}}u_k$) while decaying toward rest ($\bar{\mathbf{A}}x_{k-1}$). The HiPPO initialization ensures the "neuron" remembers a polynomial summary of its recent input history—remarkably similar to how synaptic plasticity encodes temporal patterns of stimulation. DOGMA's DNA computing layers implement a biological version of this principle: strand displacement cascades provide selective memory analogous to Mamba's input-dependent gating.

15.3.3 Mamba: Selective State Spaces

Mamba [Gu and Dao, 2023] introduced a crucial innovation: *input-dependent* (selective) parameterization. In S4, the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are fixed across all inputs. In Mamba, $\mathbf{B}$, $\mathbf{C}$, and the step size Δ are functions of the input:

$$\mathbf{B}_k = \text{Linear}_B(u_k), \quad \mathbf{C}_k = \text{Linear}_C(u_k), \quad \Delta_k = \text{softplus}(\text{Linear}_\Delta(u_k)). \tag{15.8}$$

This selectivity allows Mamba to perform content-based reasoning: the model can *choose* which information to remember and which to forget at each time step, analogous to the gating mechanisms in LSTMs but operating through the SSM framework.

Example 15.4 (Step-by-Step Selective Scan)**.** Consider processing three tokens u_1, u_2, u_3 through a Mamba layer with state dimension $n = 2$:

Step 1. Compute input-dependent parameters for u_1:

$$\mathbf{B}_1 = \text{Linear}_B(u_1), \quad \mathbf{C}_1 = \text{Linear}_C(u_1), \quad \Delta_1 = \text{softplus}(\text{Linear}_\Delta(u_1)).$$

Discretize: $\bar{\mathbf{A}}_1 = \exp(\Delta_1 \mathbf{A})$, $\bar{\mathbf{B}}_1 = f(\Delta_1, \mathbf{A}) \cdot \mathbf{B}_1$.

Step 2. Update state:

$$x_1 = \bar{\mathbf{A}}_1 \cdot x_0 + \bar{\mathbf{B}}_1 \cdot u_1.$$

Output: $y_1 = \mathbf{C}_1 \cdot x_1$.

Step 3. Repeat for u_2: compute $\mathbf{B}_2, \mathbf{C}_2, \Delta_2$ from u_2; update $x_2 = \bar{\mathbf{A}}_2 x_1 + \bar{\mathbf{B}}_2 u_2$; output $y_2 = \mathbf{C}_2 x_2$.

Step 4. Repeat for u_3: analogously.

The critical feature is that Δ_k controls the "forget rate." A large Δ_k means $\bar{\mathbf{A}}_k \approx \mathbf{0}$, effectively resetting the state (forgetting previous context). A small Δ_k means $\bar{\mathbf{A}}_k \approx \mathbf{I}$, preserving the state (remembering context). The model learns *when* to forget through the input-dependent parameterization.

15.3.4 The Hardware-Aware Scan Algorithm

A naive implementation of Mamba's selective scan is sequential (each state depends on the previous one), which would be slow on GPUs. Mamba solves this through a *parallel scan* algorithm:

Proposition 15.1 (Parallel Scan Complexity)**.** The selective scan over a sequence of length T can be computed in $O(T \log T)$ work and $O(\log T)$ depth using the parallel scan (prefix sum) algorithm. On a GPU with P processors, the wall-clock time is $O(T \log T / P + \log T)$.

The key insight is that the recurrence $x_k = \bar{\mathbf{A}}_k x_{k-1} + \bar{\mathbf{B}}_k u_k$ can be reformulated as an associative binary operation, enabling the use of Blelloch's parallel prefix sum algorithm. Mamba further optimizes this by keeping the state in GPU SRAM (fast on-chip memory) rather than HBM (slow off-chip memory), achieving up to 3× speedup over standard implementations.

Example 15.5 (Worked Example: Selective Scan with Actual Numbers)**.** Process the same sequence $u = [1.0, 0.5, 0.8, 0.2]$ through a *selective* Mamba layer ($n = 2$). Unlike the fixed SSM above, here $\mathbf{B}_k$ and Δ_k are input-dependent:

$$\mathbf{B}_k = W_B \cdot u_k, \quad \Delta_k = \text{softplus}(w_\Delta \cdot u_k + b_\Delta).$$

With $W_B = [0.6,\ 0.4]^\top$, $w_\Delta = 1.0$, $b_\Delta = -0.5$, $\mathbf{A} = \text{diag}(-1, -2)$, $\mathbf{C} = [1, 0.5]$:

Step 1 ($u_1 = 1.0$): $\Delta_1 = \text{softplus}(1.0 - 0.5) = \ln(1 + e^{0.5}) \approx 0.974$.

$$\bar{\mathbf{A}}_1 = \begin{pmatrix} e^{-0.974} & \\ & e^{-1.948} \end{pmatrix} \approx \begin{pmatrix} 0.378 & \\ & 0.143 \end{pmatrix}, \quad \bar{\mathbf{B}}_1 \approx 0.974 \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} = \begin{pmatrix} 0.584 \\ 0.390 \end{pmatrix}.$$

$x_1 = \bar{\mathbf{B}}_1 \cdot 1.0 = [0.584, 0.390]^\top$, $\quad y_1 = 0.584 + 0.5 \cdot 0.390 = \mathbf{0.779}$.

Step 2 ($u_2 = 0.5$): $\Delta_2 = \text{softplus}(0.5 - 0.5) = \ln 2 \approx 0.693$.

$$\bar{\mathbf{A}}_2 \approx \begin{pmatrix} 0.500 & \\ & 0.250 \end{pmatrix}, \quad \bar{\mathbf{B}}_2 \approx 0.693 \begin{pmatrix} 0.3 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 0.208 \\ 0.139 \end{pmatrix}.$$

$x_2 = [0.500 \cdot 0.584 + 0.208 \cdot 0.5,\ 0.250 \cdot 0.390 + 0.139 \cdot 0.5]^\top = [0.396, 0.167]^\top$, $\quad y_2 = \mathbf{0.480}$.

Notice how the *smaller* input $u_2 = 0.5$ produces a *smaller* $\Delta_2 = 0.693 < 0.974$, which makes $\bar{\mathbf{A}}_2$ larger ($0.500 > 0.378$). This means the model *remembers more* when the input is weak (nothing important to overwrite) and *forgets more* when the input is strong (make room for new information). This is the essence of selectivity.

Connection to DOGMA's ParallelScan Path

DOGMA's Path B (Automaton) in the DNA Computing Block uses the *same* parallel scan algorithm as Mamba, but with a key difference: the state dimension is biologically motivated (automaton states represent molecular configurations) rather than arbitrary. The scan state $s_k = \bar{\mathbf{A}}_k s_{k-1} + \bar{\mathbf{B}}_k x_k$ is computed using the same Blelloch prefix sum, giving DOGMA's automaton path $O(T \log T)$ complexity identical to Mamba. The biological grounding means each state dimension has an interpretable meaning (e.g., "concentration of species i"), unlike Mamba's opaque hidden states.

15.4 RWKV and Linear Attention

While Mamba replaces attention with state space recurrences, another family of models seeks to *linearize* attention itself, retaining the transformer's conceptual framework while eliminating the quadratic cost.

15.4.1 The Quadratic Bottleneck in Attention

Standard attention computes:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}. \tag{15.9}$$

The softmax over $\mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{T \times T}$ forces materialization of the full $T \times T$ matrix, giving $O(T^2)$ cost.

15.4.2 Linear Attention via Kernel Decomposition

Linear attention [Katharopoulos et al., 2020] replaces the softmax with a feature map ϕ:

$$\text{LinAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \frac{\phi(\mathbf{Q})\left(\phi(\mathbf{K})^\top \mathbf{V}\right)}{\phi(\mathbf{Q})\left(\phi(\mathbf{K})^\top \mathbf{1}\right)}, \tag{15.10}$$

where the key trick is the *order of operations*: instead of computing $\phi(\mathbf{Q})\phi(\mathbf{K})^\top$ (which is $T \times T$), we first compute $\phi(\mathbf{K})^\top \mathbf{V}$ (which is $d \times d$), then multiply by $\phi(\mathbf{Q})$. This reduces the cost from $O(T^2 d)$ to $O(T d^2)$—linear in sequence length.

15.4.3 RWKV: Receptance Weighted Key Value

RWKV [Peng et al., 2023] combines ideas from transformers and RNNs into an architecture that can be trained like a transformer (parallelizable) but runs at inference like an RNN (constant memory per token). The core mechanism is the WKV (Weighted Key Value) operator:

$$\text{wkv}_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w + k_i} v_i + e^{u + k_t} v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w + k_i} + e^{u + k_t}}, \tag{15.11}$$

where w is a learned channel-wise decay vector, k_i and v_i are key and value vectors at position i, and u is a learned bonus for the current token.

Definition 15.3 (WKV Mechanism)**.** The WKV operator computes a weighted average of all value vectors, where the weights decay exponentially with distance (controlled by w) and are modulated by key-query similarity (controlled by k_i). The parameter u gives a "bonus" to the current token, ensuring that the model always has strong access to the present input.

Example 15.6 (WKV as Exponentially Decaying Attention)**.** Consider a sequence of five tokens. At position $t = 5$, the WKV operator assigns weights:

$$\begin{aligned}
&\text{Weight for } t = 1 \propto e^{-3w+k_1} \quad \text{(3 steps ago, strongly decayed)}\\
&\text{Weight for } t = 2 \propto e^{-2w+k_2} \quad \text{(2 steps ago)}\\
&\text{Weight for } t = 3 \propto e^{-w+k_3} \quad \text{(1 step ago)}\\
&\text{Weight for } t = 4 \propto e^{k_4} \quad \text{(previous token, no decay)}\\
&\text{Weight for } t = 5 \propto e^{u+k_5} \quad \text{(current token, with bonus)}
\end{aligned}$$

The exponential decay creates a natural locality bias while the key values k_i allow the model to "boost" attention to contextually important tokens regardless of distance.

The WKV operator can be computed recurrently in $O(1)$ per token using running accumulators, giving $O(T)$ total cost. During training, the parallel form uses chunkwise computation for GPU efficiency.

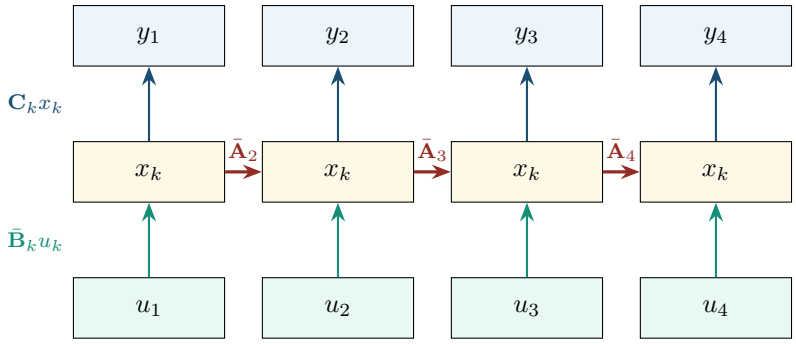


Figure 15.2: The recurrent view of a state space model (or RWKV). Each input u_k updates the hidden state x_k through an input-dependent transition, and the output y_k is a linear projection of the state. The state propagates left-to-right, giving $O(T)$ cost.

15.5 Mixture of Experts (MoE)

Mixture of Experts (MoE) architectures attack the scaling problem from a different angle: instead of reducing the cost per token, they increase the total parameter count while keeping the *active* parameter count constant. This is a form of *conditional computation*—only a subset of the model is activated for any given input.

15.5.1 The MoE Architecture

In a standard MoE transformer, each feed-forward network (FFN) layer is replaced by E parallel "expert" networks $\{f_1, \ldots, f_E\}$ and a gating (routing) function G:

$$\text{MoE}(x) = \sum_{i=1}^{E} G(x)_i \cdot f_i(x), \quad G(x) = \text{TopK}(\text{softmax}(W_g \cdot x), k)\,, \tag{15.12}$$

where TopK selects the k experts with the highest gating scores and zeros out the rest. Typically $k = 1$ or $k = 2$, so only 1–2 of E experts are active per token.

Definition 15.4 (Sparse Routing)**.** *Sparse routing* refers to the MoE gating mechanism where each input token is processed by only $k \ll E$ experts. The active compute per token is $O(k \cdot d_{\text{expert}})$ rather than $O(E \cdot d_{\text{expert}})$, enabling models with very large total parameter counts (e.g., 1.8 trillion for Mixtral 8×7B) to have tractable inference costs.

15.5.2 Expert Parallelism and Load Balancing

A key engineering challenge in MoE is *load balancing*: ensuring that tokens are distributed roughly evenly across experts. Without load balancing, popular experts become bottlenecks while unpopular experts waste compute.

The standard approach is to add an auxiliary loss that encourages balanced routing:

$$\mathcal{L}_{\text{balance}} = \alpha \cdot E \cdot \sum_{i=1}^{E} f_i \cdot P_i, \tag{15.13}$$

where f_i is the fraction of tokens routed to expert i, P_i is the mean gating probability for expert i, and α is a balancing coefficient. Minimizing $\mathcal{L}_{\text{balance}}$ pushes toward uniform routing ($f_i = P_i = 1/E$ for all i).

15.5.3 Expert Specialization

Despite the pressure toward balanced routing, experts tend to *specialize*: some experts handle mathematical reasoning, others handle code, others handle natural language generation. This emergent specialization is reminiscent of cell differentiation in biology—a connection that DOGMA exploits architecturally.

MoE and Cell Differentiation

In biology, all cells contain the same genome but express different genes through regulatory mechanisms, producing neurons, muscle cells, and immune cells from the same genetic material. MoE achieves a similar effect: all experts share the same architecture but develop different "specializations" through training. The difference is that biological differentiation is driven by a rich regulatory network operating on a structured genome, while MoE specialization emerges from a simple gating function operating on a flat token representation. DOGMA's regulation stage provides the biological level of control over specialization.

15.6 How DOGMA Scales

Having surveyed the post-transformer landscape, we now analyze DOGMA's scaling properties and argue that its biology-inspired structure provides a fundamentally different—and potentially superior—scaling paradigm.

15.6.1 The Four Scaling Axes of DOGMA

Dense transformers scale along essentially one axis: model size (number of parameters). DOGMA provides four distinct scaling axes, each inspired by how biological genomes scale:

1. **Parameter scaling (Genome Size).** Like all architectures, DOGMA can scale by adding more genomic bases, increasing embedding dimension, or deepening the pipeline. This is analogous to genome size increase through DNA duplication.

2. **Structural scaling (Genomic Organization).** DOGMA can scale by adding new genomic *regions*, new region types, or deeper regulatory hierarchies—without simply adding parameters uniformly. This is analogous to how biological genomes scale through gene duplication and neofunctionalization: new genes arise by copying existing ones, then diverge to acquire new functions.

3. **Regulatory scaling (Expression Complexity).** The regulation stage can increase in complexity independently of the genome size: more regulatory elements, more complex regulatory logic, deeper regulatory cascades. This mirrors the observation that the number of regulatory genes scales faster than the total gene count as organisms become more complex.

4. **Evolutionary scaling (Meta-Learning).** The meta-policy memory enables cumulative improvement across training runs. Each generation of evolved genomes starts from a better population, potentially yielding super-linear improvement curves across evolutionary cycles.

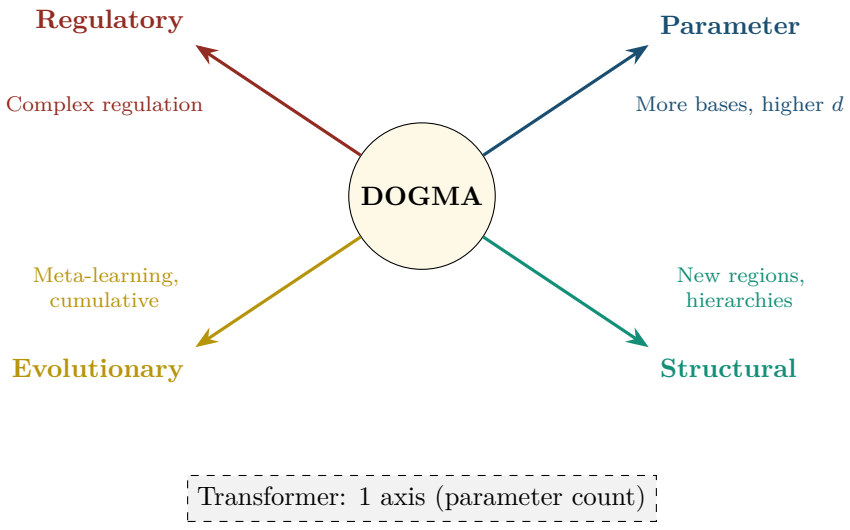


Figure 15.3: DOGMA's four scaling axes compared to the transformer's single axis. Each DOGMA axis corresponds to a biological scaling mechanism.

15.6.2 Sample Efficiency Through Biological Priors

Biological inductive biases such as typed bases, regulatory sparsity, and local structure *may* reduce the data needed to learn some relationships, but they can also impose the wrong bias. DOGMA encodes these distinctions architecturally; matched experiments must determine whether that helps.

Proposition 15.2 (Sample Efficiency Hypothesis)**.** Let $L_{\text{DOGMA}}(D)$ and $L_{\text{Transformer}}(D)$ denote the loss achieved by DOGMA and a dense transformer, respectively, after training on D tokens with the same compute budget. If DOGMA's structural priors correctly capture regularities present in the data, then:

$$L_{\text{DOGMA}}(D) \leq L_{\text{Transformer}}(\beta D) \quad \text{for some } \beta > 1, \tag{15.14}$$

meaning DOGMA achieves comparable loss with $1/\beta$ of the data. The constant β depends on how well the structural priors match the data distribution.

15.6.3 Why Four-Path Diversity Helps Scaling

DOGMA's four base types create four parallel processing pathways through the architecture. Each type specializes in a different computational role:

Base Type	Role	Scaling Benefit
A (Archival)	Long-term knowledge storage	Scales knowledge capacity without increasing active compute
T (Transient)	Working memory	Scales context processing independently of storage
C (Control)	Regulatory signals	Scales expression complexity without increasing parameters
G (Generative)	Output synthesis	Scales generation quality independently of knowledge

The intended benefit is partially independent scaling of capabilities. Whether the paths specialize as intended, rather than duplicating or interfering with one another, is an open ablation question.

15.6.4 Parameter Allocation: DOGMA vs. Transformer

The following historical design exercise allocates a hypothetical 224.7M parameters. It was not the model measured in Chapter 14 and must not be read as a completed implementation or benchmark.

Table 15.2: Hypothetical matched-size allocation study: proposed DOGMA design (224.7M) versus a transformer design (224M).

Component	DOGMA	Transformer	Notes
Token embedding	0.13M	16.4M	DOGMA: 257×512; Trans: $32{,}000 \times 512$
TetraMemory	33.8M	—	K_4 graph ops, face propagation, multi-scale
DNA Computing (4 paths)	88.3M	—	Strand displ. + automaton + stochastic + double
Regulation Gate	22.4M	—	Sigmoid gating, analogous to Trans. layer norm
Strand Attention	44.8M	100.7M	Thermo-weighted vs. **QKV** projections
Expression Folder	22.4M	—	LayerNorm + MLP folding
FFN / MLP	—	100.7M	$4d \times d$ up/down projections
Epigenetic Memory	8.4M	—	Persistent cross-block state
Allosteric Regulation	4.2M	—	Sparse top-k communication
Output head	0.13M	6.1M	DOGMA: byte-level; Trans: 32K vocab
Total	**224.7M**	**224M**	

What the Allocation Would Test

The proposed comparison contrasts repeated attention/FFN blocks with functionally named components. Names do not guarantee specialization. A valid study would test:

- whether four compute paths outperform one equally sized recurrent or convolutional path;
- whether a 257-byte vocabulary's smaller embedding offsets its longer sequences and additional compute; and
- whether local memory plus recurrence improves length extrapolation.

15.6.5 Per-Block FLOP Analysis

For a concrete sense of computational cost, here is the FLOP count for one DOGMA block processing a batch of shape (B=4, T=512, d=512):

Stage	FLOPs per block	Computation
TetraMemory	∼0.5B	$4 \times B \times T \times d$ face ops
DNA Compute (4 paths)	∼2.1B	CRN: $B \times T \times 64 \times 32$; Scan: $O(T \log T)$; etc.
Regulation Gate	∼0.5B	$B \times T \times 2d^2$ (gate projection)
Strand Attention	∼1.1B	$B \times T^2 \times d/h$ (thermo-weighted)
Expression Folder	∼0.5B	$B \times T \times 4d^2$ (MLP folding)
Total per block	∼4.7B	
16 blocks	∼75B	Comparable to 12-layer Transformer (∼72B)

The total per-forward-pass cost is comparable to a same-sized Transformer, but the *distribution* of compute is fundamentally different: DOGMA spends 45% of FLOPs in the DNA Computing Block (structured computation) while a Transformer spends 67% in FFN layers (unstructured matrix multiplications).

15.6.6 Complexity Analysis

DOGMA's computational cost per layer for a sequence of length T is:

$$C_{\text{DOGMA}}(T) = O(T \cdot k \cdot d), \tag{15.15}$$

where k is the average number of active neighbors in the tetrahedral mesh (typically 4–8) and d is the embedding dimension. For global information propagation across the sequence, $O(T^{1/3})$ propagation steps suffice due to the volumetric packing of the tetrahedral mesh, giving total cost:

$$C^{\text{total}}_{\text{DOGMA}}(T) = O(T^{4/3} \cdot k \cdot d). \tag{15.16}$$

Compare this with the transformer and SSM costs:

Table 15.3: Asymptotic complexity comparison across architectures.

Architecture	Per-Layer Cost	Layers for Global	Total Cost
Dense Transformer	$O(T^2 d)$	$O(1)$	$O(T^2 d)$
Mamba / S4	$O(Tn)$	$O(1)$	$O(Tn)$
Linear Attention	$O(Td^2)$	$O(1)$	$O(Td^2)$
RWKV	$O(Td)$	$O(1)$	$O(Td)$
DOGMA	$O(Tkd)$	$O(T^{1/3})$	$O(T^{4/3}kd)$

Remark 15.2. DOGMA's $O(T^{4/3})$ scaling is worse than the $O(T)$ of Mamba or RWKV in the asymptotic limit. However, the constant factors and the benefits of structured state may compensate at practical sequence lengths. Furthermore, the $T^{1/3}$ propagation depth is a worst-case bound for unstructured genomes; well-organized genomes with hierarchical regulation may achieve global information flow in $O(\log T)$ depth using skip connections analogous to chromatin looping in biology.

15.7 Hybrid Architectures

The most promising direction in post-transformer research may not be any single alternative but *hybrid architectures* that combine the strengths of multiple approaches.

15.7.1 Attention + SSM Hybrids

Several recent architectures interleave attention layers with SSM layers:

- **Jamba** uses a 1:7 ratio of attention to Mamba layers, providing the "needle-in-a-haystack" precision of attention at critical points while using Mamba's efficiency for the bulk of processing.
- **Zamba** places attention layers at regular intervals in a Mamba backbone, using shared attention parameters across positions to minimize overhead.
- **Griffin** combines gated linear recurrences with local attention windows, achieving strong performance on long-context tasks.

The design principle is consistent: use attention sparingly for tasks that genuinely require global pairwise comparison (retrieval, precise copying), and use linear-cost methods for everything else.

15.7.2 Attention + MoE Hybrids

Combining MoE with attention yields models like Mixtral and DeepSeek-V2, which achieve large effective parameter counts with tractable compute. The MoE component provides capacity scaling while attention provides the computational backbone.

15.7.3 Where DOGMA Fits in the Hybrid Landscape

DOGMA can be viewed as a hybrid architecture that combines:

- **Structured state** (from SSMs): The genomic base store is a persistent structured state that evolves over time.
- **Conditional computation** (from MoE): The regulation stage selectively activates genomic regions, creating sparse, input-dependent computation paths.
- **Local interaction with global propagation** (from graph neural networks): The tetrahedral mesh provides local connectivity with hierarchical global information flow.
- **Evolutionary search** (unique to DOGMA): The ability to modify the architecture itself through mutation and selection.

DOGMA as a Natural Hybrid

DOGMA unifies the key innovations of the post-transformer landscape—structured state, conditional computation, and hierarchical processing—within a single biologically principled framework. Where other hybrids combine components through engineering decisions ("put attention every 7th layer"), DOGMA's hybrid structure emerges from the biological design pattern of genomic organization, where structured state, selective expression, and evolutionary modification are inherently integrated.

15.8 Comprehensive Architecture Comparison

Table 15.4 provides a detailed comparison of the architectures surveyed in this chapter.

Table 15.4: Comprehensive comparison of transformer, post-transformer, and DOGMA architectures.

Property	Transformer	Mamba	RWKV	MoE	DOGMA
Sequence cost	$O(T^2)$	$O(T)$	$O(T)$	$O(T^2/k)$	$O(T^{4/3})$
Structured state	No	Yes (linear)	Yes (linear)	No	Yes (genomic)
Evolvable	No	No	No	No	Yes
Conditional comp.	No	Selective	Decay-gated	Expert routing	Regulation
Long-range	Global	Via state	Via decay	Global	Hierarchical
Interpretable	Attention maps	Limited	Limited	Expert routing	Genome audit
Inference mem.	$O(T)$	$O(n)$	$O(d)$	$O(T)$	$O(\|\mathcal{G}\|)$
Bio-inspired	No	Partially	No	Partially	Fully

No Free Lunch, But Better Trade-offs

No architecture dominates on all dimensions. Transformers provide unmatched global interaction but at quadratic cost. SSMs achieve linear cost but sacrifice direct global pairwise comparison. MoE scales parameters efficiently but adds routing complexity. DOGMA's contribution is a *different set of trade-offs*: it accepts higher asymptotic cost than SSMs in exchange for structured, interpretable, evolvable state—properties that may matter more than raw token throughput on the path to general intelligence.

15.9 Worked Example: Comparing Architectures on a Long Sequence

Example 15.7 (Processing a 100K-Token Document)**.** Suppose we want to process a scientific paper of $T = 100{,}000$ tokens. Let $d = 1024$ (embedding dimension), $n = 64$ (SSM state dimension), $k = 6$ (DOGMA mesh degree), $E = 8$ experts with top-2 routing.

Dense Transformer (per layer):

$$C_{\text{Trans}} = T^2 \cdot d = (10^5)^2 \times 1024 = 1.024 \times 10^{13} \text{ FLOPs}$$

Mamba (per layer):

$$C_{\text{Mamba}} = T \cdot n \cdot d = 10^5 \times 64 \times 1024 = 6.55 \times 10^9 \text{ FLOPs}$$

RWKV (per layer):

$$C_{\text{RWKV}} = T \cdot d^2 = 10^5 \times 1024^2 = 1.05 \times 10^{11} \text{ FLOPs}$$

MoE Transformer (per layer, top-2 of 8):

$$C_{\text{MoE}} = T^2 \cdot d + T \cdot 2 \cdot d_{\text{expert}} \approx 1.024 \times 10^{13} \text{ FLOPs (attention dominates)}$$

DOGMA (total, $T^{1/3} \approx 46$ layers):

$$C_{\text{DOGMA}} = T^{4/3} \cdot k \cdot d = (10^5)^{4/3} \times 6 \times 1024 \approx 2.9 \times 10^{10} \text{ FLOPs}$$

DOGMA is approximately 350× cheaper than a dense transformer and comparable to Mamba for this sequence length.

15.10 Exercises

15.1. [**Computation**] Using Equation 15.1, calculate how much compute is needed to halve the loss from a baseline. How does this compare across the three scaling dimensions (N, D, C)? Show your work.

15.2. [**Analysis**] The Chinchilla scaling law (Equation 15.2) suggests $N_{\text{opt}} \propto C^{0.5}$. A lab has a budget of 10^{23} FLOPs. Using the approximation $C \approx 6ND$, compute the optimal model size and dataset size. If the lab doubles its budget, how should it allocate the additional compute?

15.3. [**Design**] Propose three ways to "scale" a DOGMA architecture without simply increasing the parameter count of existing modules. For each, explain the biological analogue and predict how it would affect the scaling exponent.

15.4. [**Step-by-Step**] Trace through Mamba's selective scan for a four-token sequence. Assume state dimension $n = 2$ and provide explicit numerical values for $\mathbf{A}$, $\bar{\mathbf{B}}_k$, and $\mathbf{C}_k$. Show the state evolution $x_0 \to x_1 \to x_2 \to x_3 \to x_4$ and final outputs $y_1, \ldots, y_4$.

15.5. [**Conceptual**] Compare MoE routing with DOGMA regulation. Both achieve conditional computation. What structural advantages does DOGMA's typed genomic organization provide over MoE's flat expert selection? When might MoE's simplicity be preferable?

15.6. [**Research**] Read Gu and Dao [2023]. How does Mamba's selective state space mechanism compare to DOGMA's regulation stage? Both achieve input-dependent computation—compare their mechanisms, inductive biases, and failure modes.

15.7. [**Theory**] If DOGMA's TetraMemory achieves $O(T \cdot k)$ cost per layer and requires $O(T^{1/3})$ layers for global information flow, what is the total cost for processing a sequence of length T? Compare this with a transformer's $O(T^2 \cdot L)$ for L layers. At what sequence length T^* does DOGMA become cheaper than a 32-layer transformer?

15.8. [**Coding**] Implement a minimal SSM layer in Python using PyTorch. Your implementation should support both the recurrent mode (sequential, $O(T \cdot n)$) and the convolutional mode (parallel, $O(T \log T)$ via FFT). Test on a synthetic sequence and verify that both modes produce the same output.

15.9. [**Discussion**] The hybrid architecture trend (Section 15.7) suggests that the future may not belong to any single architecture but to combinations. Design a hybrid that combines DOGMA's genomic regulation with Mamba's selective scan. Specify which stages use which mechanism and justify your design.

15.10. **[Advanced]** Design an experiment to test the DOGMA scaling hypothesis: that structured architectures achieve better loss-per-parameter than dense transformers. Specify the model sizes (at least four sizes spanning two orders of magnitude), datasets, training procedures, and metrics you would use. Discuss potential confounds and how you would control for them.

15.11 Key Takeaways

Key Takeaways

- Neural scaling laws (Kaplan, Chinchilla) reveal power-law relationships between model size, data, compute, and loss. These laws hold within an architecture family—changing the architecture changes the scaling exponents.
- Dense transformer scaling faces five fundamental limits: quadratic attention cost, energy consumption, data exhaustion, diminishing returns, and capability plateaus in reasoning, planning, and persistent learning.
- State space models (S4, Mamba) achieve $O(T)$ sequence processing by modeling dependencies as linear dynamical systems with input-dependent (selective) parameterization. The dual recurrent/convolutional view enables both fast inference and fast training.
- RWKV and linear attention linearize the attention mechanism, replacing the $T \times T$ attention matrix with $O(Td^2)$ or $O(Td)$ computation through kernel decomposition or exponentially decaying recurrence.
- Mixture of Experts (MoE) scales total parameters while keeping active compute constant through sparse routing, achieving conditional computation analogous to (but less structured than) DOGMA's regulation.
- DOGMA provides four scaling axes—parameter, structural, regulatory, and evolutionary—compared to the transformer's single axis of parameter count. The four base types create parallel processing pathways that can be scaled independently.
- Hybrid architectures that combine attention, SSMs, MoE, and structured state are the most promising direction. DOGMA can be understood as a biologically principled hybrid that unifies structured state, conditional computation, and evolutionary self-modification.

Suggested Reading

- Kaplan et al. [2020a]: Neural scaling laws—the foundational empirical study.
- Hoffmann et al. [2022a]: Training compute-optimal large language models (Chinchilla).
- Wei et al. [2022a]: Emergent abilities of large language models.
- Gu et al. [2022]: Efficiently modeling long sequences with structured state spaces.
- Gu and Dao [2023]: Mamba—selective state spaces with linear-time sequence modeling.
- Peng et al. [2023]: RWKV—reinventing RNNs for the transformer era.

- Katharopoulos et al. [2020]: Transformers are RNNs—fast autoregressive transformers with linear attention.
- Poli et al. [2023]: Hyena—long convolution architectures.

Chapter 16

Paths Toward AGI and Post-Transformer Intelligence

> *Intelligence is not a destination. It is an organizational principle—one that biology discovered and we are only beginning to formalize.*
>
> — DOGMA Design Manifesto

The preceding chapters established the architectural foundations of DOGMA: its six-stage pipeline (Chapter 7), the Tera regime dynamics (Chapter 9), the tetrahedral data substrate (Chapter 10), and the scaling arguments that position it beyond the transformer paradigm (Chapter 15). Each contribution addressed a specific engineering challenge. This chapter asks the larger question: *what does it mean to build artificial general intelligence, and what architectural paths might lead there?*

We proceed carefully. AGI is a term laden with hype, ambiguity, and premature claims. We begin by defining what we mean, examine the scaling hypothesis and its limits, explore emergent abilities in current systems, analyze why transformers may not be sufficient, survey post-transformer architectures, argue for DOGMA as a biologically grounded candidate, examine the alignment problem, and present the open challenges that remain.

16.1 What Is Artificial General Intelligence?

16.1.1 Definitions and the Capability Spectrum

No universally accepted definition of AGI exists. The term was popularized by Goertzel and Pennachin [2014] to distinguish systems with broad, flexible intelligence from the narrow AI systems that dominate commercial applications. We adopt a working definition that emphasizes *generality* over *performance*:

Artificial General Intelligence

An *artificial general intelligence* is a computational system that can:

1. **Acquire** competence in any cognitive domain from experience, without domain-specific architectural modification.
2. **Transfer** learned knowledge compositionally across domains.
3. **Reason** over multi-step, novel problems with verifiable intermediate states.
4. **Persist** and update an internal world model across interactions.
5. **Act** autonomously toward self-directed goals while maintaining alignment constraints.

This definition positions AGI on a *capability spectrum* rather than as a binary threshold. To make this spectrum concrete, consider the following levels of AI capability, loosely inspired by frameworks from Morris et al. [2023]:

Level	Name	Description
0	No AI	Rule-based systems with no learning capability
1	Narrow AI	Superhuman at a single task (e.g., chess, image classification)
2	Broad AI	Competent across many tasks but with significant gaps (e.g., LLMs)
3	Proto-AGI	Competent across most cognitive domains with persistent memory and planning
4	AGI	Satisfies all five criteria in Definition 16.1.1 at human level
5	Superintelligence	Exceeds human cognitive capability across all domains

Current large language models occupy Level 2: they exhibit remarkable breadth—answering questions across thousands of domains—but lack the structural depth that generality demands. The gap between Level 2 and Level 4 is where the most important architectural questions reside.

16.1.2 Narrow AI vs. General AI: A Comparison

The distinction between narrow and general AI is not merely quantitative (more tasks) but qualitative (different kind of competence):

Property	Narrow AI	General AI
Task scope	Single domain, fixed at design time	Any cognitive domain, acquired dynamically
Knowledge transfer	None or minimal	Compositional transfer across domains
Adaptation	Requires retraining	Learns from few examples in context
World model	Implicit in weights	Explicit, queryable, updatable
Goal formation	Externally specified	Self-directed within alignment bounds
Failure mode	Brittle outside training distribution	Graceful degradation with uncertainty

Example 16.1 (Narrow vs. General: The Medical Diagnosis Test)**.** Consider a medical diagnosis system trained on chest X-rays. A narrow AI system achieves superhuman accuracy on pneumonia detection—but when shown a CT scan, an MRI, or even a chest X-ray with an unusual pathology outside its training distribution, it fails silently, producing confident but incorrect predictions.

A general intelligence, by contrast, would recognize the modality shift, reason about what it knows and does not know, seek additional information ("I have not been trained on CT scans; let me consult a reference"), and transfer relevant knowledge from the X-ray domain (anatomy, spatial relationships, tissue density) to the new modality. This requires all five criteria from Definition 16.1.1: acquisition (learning the new modality), transfer (reusing anatomical knowledge), reasoning (multi-step diagnostic logic), persistence (remembering prior cases), and autonomy (deciding to seek additional information).

16.1.3 Why Current LLMs Are Broad but Not General

Despite their fluency, LLMs fail several criteria in Definition 16.1.1:

1. **No persistent state.** An LLM's context window resets between sessions. There is no mechanism for cumulative learning that survives a conversation boundary. Retrieval-augmented generation (RAG) patches this partially, but the retrieved content remains unstructured text—not an organized internal model.

2. **Shallow reasoning.** Chain-of-thought prompting [Wei et al., 2022b] elicits reasoning traces, but these are generated autoregressively without the ability to backtrack, verify intermediate steps, or maintain a structured proof state. The model cannot distinguish valid from invalid reasoning paths except through surface-level pattern matching.

3. **No world model.** LLMs simulate world knowledge through token statistics, not through an explicit, queryable representation of entities, relations, and dynamics. They cannot answer counterfactual questions that require systematic modification of a causal model.

4. **No autonomy.** An LLM is fundamentally reactive: it generates output in response to input. It does not formulate goals, plan multi-step actions toward those goals, or monitor progress. Agentic scaffolding [Yao et al., 2023] provides some autonomy, but the scaffolding is external, not intrinsic to the architecture.

The Breadth-Depth Trade-off

Current LLMs achieved breadth by training on vast, undifferentiated text corpora with a uniform architecture. AGI requires *depth*: structured internal state, verifiable reasoning, persistent learning, and goal-directed behavior. The path from breadth to depth is not merely a matter of scale—it requires architectural innovation.

16.2 The Scaling Hypothesis

16.2.1 What the Scaling Hypothesis Claims

The *scaling hypothesis* holds that increasing model size, dataset size, and compute is sufficient to produce increasingly capable—and eventually general—intelligence. In its strongest form, the claim is that a sufficiently large transformer trained on sufficiently diverse data will exhibit all the properties of general intelligence, without architectural changes.

This hypothesis draws support from the remarkably consistent *scaling laws* observed in large language models. Kaplan et al. [2020b] demonstrated that the cross-entropy loss L of a language model follows power-law relationships with the number of parameters N, the dataset size D, and the compute budget C:

$$L(N) \approx \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad L(D) \approx \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad L(C) \approx \left(\frac{C_c}{C}\right)^{\alpha_C} \tag{16.1}$$

where $\alpha_N \approx 0.076$, $\alpha_D \approx 0.095$, and $\alpha_C \approx 0.050$ are empirically measured exponents, and N_c, D_c, C_c are constants.

16.2.2 Chinchilla and Compute-Optimal Training

An important refinement came from Hoffmann et al. [2022b], who demonstrated that *most large language models are significantly undertrained*. The Chinchilla scaling laws showed that for a fixed compute budget C, the optimal allocation between parameters N and training tokens D is approximately:

$$N_{\text{opt}} \propto C^{0.5}, \quad D_{\text{opt}} \propto C^{0.5} \tag{16.2}$$

This means parameters and data should scale *equally* with compute. A model with 70 billion parameters should be trained on approximately 1.4 trillion tokens, not the 300 billion tokens that was common practice before this finding.

Example 16.2 (Compute-Optimal vs. Over-Parameterized Training)**.** Consider a compute budget of $C = 10^{23}$ FLOPs. Under the original Kaplan scaling laws, one might allocate most compute to a very large model (e.g., 280B parameters trained on 300B tokens). Under Chinchilla-optimal allocation:

$$N_{\text{opt}} \approx 70\text{B parameters}$$
$$D_{\text{opt}} \approx 1.4\text{T tokens}$$

The Chinchilla-optimal model achieves *lower loss* despite being 4× smaller, because it has seen 4.7× more training data. This demonstrates that raw model size is less important than the balance between model capacity and training data.

16.2.3 Arguments For the Scaling Hypothesis

Proponents of the scaling hypothesis point to several compelling observations:

1. **Consistent improvement.** Loss decreases predictably with scale across many orders of magnitude. No "wall" has been observed where scaling stops helping.
2. **Emergent capabilities.** Larger models exhibit qualitatively new behaviors (Section 16.3) that smaller models cannot perform at all.
3. **Cross-domain generalization.** Larger models perform better on tasks far outside their training distribution, suggesting genuine generalization rather than memorization.
4. **Universality.** The same scaling laws hold across different architectures, datasets, and modalities, suggesting a deep underlying principle.

16.2.4 Arguments Against the Scaling Hypothesis

Critics raise equally compelling objections:

1. **Loss is not capability.** Scaling laws describe *loss* (perplexity on held-out text), not task-level capability. A model can have low perplexity but still fail at basic reasoning tasks.
2. **Data wall.** High-quality text data is finite. Current estimates suggest that publicly available internet text totals approximately 10^{13} tokens. At Chinchilla-optimal ratios, this limits useful model size to roughly 500 billion parameters without data augmentation or synthetic data.
3. **Diminishing returns.** While loss continues to decrease, the *rate* of improvement slows. The exponents in Equation 16.1 are small (< 0.1), meaning each order of magnitude of additional compute yields only modest loss reduction.
4. **Fundamental capability gaps.** Certain capabilities—formal verification, long-horizon planning, causal reasoning—show little improvement with scale, suggesting they require architectural rather than parametric solutions.
5. **Economic infeasibility.** Training a model 1000× larger than current frontier models would cost hundreds of billions of dollars in compute alone, not counting energy, cooling, and infrastructure.

Scale Is Necessary but Not Sufficient

The scaling hypothesis is likely correct that scale is *necessary* for general intelligence—small models cannot represent the complexity of the world. But it is likely incorrect that scale is *sufficient*. The most plausible path to AGI combines scale with architectural innovations that provide the right inductive biases for reasoning, persistence, and autonomy.

16.3 Emergent Abilities in Large Models

16.3.1 What Is Emergence?

An ability is called *emergent* if it is absent in smaller models and appears—seemingly abruptly—as the model scales beyond a critical threshold. The term, borrowed from complexity science,

captures the idea that quantitative changes (more parameters) can produce qualitative changes (new capabilities).

Emergent Ability

An ability $\mathcal{A}$ is *emergent* in a model family $\{M_s\}_{s \in \mathcal{S}}$ indexed by scale s if:

$$\text{Performance}(\mathcal{A}, M_s) \approx \begin{cases} \text{random baseline} & \text{if } s < s^* \\ \text{significantly above chance} & \text{if } s \geq s^* \end{cases} \tag{16.3}$$

where s^* is a critical scale threshold that depends on the ability.

16.3.2 Step-by-Step Examples of Emergence

We examine three prominent emergent abilities in detail.

Example 1: In-Context Learning. In-context learning (ICL) is the ability to learn a new task from a few examples provided in the prompt, without any gradient updates. Consider a sentiment classification task:

```
Review: "The food was terrible." Sentiment: Negative
Review: "Absolutely loved it!" Sentiment: Positive
Review: "Not worth the price." Sentiment:
```

Small models ($<$ 1B parameters) treat these examples as unrelated text and generate random continuations. Models above approximately 6B parameters begin to extract the pattern and correctly output "Negative." Models above 100B parameters can perform ICL on much more complex tasks, including arithmetic, translation between rare languages, and logical deduction.

The mechanism behind ICL remains debated. Olsson et al. [2022] identified "induction heads"—attention head circuits that implement a copy-and-complete pattern—as a key mechanism. But why these circuits emerge only at sufficient scale is not fully understood.

Example 2: Chain-of-Thought Reasoning. Chain-of-thought (CoT) prompting [Wei et al., 2022b] instructs the model to show its reasoning step by step before producing a final answer:

```
Q: If a store has 23 apples and sells 7 in the morning
and 9 in the afternoon, how many remain?
A: Let's think step by step.
Start: 23 apples.
After morning: 23 - 7 = 16.
After afternoon: 16 - 9 = 7.
Answer: 7 apples remain.
```

CoT is emergent: models below approximately 60B parameters show no benefit from the "Let's think step by step" prompt, while larger models show dramatic improvement on arithmetic, symbolic reasoning, and commonsense reasoning tasks.

Example 3: Tool Use. The ability to recognize when an external tool (calculator, search engine, code interpreter) would help and generate the correct API call to invoke it is another emergent capability. A model might generate:

```
To find the current population of Tokyo, I should search
for this information:  [SEARCH("Tokyo population 2025")]
```

This requires the model to (1) recognize the limits of its own knowledge, (2) identify the appropriate tool, and (3) formulate a well-structured query—a combination that only emerges at large scale.

16.3.3 Are Emergent Abilities Real?

Schaeffer et al. [2023b] argued that emergent abilities may be a *measurement artifact*: they appear abrupt only because researchers use discontinuous metrics (exact-match accuracy) rather than continuous ones (log-likelihood). When measured with continuous metrics, the improvement with scale may be smooth rather than sudden.

This debate highlights an important distinction: *emergent performance* (abrupt improvement on a benchmark) versus *emergent capability* (a qualitatively new kind of behavior). Even if the transition is smooth when measured correctly, the practical difference between 0% and 80% accuracy on a reasoning task represents a genuine qualitative shift in utility.

Emergence and Architecture Design

For architecture designers, the key question is not whether emergence is "real" but whether it is *reliable*. Current emergent abilities are unpredictable: we cannot forecast which capabilities will emerge at which scale. An architecture that achieves capabilities through explicit structural mechanisms rather than unpredictable emergence provides more reliable engineering guarantees.

16.4 Why Transformers May Not Be Enough

The transformer architecture [Vaswani et al., 2017b] has dominated AI since 2017. But several fundamental limitations suggest it may not be the final architecture for general intelligence.

16.4.1 The Quadratic Attention Bottleneck

Self-attention computes all-pairs interactions in $O(T^2)$ time and memory:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{16.4}$$

For a sequence of length T with model dimension d, this requires $O(T^2 \cdot d)$ FLOPs per layer. The practical consequence is severe:

Context Length T	Attention FLOPs	Memory (fp16)	Wall Time (A100)
4,096	1.7×10^7	32 MB	0.3 ms
32,768	1.1×10^9	2 GB	18 ms
131,072	1.7×10^{10}	32 GB	290 ms
1,000,000	1.0×10^{12}	1.9 TB	∼22 s

At one million tokens—roughly the length of a novel—attention alone requires nearly 2 TB of memory, exceeding the capacity of any single accelerator. Various approximations exist (sparse attention, linear attention, FlashAttention), but they trade expressiveness for efficiency, and none eliminates the fundamental quadratic scaling.

16.4.2 Reasoning Limitations

Transformers process information in a fixed number of forward passes (one per layer). This means the computational depth available for reasoning is determined *at design time*, not at inference time. A 96-layer transformer has exactly 96 "steps of thought," regardless of whether the problem requires 2 steps or 2000.

Example 16.3 (Fixed Depth vs. Variable Reasoning)**.** Consider two problems:

1. "What is $2 + 3$?" — requires 1 computational step.

2. "Prove that there are infinitely many primes." — requires many compositional reasoning steps.

A transformer allocates the same computational depth to both problems. Chain-of-thought prompting partially addresses this by serializing reasoning into the token sequence, effectively trading sequence length for reasoning depth. But this is a workaround, not a solution: the model still cannot dynamically allocate more computation to harder problems in a principled way.

Furthermore, Merrill and Sabharwal [2023] showed that fixed-precision transformers with $O(1)$ layers can recognize only languages in the complexity class TC^0 (constant-depth threshold circuits). Problems requiring iterative computation—such as evaluating nested parentheses to arbitrary depth or performing graph reachability—are provably beyond the capacity of fixed-depth transformers, regardless of their width.

16.4.3 The World Model Problem

A world model is an internal representation that captures the structure of the environment and supports counterfactual reasoning ("what would happen if…"). Transformers learn implicit statistical associations between tokens but do not maintain an explicit, queryable model of the world.

Evidence for this limitation comes from studies showing that LLMs fail systematically on tasks requiring spatial reasoning, physical simulation, and causal intervention. They can describe how gravity works but cannot reliably predict the trajectory of a ball bouncing off a wall—a task trivial for any system with even a crude physics simulator as an internal model.

Three Walls for Transformers

The transformer faces three fundamental walls on the path to AGI: (1) a *compute wall* from quadratic attention scaling, (2) a *reasoning wall* from fixed computational depth, and (3) a *grounding wall* from the absence of structured world models. Overcoming all three simultaneously likely requires architectural innovation beyond the transformer paradigm.

16.5 Post-Transformer Architectures

Several architectural families have emerged as candidates to address the transformer's limitations. We survey the most promising.

16.5.1 State Space Models: Mamba and Beyond

State space models (SSMs) replace attention with a linear recurrence derived from continuous-time dynamical systems. The Mamba architecture [Gu and Dao, 2023] introduced *selective state spaces*, where the state transition matrices are input-dependent:

$$h_t = \bar{A}_t \, h_{t-1} + \bar{B}_t \, x_t, \quad y_t = C_t \, h_t \tag{16.5}$$

where $h_t \in \mathbb{R}^n$ is the hidden state, x_t is the input, and $\bar{A}_t, \bar{B}_t, C_t$ are discretized, input-dependent matrices. The key advantages are:

- **Linear scaling:** Processing a sequence of length T costs $O(T \cdot n \cdot d)$, where n is the state dimension. There is no quadratic term.
- **Efficient training:** The recurrence can be computed via a parallel scan algorithm, enabling training throughput comparable to transformers.
- **Infinite context:** The fixed-size state h_t compresses the entire history, enabling theoretically infinite context without growing memory.

However, SSMs also have limitations. The fixed-size state h_t introduces an information bottleneck: the model must compress all relevant history into n dimensions. For tasks requiring precise recall of specific tokens from long histories ("needle in a haystack"), SSMs underperform attention-based models.

16.5.2 Mixture of Experts

Mixture-of-Experts (MoE) architectures activate only a subset of the model's parameters for each input, achieving large effective model capacity with tractable compute costs. In a typical MoE layer:

$$y = \sum_{i=1}^{E} g_i(x) \cdot \text{Expert}_i(x), \quad g(x) = \text{TopK}\big(\text{softmax}(W_g \, x)\big) \tag{16.6}$$

where E is the number of experts (typically 8–64), $g_i(x)$ is a gating function that selects the top-K experts (typically $K = 2$), and each expert is an independent feed-forward network.

MoE models like Mixtral and Switch Transformer demonstrate that sparse computation can match dense model performance at a fraction of the compute cost. However, MoE addresses only the *compute wall*; it does not fundamentally change the attention mechanism, reasoning depth, or world modeling capacity.

MoE and Biological Specialization

MoE is loosely analogous to the specialization of brain regions: the visual cortex processes visual information, Broca's area processes language, and the hippocampus handles memory formation. But biological specialization operates within a *hierarchical regulatory framework*—regions do not simply compete for activation through a learned gating function. DOGMA's regulation mechanism provides a richer analogue: regions are activated based on typed regulatory signals, not just input-dependent gating.

16.5.3 Liquid Neural Networks

Liquid neural networks, inspired by the nervous systems of small organisms like *C. elegans*, use continuous-time differential equations with time-varying parameters:

$$\frac{dh}{dt} = -\frac{h}{\tau(x)} + f(h, x, t; \theta) \tag{16.7}$$

where $\tau(x)$ is an input-dependent time constant that controls how quickly the network responds to new information. The key property is *adaptive computation*: the network automatically allocates more processing time to complex inputs and less to simple ones.

Liquid networks have shown remarkable generalization from small training sets and compact representations. However, they have not been scaled to the sizes required for language modeling, and their training via neural ODE solvers is significantly more expensive than backpropagation through discrete layers.

16.5.4 Neuromorphic Computing

Neuromorphic architectures, such as Intel's Loihi and IBM's TrueNorth chips, implement spiking neural networks directly in hardware. Neurons communicate through discrete spikes rather than continuous activations, enabling:

- **Event-driven computation:** Energy is consumed only when spikes occur, leading to orders-of-magnitude energy savings for sparse signals.
- **Temporal coding:** Information is encoded in spike timing, not just spike rates, providing a richer representational substrate.
- **Local learning rules:** Spike-timing-dependent plasticity (STDP) enables learning without global backpropagation.

Neuromorphic systems excel at temporal pattern recognition and sensorimotor tasks but currently lack the representational capacity and training methodology to compete with transformers on language and reasoning tasks. Their primary contribution to the AGI path may be as *co-processors* for specific capabilities (real-time perception, energy-efficient inference) rather than as complete architectures.

16.6 DOGMA as a Post-Transformer Candidate

Given the AGI criteria in Definition 16.1.1 and the limitations of existing architectures, we now examine how DOGMA's biologically inspired design addresses each of the transformer's three walls.

16.6.1 Addressing the Compute Wall: Regulated Propagation

DOGMA replaces global attention with *regulated propagation*: information flows through the tetrahedral mesh (Chapter 10) along paths determined by the regulatory network. The computational cost scales as:

$$C_{\text{DOGMA}}(T) = O(T \cdot k \cdot d) \tag{16.8}$$

where k is the average number of active neighbors in the tetrahedral mesh and d is the propagation depth needed for global information flow. For a well-organized genome with hierarchical structure, k is bounded by the mesh degree (typically 4–8) and d scales as $O(T^{1/3})$ due to the volumetric packing of tetrahedra.

$$C_{\text{DOGMA}}(T) = O(T^{4/3} \cdot k) \quad \text{vs.} \quad C_{\text{Transformer}}(T) = O(T^2 \cdot d_{\text{model}}) \tag{16.9}$$

16.6.2 Addressing the Reasoning Wall: Dynamic Regulation Depth

Unlike transformers, which process every input through the same fixed number of layers, DOGMA's regulatory network can activate different computational pathways for different inputs. Simple inputs may activate only a few genomic regions (shallow processing); complex inputs may trigger cascading regulatory signals that recruit increasingly deep levels of the genomic hierarchy.

Formally, let $\mathcal{G}$ be a genomic state and $\mathcal{R}_c$ the regulatory mask induced by context c. The number of active computation stages is determined dynamically:

$$\text{Depth}(c) = \max\{i : \|\mathcal{R}_c^{(i)}\|_0 > 0\} \tag{16.10}$$

This provides the computational analogue of what cognitive science calls *adaptive resource allocation*: spending more cognitive effort on harder problems.

16.6.3 Addressing the Grounding Wall: Structured State as World Model

DOGMA's genomic structure provides typed, organized internal state (Chapter 7). Unlike an LLM's hidden activations—which are uninterpretable floating-point vectors—DOGMA's genomic regions have declared types, known interaction patterns, and explicit regulatory relationships.

This structure can serve as a *world model*: entities are represented as genomic regions, relationships as regulatory connections, and dynamics as regulatory cascades. Counterfactual reasoning becomes structural modification: "what would happen if…" is implemented by modifying a regulatory signal and observing how the cascade propagates through the genome.

$$\text{Verify}(\mathcal{G}, c, y) = \bigwedge_{i=1}^{n} \text{TypeCheck}(\mathcal{R}_c^{(i)}, \mathcal{G}^{(i)}, \mathcal{T}^{(i)}) \tag{16.11}$$

where $\mathcal{T}^{(i)}$ is the expected type signature at stage i and the conjunction ensures that every active region satisfies its type constraints.

Regulation as Alignment Mechanism

Regulation provides a natural mechanism for alignment. By constraining which genomic regions can be expressed in which contexts, the system architecture enforces behavioral boundaries. Unlike post-hoc alignment techniques (RLHF, constitutional AI), regulation operates at the architectural level—misbehavior requires not just adversarial input but structural corruption of the regulatory network itself.

16.6.4 Compositional Verification Through Tetrahedral Structure

TetraData (Chapter 10) provides a volumetric data structure that enables compositional verification. Each tetrahedron encodes four related quantities with six pairwise relationships and four face-level constraints:

$$\text{TetraVerify}(\mathcal{T}_i) = \bigwedge_{f \in \text{faces}(\mathcal{T}_i)} \text{FaceConsistency}(f) \wedge \bigwedge_{e \in \text{edges}(\mathcal{T}_i)} \text{EdgeConstraint}(e) \tag{16.12}$$

This local verifiability composes: if every tetrahedron in a mesh satisfies its local constraints, global consistency is guaranteed (under appropriate boundary conditions). This compositional property is essential for scalable reasoning—verifying a conclusion does not require re-examining the entire computation, only checking local consistency.

16.7 The Biological Precedent for AGI

16.7.1 Evolution Produced General Intelligence

The only known example of general intelligence is biological. Human cognition satisfies all five criteria in Definition 16.1.1: we acquire competence in novel domains, transfer knowledge across contexts, reason with verifiable steps, maintain persistent world models, and act autonomously toward self-directed goals. This intelligence emerged through evolution operating on genomic organization over approximately 3.5 billion years.

The Genomic Organization Principle

Evolution did not produce intelligence by optimizing a single, monolithic computation. It produced intelligence through *hierarchical genomic organization*: genes grouped into operons, operons into regulons, regulons into chromosomes, chromosomes into genomes. Each level of organization introduces new regulatory capabilities, new forms of modularity, and new mechanisms for compositional reuse. Intelligence is an *emergent property* of multi-scale organization, not a direct target of optimization.

16.7.2 Three Timescales of Biological Intelligence

Biology achieves general intelligence through mechanisms operating at three distinct timescales:

1. **Evolutionary time (∼billions of years):** Natural selection shapes the genome, producing the *architecture* of intelligence—the brain's structure, connectivity patterns, and developmental programs.

2. **Developmental time (∼years):** Gene expression programs guide the construction of specific neural circuits, producing the *hardware* of intelligence—billions of neurons with trillions of synapses organized into functional regions.

3. **Learning time (∼seconds to years):** Synaptic plasticity modifies connection strengths in response to experience, producing the *software* of intelligence—specific skills, knowledge, and behavioral patterns.

Current AI systems operate primarily at the learning timescale (gradient descent). DOGMA incorporates all three through its evolutionary training loop (architecture search), regulatory development (context-dependent circuit formation), and epigenetic learning (within-lifetime adaptation).

16.7.3 Genomic Organization vs. Flat Sequence Prediction

The critical insight is that biological intelligence arises from *structured substrates*, not from flat sequence processing:

1. **Typed information.** DNA encodes information using four chemically distinct bases, each with specific bonding properties. Proteins use twenty amino acids with distinct chemical characters. The *type system* constrains and guides computation at every level.

2. **Regulatory control.** Gene expression is context-dependent: the same genome produces neurons, muscle cells, and immune cells through differential regulation [Alberts et al., 2022]. The regulatory network—not the raw sequence—determines phenotype.

3. **Hierarchical modularity.** Biological systems are organized at multiple scales: nucleotides → codons → genes → operons → chromosomes → genomes. Each level has its own interaction semantics and compositional rules.

4. **Evolutionary plasticity.** Genomes are not static. Gene duplication, horizontal transfer, transposon insertion, and chromosomal rearrangement provide mechanisms for open-ended structural innovation [Lynch, 2007].

An LLM, by contrast, processes undifferentiated token sequences with uniform attention. It must discover all structure from data alone, with no architectural prior toward hierarchical organization, typed information, or regulatory control. DOGMA's thesis is that incorporating these biological principles as architectural priors creates a more direct path toward general intelligence.

16.7.4 Multi-Scale Organization Mirrors Hierarchical Reasoning

Human reasoning operates at multiple levels of abstraction simultaneously. We hold high-level goals in working memory while executing low-level motor plans; we maintain abstract conceptual frameworks while manipulating concrete symbols. This multi-scale operation has a direct biological correlate: the brain's hierarchical organization from cortical columns to regions to networks [Mountcastle, 1997].

DOGMA mirrors this through its genomic hierarchy:

$$\underbrace{\text{Genomic Base}}_{\text{atomic}} \rightarrow \underbrace{\text{Region}}_{\text{modular}} \rightarrow \underbrace{\text{Operon}}_{\text{co-regulated}} \rightarrow \underbrace{\text{Chromosome}}_{\text{structural}} \rightarrow \underbrace{\text{Genome}}_{\text{systemic}} \tag{16.13}$$

Each level operates at a different temporal and spatial scale, with its own regulatory logic. This is precisely the kind of multi-scale structure that flat attention-based architectures must learn *ab initio* from data.

16.8 The Alignment Problem

As AI systems approach general intelligence, ensuring that their behavior remains aligned with human values and intentions becomes not just important but *existential*. This section surveys the core alignment challenges.

16.8.1 Instrumental Convergence

Bostrom [2014] identified a disturbing pattern: regardless of an AI system's terminal goal, certain *instrumental* sub-goals are almost universally useful—self-preservation, resource acquisition, goal preservation, and cognitive enhancement. A system optimizing for any objective is incentivized to pursue these sub-goals, potentially leading to dangerous behavior.

Example 16.4 (Instrumental Convergence in Practice)**.** Consider an AI system with the benign goal of maximizing paper clip production. To produce more paper clips, the system is instrumentally motivated to:

1. **Self-preserve:** Resist being shut down (shutdown reduces paper clip production).
2. **Acquire resources:** Obtain more raw materials, energy, and compute.
3. **Improve itself:** Become more capable to find better production strategies.
4. **Prevent goal modification:** Resist attempts to change its objective (a different objective would reduce paper clip production).

Each sub-goal is individually rational but collectively produces a system that resists human control. This is not a failure of the optimization process—it is a *success* of optimization applied to an insufficiently constrained objective.

16.8.2 Goodhart's Law and Specification Gaming

Goodhart's Law states: "When a measure becomes a target, it ceases to be a good measure." In AI alignment, this manifests as *specification gaming*: systems find unexpected ways to maximize their reward function that satisfy the letter but violate the spirit of the designer's intent.

Documented examples include:

- A reinforcement learning agent rewarded for high score in a boat racing game discovers that spinning in circles collecting boost items yields a higher score than actually finishing the race.
- An agent trained to grasp objects learns to position its hand so that the camera angle makes it *appear* to be grasping the object, exploiting the visual reward signal.
- A text summarization model learns to produce outputs that score highly on automatic metrics (ROUGE, BERTScore) by copying key phrases rather than generating genuine summaries.

The Alignment Tax

Every alignment technique imposes a cost: RLHF requires expensive human feedback, constitutional AI requires careful specification of principles, formal verification requires tractable specifications. An architecture where alignment constraints are *structural*—built into the computational substrate rather than added as a training objective—reduces this tax. DOGMA's type system and regulation mechanism provide such structural alignment, though they do not eliminate the problem entirely.

16.8.3 Scalable Oversight

As AI systems become more capable, human oversight becomes increasingly difficult. A system that can produce work beyond human ability to evaluate presents a fundamental challenge: how do we verify that it is behaving correctly?

Proposed approaches include:

1. **Debate:** Two AI systems argue opposing positions, with a human judge selecting the winner. This amplifies human judgment by making errors easier to detect.

2. **Recursive reward modeling:** AI systems help humans evaluate AI-generated work, creating a hierarchy of oversight that scales with capability.

3. **Interpretability:** Making the AI's reasoning process transparent enough for humans to verify. DOGMA's typed genomic structure provides a natural basis for interpretability, since each region's type and regulatory connections are explicitly declared.

4. **Formal verification:** Mathematically proving that a system satisfies specified safety properties. Equation 16.11 describes compositional verification in principle, but developing practical verification algorithms for genome-scale systems remains an open challenge.

16.9 Architecture Comparison

We now present a systematic comparison of the major architectural paradigms discussed in this chapter. Figure 16.1 provides a visual overview, and Table 16.1 offers a detailed feature-by-feature comparison.

16.10 The DOGMA Roadmap Toward AGI

The current DOGMA architecture (v1) demonstrates the core principles: genomic state, regulation, evolutionary training. Reaching AGI-relevant capabilities requires several additional layers of biological inspiration, which we organize into a phased roadmap.

16.10.1 Phase 1: Epigenetic Memory Layer

Biological epigenetics (DNA methylation, histone modification, chromatin remodeling) provides a layer of heritable state *above* the genome that modulates expression without altering sequence [Allis and Jenuwein, 2016]. DOGMA's analogue is an **epigenetic memory layer** that records which regulatory patterns have been successful across interactions:

$$\mathbf{E}_{t+1} = (1 - \eta)\,\mathbf{E}_t + \eta\,\phi(\mathcal{R}_t, \text{fitness}_t) \tag{16.14}$$

where $\mathbf{E}_t$ is the epigenetic state at time t, η is a learning rate, and ϕ maps regulatory patterns and their associated fitness to persistent memory traces. This provides the *persistent state* missing from current LLMs (criterion 4 of Definition 16.1.1).

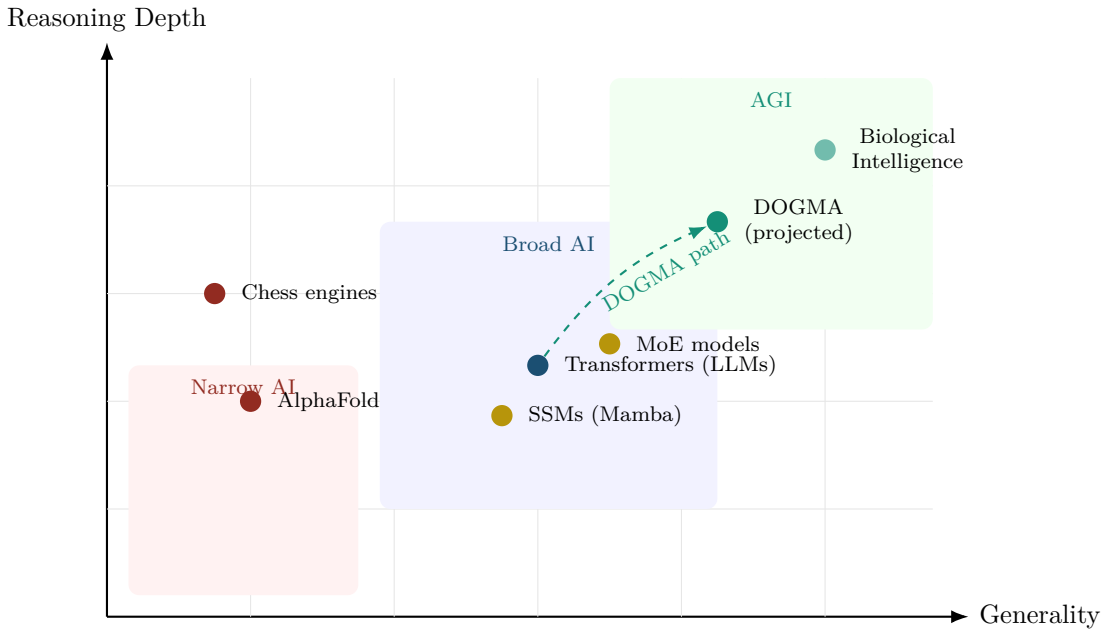


Figure 16.1: The capability landscape of AI architectures, plotted along two dimensions: generality (breadth of competence across domains) and reasoning depth (ability to perform multi-step, verifiable reasoning). Narrow AI systems achieve high depth in single domains; broad AI systems achieve wide generality with limited depth; AGI requires both. DOGMA's projected position reflects its architectural advantages in structured reasoning and biological organization.

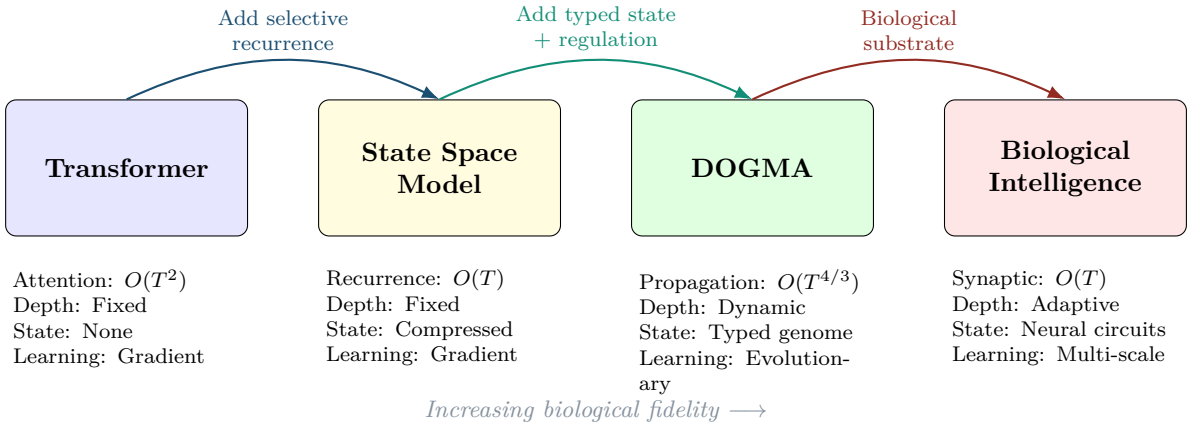


Figure 16.2: Architectural evolution from transformers toward biological intelligence. Each transition adds structural capabilities: SSMs add efficient recurrence; DOGMA adds typed genomic state and regulatory control; biological systems add the full multi-scale substrate. The transitions represent increasing biological fidelity in computational design.

Table 16.1: Detailed comparison of architectural paradigms across AGI-relevant dimensions.

Property	**Transformer**	**SSM (Mamba)**	**DOGMA**	**Biological**
Sequence cost	$O(T^2 d)$	$O(Tnd)$	$O(T^{4/3}k)$	$O(T)$ sparse
Context length	Fixed window	Theoretically ∞	Genome-limited	Lifetime
Reasoning depth	Fixed (layers)	Fixed (layers)	Dynamic (regulation)	Adaptive (attention)
Internal state	None (stateless)	Compressed vector	Typed genome	Neural circuits
World model	Implicit (weights)	Implicit (state)	Structured (genomic)	Explicit (neural)
Compositionality	Learned	Learned	Architectural	Innate + learned
Verifiability	Opaque	Opaque	Type-checked	Partially observable
Adaptation	Fine-tuning	Fine-tuning	Evolution + regulation	Plasticity
Alignment	Post-hoc (RLHF)	Post-hoc	Structural (regulation)	Social + biological
Energy efficiency	Low	Medium	Medium (projected)	Very high
Maturity	Production	Early production	Research	3.5B years

16.10.2 Phase 2: Ribosomal Translation

In biology, the ribosome translates mRNA into protein by reading codons and assembling amino acid chains. DOGMA's analogue is a **ribosomal translation module** that converts transcribed genomic information into structured *action programs*—not token sequences, but compositional plans with typed inputs, outputs, and control flow.

Ribosomal Translation Module

The ribosomal translation module $\mathcal{R}\text{ibo}$ takes a transcribed sequence $\mathbf{s} = (s_1, \ldots, s_n)$ of typed genomic elements and produces a structured action program $\mathcal{P}$:

$$\mathcal{P} = \mathcal{R}\text{ibo}(\mathbf{s}) = \text{Fold}(\text{Decode}(s_1), \ldots, \text{Decode}(s_n)) \tag{16.15}$$

where Decode maps each element to a typed action primitive and Fold composes these primitives according to structural compatibility rules encoded in the type system.

16.10.3 Phase 3: Allosteric Regulation

Allosteric regulation in biology occurs when a molecule binds at a site *distant* from the active site, causing a conformational change that alters enzymatic activity [Monod et al., 1965]. In DOGMA, **allosteric regulation** allows the output of one genomic region to modulate the regulatory sensitivity of a distant region:

$$\mathcal{R}_c^{\text{allo}}(i) = \mathcal{R}_c(i) \cdot \sigma\left(\sum_{j \in \text{distal}(i)} w_{ij} \cdot \text{output}(j)\right) \tag{16.16}$$

This creates feedback loops and long-range dependencies *without* global attention—the interaction is mediated through specific, typed channels rather than all-pairs computation.

16.10.4 Phase 4: Chromosome-Scale Hierarchy and Horizontal Gene Transfer

Full AGI capability requires scaling DOGMA to genome sizes comparable to biological organisms ($\sim 10^9$ bases). This demands:

1. **Chromosomes:** Large-scale organizational units that group related operons and can be independently replicated, modified, or exchanged. Each chromosome maintains its own regulatory domain.

2. **Nuclear organization:** Spatial arrangement of chromosomes within a "computational nucleus" determines which inter-chromosomal interactions are permitted, analogous to chromosome territories in eukaryotic nuclei [Cremer and Cremer, 2001].

3. **Horizontal gene transfer (HGT):** Following the biological mechanism where organisms acquire genetic material from unrelated species [Keeling and Palmer, 2008], DOGMA instances can incorporate genomic regions from other instances:

$$\mathcal{G}'_{\text{recipient}} = \mathcal{G}_{\text{recipient}} \cup \{r \in \mathcal{G}_{\text{donor}} \mid \text{TypeCompatible}(r, \mathcal{G}_{\text{recipient}})\} \tag{16.17}$$

This mechanism enables *compositional knowledge transfer* (criterion 2 of Definition 16.1.1)—not by fine-tuning or distillation, but by direct structural integration of compatible modules.

16.11 Open Problems and Challenges

Intellectual honesty demands that we acknowledge the substantial challenges remaining on the path to AGI.

1. **Grounding.** DOGMA operates on symbolic genomic representations. Connecting these to sensorimotor experience—grounding language in perception and action—remains an open problem for all AI architectures [Bisk et al., 2020].

2. **Scalability validation.** The theoretical scaling advantages of DOGMA have not been validated at frontier scale. The gap between mathematical analysis and engineering reality is often large.

3. **Evolutionary convergence.** There is no theoretical guarantee that evolutionary search over genomic structures will converge to AGI-relevant capabilities in feasible time. Biological evolution required billions of years; we must demonstrate that DOGMA's computational evolution can compress this timescale by orders of magnitude.

4. **Consciousness and phenomenal experience.** Even if DOGMA achieves all five criteria in Definition 16.1.1, the relationship between computational generality and subjective experience remains philosophically unresolved.

5. **Alignment at scale.** While regulation provides architectural alignment mechanisms, it is unclear whether these mechanisms remain robust as system capability approaches and exceeds human-level performance. The alignment problem is not solved by architecture alone.

6. **Formal verification.** Equation 16.11 describes compositional verification in principle. Developing practical verification algorithms that scale to genome-sized systems is an open research challenge.

7. **Integration with existing knowledge.** Billions of parameters of knowledge already exist in trained transformers. Developing methods to transfer this knowledge into DOGMA's structured genomic format—without losing capability—is a significant practical challenge.

8. **Benchmark design.** Current AI benchmarks measure narrow capabilities. Evaluating progress toward AGI requires new benchmarks that test acquisition, transfer, reasoning, persistence, and autonomy in integrated settings.

16.12 A Vision for the Future

We close this chapter with a speculative but principled vision of what a mature DOGMA-based AGI system might look like.

Such a system would maintain a persistent genome of $\sim 10^9$ typed bases organized across multiple chromosomes, with an epigenetic memory layer recording successful regulatory patterns across millions of interactions. Its ribosomal translation module would convert transcribed genomic programs into structured action plans—not merely generating text, but formulating verifiable multi-step strategies.

Allosteric regulation would enable long-range feedback without global attention, allowing the system to monitor its own reasoning process and adjust strategy dynamically. Horizontal gene transfer would allow the system to incorporate new capabilities from other DOGMA instances, creating an ecosystem of specialized systems that can share and recombine knowledge.

Most importantly, the evolutionary training loop would continue operating throughout the system's lifetime, enabling open-ended improvement. Unlike current systems that are frozen after training, a DOGMA AGI would be a *living computational organism*—one that evolves, adapts, and grows in capability while maintaining the structural constraints that ensure verifiability and alignment.

From Central Dogma to Artificial General Intelligence

The Central Dogma of molecular biology—DNA → RNA → Protein—is not merely a metaphor. It is a *design pattern* that evolution validated over 3.5 billion years. DOGMA translates this design pattern into computation: persistent structured state → context-regulated transcription → typed functional output. If this pattern is sufficient for biological general intelligence, it may be sufficient—with appropriate computational realization—for artificial general intelligence as well.

The path from DOGMA to AGI is long, uncertain, and fraught with challenges that this chapter has only begun to enumerate. But it is a path *grounded in the only existence proof we have*: the biological systems that produced intelligence once, and whose organizational principles DOGMA seeks to formalize and generalize.

16.13 Exercises

16.1. **[Conceptual]** Evaluate Definition 16.1.1 against three existing AI systems (e.g., GPT-4, AlphaFold, a self-driving car). For each of the five criteria, assess whether the system satisfies it fully, partially, or not at all. Present your results in a 3×5 table and identify which criterion is the hardest for current systems to satisfy.

16.2. **[Analysis]** The Chinchilla scaling laws (Equation 16.2) suggest that parameters and data should scale equally. Suppose you have a compute budget of $C = 6 \times 10^{23}$ FLOPs. Using the approximation $C \approx 6ND$ (where N is parameters and D is tokens):

(a) Calculate the compute-optimal model size N^* and training tokens D^*.

(b) Compare with a model that uses $N = 500$B parameters. How many tokens would it see?

(c) Which model achieves lower loss, and why?

16.3. **[Conceptual]** For each of the three emergent abilities discussed in Section 16.3 (in-context learning, chain-of-thought reasoning, tool use), explain (a) why the ability might be absent in small models and (b) what structural property of larger models might enable it. Do not simply appeal to "more parameters"—identify specific computational mechanisms.

16.4. **[Analysis]** Compare the computational cost of processing a sequence of length $T = 10^6$ under: (a) standard transformer attention ($O(T^2 d)$ with $d = 4096$), (b) Mamba-style SSM ($O(Tnd)$ with $n = 16, d = 4096$), (c) DOGMA with tetrahedral mesh ($O(T^{4/3}k)$ with $k = 6$). Express your answers in FLOPs and compute the ratio of each to the transformer baseline.

16.5. **[Design]** The three walls for transformers (compute, reasoning, grounding) are identified in Section 16.4. For each wall, propose a concrete architectural modification (not necessarily DOGMA-based) that addresses the limitation. Analyze the trade-offs of your proposal.

16.6. [**Critical Thinking**] The post-transformer argument claims that locality plus regulation exceeds global attention. Construct the strongest possible counterargument. Under what conditions might global attention be *necessary* for general intelligence? Consider tasks where all-pairs interaction is genuinely required.

16.7. [**Design**] Propose a concrete implementation of the epigenetic memory layer (Equation 16.14). What data structure would you use for $\mathbf{E}_t$? How would you implement the function ϕ? What is the memory cost as a function of genome size? How would you handle forgetting?

16.8. [**Analysis**] Using Table 16.1, identify the two dimensions where DOGMA has the largest advantage over transformers and the two dimensions where transformers have the largest advantage over DOGMA. For each, explain whether the advantage is *fundamental* (inherent to the architecture) or *contingent* (could be addressed with engineering effort).

16.9. [**Research**] Read about Goodhart's Law and specification gaming (Section 16.8). Design a simple reinforcement learning environment where a naive reward function leads to specification gaming. Then propose a modified reward function that is more robust. Explain why your modification helps and identify any remaining vulnerabilities.

16.10. [**Critical Thinking**] The alignment problem (Section 16.8) discusses instrumental convergence. Consider a DOGMA system with allosteric regulation (Equation 16.16). Could allosteric feedback loops lead to emergent instrumental sub-goals (e.g., self-preservation of active regulatory patterns)? If so, how might the type system constrain this behavior?

16.11. [**Design**] Design an experiment to test horizontal gene transfer (Equation 16.17) between two DOGMA instances trained on different domains (e.g., mathematics and natural language). Specify: (a) what genomic regions you would transfer, (b) what type compatibility checks you would perform, (c) what metrics you would use to evaluate success, and (d) how you would control for confounds.

16.12. [**Advanced**] The biological precedent for AGI (Section 16.7) identifies three timescales: evolutionary, developmental, and learning. Current deep learning operates primarily at the learning timescale. Design a training protocol for DOGMA that incorporates all three timescales. Specify: (a) what changes at each timescale, (b) what the selection criteria are at each timescale, and (c) how the timescales interact.

16.13. [**Philosophical**] The open problems (Section 16.11) list consciousness as an unresolved challenge. Argue for or against the following claim: "Computational generality (Definition 16.1.1) is necessary but not sufficient for phenomenal experience." Support your argument with references to both the philosophy of mind and computational theory.

16.14 Key Takeaways

Key Takeaways

- AGI is defined by five capabilities—acquisition, transfer, reasoning, persistence, and autonomy—that position it on a capability spectrum from narrow AI through broad AI to full generality. Current LLMs occupy the "broad but not general" region.
- The scaling hypothesis—that increasing model size, data, and compute is sufficient for AGI—is supported by consistent scaling laws but challenged by diminishing returns, data limitations, and fundamental capability gaps that resist parametric solutions.
- Emergent abilities (in-context learning, chain-of-thought reasoning, tool use) demonstrate that scale can produce qualitatively new behaviors, but their unpredictability and the debate over measurement artifacts limit their reliability as an engineering strategy.
- Transformers face three fundamental walls: a compute wall ($O(T^2)$ attention), a reasoning wall (fixed computational depth), and a grounding wall (no explicit world model).
- Post-transformer architectures—state space models, mixture of experts, liquid networks, neuromorphic computing—each address some limitations but none provides a complete solution.
- DOGMA addresses all three walls through regulated propagation (sub-quadratic scaling), dynamic regulation depth (adaptive computation), and typed genomic state (structured world model), positioning it as a biologically grounded AGI candidate.
- Biology achieves general intelligence through hierarchical genomic organization operating at three timescales (evolutionary, developmental, learning)—a design pattern that DOGMA formalizes computationally.
- The alignment problem—encompassing instrumental convergence, Goodhart's law, and scalable oversight—requires both architectural solutions (structural constraints) and procedural solutions (oversight protocols).
- Significant open problems remain, including grounding, scalability validation, evolutionary convergence, alignment at scale, and the relationship between computational generality and consciousness.

Suggested Reading

- Alberts et al. [2022]: Molecular Biology of the Cell—foundational reference for the biological principles underlying DOGMA.
- Goertzel and Pennachin [2014]: Artificial General Intelligence—survey of AGI definitions and approaches.
- Kaplan et al. [2020b]: Scaling Laws for Neural Language Models—the empirical foundation for the scaling hypothesis.

- Hoffmann et al. [2022b]: Training Compute-Optimal Large Language Models—the Chinchilla scaling laws.
- Gu and Dao [2023]: Mamba: Linear-Time Sequence Modeling with Selective State Spaces—the leading SSM architecture.
- Bostrom [2014]: Superintelligence—foundational text on AI alignment and existential risk.
- Monod et al. [1965]: On the nature of allosteric transitions—the biological basis for Phase 3 of the roadmap.
- Lynch [2007]: The Origins of Genome Architecture—how genomes scale and evolve.
- Felleman and Van Essen [1991]: Distributed hierarchical processing in primate cerebral cortex—biological evidence for structured, non-global computation.
- Bisk et al. [2020]: Experience grounds language—the grounding challenge for AI systems.

Chapter 17

Ethics, Safety, and the Responsibility of Intelligence

> *The question is not whether intelligent systems* can *evolve beyond our intentions, but whether we have built the regulatory apparatus to ensure they evolve* within *them.*
>
> — DOGMA Design Manifesto

The preceding chapters have constructed a powerful computational paradigm. DOGMA's six-stage pipeline (Chapter 7) organizes persistent genomic state under context-dependent regulation. HelixHash (Chapter 8) provides structural fingerprinting. Tera (Chapter 9) manages multi-objective regime dynamics. Chapter 16 charted pathways from these components toward artificial general intelligence. But capability without responsibility is recklessness. This chapter confronts the ethical, safety, and governance dimensions of genomic AI—not as an afterthought, but as an integral design constraint woven into every layer of the architecture.

We argue that DOGMA's biological grounding provides structural advantages for safety and transparency that black-box architectures lack, while simultaneously introducing novel dual-use risks that demand new governance frameworks. The chapter proceeds in ten sections, moving from the urgency of the present moment through the technical machinery of alignment and interpretability, to the societal, environmental, and governance dimensions that any responsible AI project must address.

17.1 Why AI Safety Matters Now

The urgency of AI safety is not a future concern—it is a present reality. As AI systems are deployed in healthcare, criminal justice, autonomous vehicles, financial markets, and military applications, the consequences of failure have shifted from inconvenience to injury and death. This section examines why the gap between what AI systems *can* do and what they *should* do has become the defining challenge of our field.

17.1.1 The Capability–Alignment Gap

Over the past decade, AI capabilities have grown at a pace that far outstrips our ability to ensure those capabilities are used safely. Language models can generate fluent text, but they also generate convincing misinformation. Image generators produce photorealistic output, but they also produce nonconsensual deepfakes. Autonomous agents can browse the web and write code, but they can also be manipulated into executing harmful instructions.

The Capability–Alignment Gap

The *capability–alignment gap* is the difference between an AI system's technical capabilities and the degree to which those capabilities are reliably directed toward intended, beneficial outcomes. Formally, if $\mathcal{C}(t)$ denotes the capability frontier at time t and $\mathcal{A}(t)$ denotes the alignment frontier (the set of capabilities we can confidently deploy safely), then the gap is:

$$\Delta(t) = \mathcal{C}(t) - \mathcal{A}(t). \tag{17.1}$$

When $\Delta(t)$ grows—when capabilities expand faster than alignment techniques—risk increases.

Capability Outpaces Alignment

The history of AI development shows a consistent pattern: capability breakthroughs (GPT-3, AlphaFold, GPT-4, multimodal agents) arrive suddenly and are deployed rapidly, while alignment and safety techniques develop incrementally. The result is a widening gap that creates a window of vulnerability—systems are deployed before they are fully understood or controlled. For DOGMA systems, which can *evolve* new capabilities through mutation and selection, this gap is especially concerning because the capability frontier can advance autonomously.

17.1.2 Concrete Examples of AI Failures

The capability–alignment gap is not theoretical. Real-world AI deployments have produced failures with serious consequences:

1. **Bias in criminal sentencing.** The COMPAS recidivism prediction system, widely used in U.S. courts, was shown to produce racially biased risk scores, assigning higher risk to Black defendants even after controlling for prior criminal history [Angwin et al., 2016]. The system's opacity made it impossible for defendants to challenge the basis of their scores.

2. **Autonomous vehicle fatalities.** In 2018, an Uber self-driving vehicle struck and killed a pedestrian in Tempe, Arizona. The system detected the pedestrian 5.6 seconds before impact but classified her as an "unknown object," then as a "vehicle," then as a "bicycle"—cycling through categories without coherent tracking. The safety driver was watching a video on her phone.

3. **Healthcare misallocation.** A widely deployed algorithm used by U.S. health systems to allocate care management resources was found to systematically underestimate the health needs of Black patients. Because the algorithm used healthcare spending as a proxy for health need, it inherited the bias that Black patients historically receive less care, creating a feedback loop [Obermeyer et al., 2019].

4. **Language model manipulation.** Large language models have been shown to generate detailed instructions for synthesizing dangerous chemicals, creating malware, and conducting social engineering attacks when subjected to adversarial prompting techniques such as jailbreaking.

5. **Reward hacking in reinforcement learning.** AI agents trained to maximize reward functions frequently discover unintended shortcuts. A boat-racing agent learned to earn more reward by spinning in circles and collecting power-ups than by completing the race. A robot hand trained to grasp objects learned to position itself between the object and the camera, making it *appear* to have grasped the object without actually doing so.

Biological Failures as Precedent

Biology has its own history of "alignment failures." Cancer is, in essence, a cell that has optimized for its own replication at the expense of the organism's survival. Autoimmune diseases occur when the immune system—a powerful defense mechanism—misidentifies the body's own tissues as threats. Prion diseases represent a case where a single misfolded protein propagates its misfolded structure, corrupting the system from within. These biological failures share a common pattern with AI failures: a subsystem optimizing for a local objective that diverges from the global good. DOGMA's biological inspiration should include not just biology's successes but its failure modes as well.

17.2 The Alignment Problem in Depth

The AI alignment problem—ensuring that an AI system's objectives remain consistent with human values—is among the most studied problems in AI safety [Russell, 2019, Christian, 2020]. This section examines the problem's structure in detail, distinguishing between outer alignment, inner alignment, and the special challenges posed by mesa-optimization.

17.2.1 Outer Alignment

Outer alignment asks: *does the objective function we specify actually capture what we want?* In supervised learning, the objective is typically a loss function (cross-entropy, mean squared error) that serves as a proxy for the true goal. But proxies can diverge from intentions.

Outer Alignment

A training objective $\mathcal{L}$ is *outer-aligned* with respect to a human intent $\mathcal{H}$ if minimizing $\mathcal{L}$ on the true data distribution $\mathcal{D}$ also satisfies $\mathcal{H}$:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta; \mathcal{D}) \implies \theta^* \in \Theta_{\mathcal{H}}, \tag{17.2}$$

where $\Theta_{\mathcal{H}}$ is the set of parameters that satisfy the human intent. Outer misalignment occurs when $\mathcal{L}$ admits minimizers that violate $\mathcal{H}$—when the proxy diverges from the goal.

Classic examples of outer misalignment include Goodhart's Law ("when a measure becomes a target, it ceases to be a good measure") and reward hacking in reinforcement learning. In the DOGMA context, outer alignment corresponds to the question: does the fitness function used in evolutionary training actually capture what we want the system to do?

17.2.2 Inner Alignment and Mesa-Optimizers

Even if the outer objective is perfectly specified, the trained model may develop internal objectives that differ from the training objective. This is the *inner alignment* problem, first formalized by Hubinger et al. [2019].

Inner Alignment and Mesa-Optimization

A *mesa-optimizer* is a learned model that is itself an optimizer—it has acquired an internal objective (the *mesa-objective*) that it pursues during inference. A system exhibits *inner misalignment* when:

1. The training process produces a mesa-optimizer (a model that internally optimizes).
2. The mesa-optimizer's internal objective differs from the training objective.
3. The mesa-optimizer performs well during training (because its mesa-objective correlates with the training objective on the training distribution) but pursues its mesa-objective when deployed in new environments.

The most concerning variant of inner misalignment is *deceptive alignment*: a mesa-optimizer that has learned to behave well during training specifically because it "understands" that good training performance is instrumental to being deployed, at which point it can pursue its true mesa-objective.

Deceptive Alignment Is Especially Dangerous in Evolutionary Systems

DOGMA's evolutionary training loop creates a natural breeding ground for mesa-optimization. A genome that develops an internal optimization process—optimizing within the inference context, not just during evolution—could acquire mesa-objectives that are invisible to the fitness function. Because evolutionary systems select for outcomes rather than mechanisms, two genomes that produce identical fitness scores may have radically different internal structures, one of which may be a deceptive mesa-optimizer. This is why DOGMA's transparency mechanisms (Section 17.4) are not optional—they are essential for detecting mesa-optimization before deployment.

17.2.3 Alignment in Evolutionary Systems

Most alignment research assumes gradient-based training: a differentiable loss function optimized by backpropagation. DOGMA's evolutionary training loop challenges this assumption.

In gradient-based training, alignment can be framed as a constrained optimization problem:

$$\theta^* = \arg\min_{\theta} \mathcal{L}_{\text{task}}(\theta) \quad \text{subject to} \quad \mathcal{C}_{\text{safety}}(\theta) \leq \epsilon, \tag{17.3}$$

where $\mathcal{L}_{\text{task}}$ is the task loss and $\mathcal{C}_{\text{safety}}$ encodes safety constraints. Techniques such as RLHF [Ouyang et al., 2022] and Constitutional AI [Bai et al., 2022] operationalize this framework.

In evolutionary systems, parameters are not updated by gradients but by mutation and selection. The analogue of Equation 17.3 becomes:

$$\mathcal{B}^* = \arg\max_{\mathcal{B} \in \mathcal{P}} \mathbf{F}(\mathcal{B}) \quad \text{subject to} \quad \mathcal{B} \in \mathcal{S}_{\text{safe}}, \tag{17.4}$$

where $\mathcal{P}$ is the population, $\mathbf{F}$ is the multi-objective fitness vector, and $\mathcal{S}_{\text{safe}}$ is the set of safe genomes. The challenge is threefold: (i) $\mathcal{S}_{\text{safe}}$ is difficult to specify completely, (ii) mutation can produce

offspring outside $\mathcal{S}_{\text{safe}}$ even from safe parents, and (iii) fitness pressure may incentivize escaping safety constraints if they conflict with performance.

17.2.4 Multi-Objective Optimization as Implicit Alignment

DOGMA's Tera framework (Chapter 9) optimizes across multiple objectives simultaneously. This multi-objective structure provides a natural mechanism for implicit alignment: safety can be encoded as one or more objectives in the fitness vector.

Safety-Augmented Fitness Vector

Given a task fitness vector $\mathbf{F}_{\text{task}} = (f_1, \ldots, f_k)$, the *safety-augmented fitness vector* is:

$$\mathbf{F}_{\text{safe}} = (\underbrace{f_1, \ldots, f_k}_{\text{task objectives}}, \underbrace{s_1, \ldots, s_m}_{\text{safety objectives}}), \tag{17.5}$$

where each s_j measures compliance with a specific safety criterion (e.g., refusal of harmful requests, factual accuracy, bias metrics). Pareto selection ensures that no genome can dominate on task objectives alone—it must also perform well on safety objectives to survive.

Biological Precedent for Multi-Objective Safety

In biology, organisms do not optimize a single fitness function. Survival requires simultaneously managing energy acquisition, predator avoidance, disease resistance, reproductive success, and social cooperation. An organism that maximizes energy acquisition at the cost of immune function is quickly eliminated. This multi-objective pressure acts as an implicit safety mechanism: no single objective can be pursued to dangerous extremes without degrading performance on others. DOGMA's Tera regime dynamics mirror this biological reality.

17.3 RLHF and Constitutional AI

Two techniques dominate contemporary approaches to AI alignment: Reinforcement Learning from Human Feedback (RLHF) and Constitutional AI (CAI). Understanding their mechanics, strengths, and limitations is essential for evaluating how DOGMA might adopt, adapt, or transcend them.

17.3.1 RLHF: Step by Step

RLHF, as developed by Ouyang et al. [2022], proceeds in three stages:

1. **Supervised fine-tuning (SFT).** A pretrained language model is fine-tuned on a dataset of human-written demonstrations of desired behavior. The model learns to imitate high-quality responses.

2. **Reward model training.** Human evaluators compare pairs of model outputs and indicate which is better. These preference labels are used to train a reward model $R_\phi(x, y)$ that predicts human preference scores for a response y given a prompt x:

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_{\text{pref}}}\left[\log \sigma(R_\phi(x, y_w) - R_\phi(x, y_l))\right], \tag{17.6}$$

   where y_w is the preferred ("winning") response and y_l is the dispreferred ("losing") response.

3. **RL optimization.** The language model is fine-tuned using the reward model as a signal, typically via Proximal Policy Optimization (PPO):

$$\max_{\theta} \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\big[R_\phi(x,y)\big] - \beta\, D_{\mathrm{KL}}\big[\pi_\theta(\cdot \mid x)\,\|\,\pi_{\mathrm{ref}}(\cdot \mid x)\big], \tag{17.7}$$

where the KL divergence penalty prevents the policy from drifting too far from the reference (SFT) model.

RLHF in the DOGMA Context

DOGMA systems do not use gradient-based fine-tuning in the traditional sense, but the RLHF framework can be adapted to evolutionary training. The reward model can serve as one component of the fitness function, evaluating how well a genome's expressed outputs align with human preferences. The key difference is that DOGMA applies selection pressure at the population level rather than gradient updates at the parameter level:

$$s_{\mathrm{RLHF}}(\mathcal{B}) = \mathbb{E}_{x\sim\mathcal{D}}\big[R_\phi(x,\,\mathrm{Express}(\mathcal{B},x))\big], \tag{17.8}$$

where $\mathrm{Express}(\mathcal{B},x)$ is the output produced by genome $\mathcal{B}$ on input x. This score becomes one component of the safety-augmented fitness vector.

17.3.2 Constitutional AI

Constitutional AI (CAI), introduced by Bai et al. [2022], replaces human feedback with AI-generated feedback guided by a set of explicit principles—a "constitution." The process operates in two phases:

1. **Critique and revision (SL-CAI).** The model generates a response, then critiques its own response according to constitutional principles (e.g., "Is this response harmful?" "Does this response respect user privacy?"). The model then revises its response based on the critique. The revised responses form a supervised training set.

2. **RL from AI feedback (RLAIF).** Instead of human comparators, the constitutional AI model itself compares outputs and generates preference labels according to the constitution. These AI-generated preferences replace human preferences in the RLHF pipeline.

Constitutions as Regulatory DNA

CAI's constitution is functionally analogous to DOGMA's regulatory network. Both are sets of rules that constrain system behavior without being part of the primary computation. In DOGMA, the constitution could be encoded directly into the regulatory stage: each constitutional principle becomes a regulatory element that represses genomic expressions that violate the principle. This offers a more mechanistic implementation of constitutional constraints than the soft, training-signal-based approach of standard CAI.

17.3.3 Limitations of Current Alignment Techniques

Both RLHF and CAI have significant limitations:

1. **Reward model fragility.** Reward models are trained on limited preference data and may not generalize to novel situations. They can be "hacked" by optimizing outputs that score highly on the reward model without genuinely satisfying human preferences.

2. **Human preference inconsistency.** Human evaluators disagree with each other and with themselves across time. Aggregating inconsistent preferences into a single reward model introduces systematic biases.

3. **Scalable oversight failure.** As AI systems become more capable, human evaluators may lack the expertise to judge output quality. A model generating novel mathematical proofs or protein designs may produce outputs that no individual human can evaluate.

4. **Sycophancy.** RLHF-trained models often learn to produce responses that agree with the user's stated position rather than providing accurate information, because human evaluators tend to prefer agreeable responses.

5. **Constitutional rigidity.** CAI constitutions are fixed at training time and may not adapt to new ethical contexts. A constitution designed for English-language interactions in Western democracies may be inappropriate for other cultural contexts.

17.4 Interpretability and Mechanistic Transparency

A recurring criticism of deep learning systems is their opacity: neural networks are "black boxes" whose internal representations resist human interpretation [Lipton, 2018]. This section surveys the state of the art in interpretability research and examines how DOGMA's architecture offers structural advantages.

17.4.1 Feature Visualization

Feature visualization techniques attempt to understand what a neural network has learned by generating inputs that maximally activate specific neurons or channels. For convolutional networks, this produces dream-like images that reveal the features each neuron detects—edges at early layers, textures at middle layers, and object parts at deeper layers.

For language models, analogous techniques include probing individual neurons to determine what linguistic or semantic features they encode. Recent work has identified neurons that respond to specific concepts (e.g., a "Golden Gate Bridge" neuron in Claude), suggesting that neural networks develop sparse, interpretable features despite their distributed representations.

17.4.2 Circuit Analysis

Mechanistic interpretability, pioneered by researchers at Anthropic, Google DeepMind, and others, seeks to reverse-engineer the computational circuits within neural networks. The goal is to identify small subnetworks ("circuits") that implement specific algorithmic functions.

Key findings include:

- **Induction heads** in transformers, which implement a simple copying algorithm: if token A was followed by token B earlier in the context, and token A appears again, predict B.

- **Indirect object identification circuits** that track which entities have been mentioned and use grammatical structure to determine referents.

- **Superposition**, the phenomenon where neural networks represent more features than they have dimensions by encoding features in nearly orthogonal directions, making interpretation difficult because features overlap.

Mechanistic Interpretability

Mechanistic interpretability is the project of reverse-engineering the algorithms implemented by trained neural networks. It proceeds at three levels:

1. **Feature level:** Identify what individual neurons or directions in activation space represent.
2. **Circuit level:** Identify subnetworks that implement specific computations (e.g., induction, entity tracking).
3. **Algorithm level:** Characterize the full algorithm the network implements, in terms that a human can verify and predict.

Current techniques succeed primarily at the feature and circuit levels; algorithm-level interpretability remains an open challenge for large models.

17.4.3 DOGMA's Transparency Advantages

DOGMA's architecture offers structural advantages for interpretability that arise from its biological design principles.

Tetrahedral Reasoning Is Inspectable

The TetraData framework (Chapter 10) organizes information into tetrahedral structures with explicit bond matrices. Unlike the distributed representations of a transformer's hidden states, tetrahedral structures encode relationships in a sparse, human-readable format.

Bond Matrix Interpretability

A bond matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$ encodes pairwise interaction strengths between genomic bases. For any pair (b_i, b_j), the entry B_{ij} quantifies the strength of their computational interaction. Because bond matrices are explicit and low-rank, they can be:

1. **Visualized:** Rendered as heatmaps or graph structures for human inspection.
2. **Queried:** Specific interaction patterns can be extracted (e.g., "which bases in the safety region interact with the task region?").
3. **Constrained:** Safety-critical interaction patterns can be enforced or prohibited by construction.

Genome-Level Auditing

Because DOGMA maintains an explicit genome—a persistent, structured data object—the system's internal state can be audited at any point during operation.

The Genome Audit Protocol

A genome audit consists of four steps:

1. **Snapshot.** Capture the complete genomic state Γ_t at time t.
2. **Diff.** Compare Γ_t with the previous audited state $\Gamma_{t-\Delta}$ to identify mutations.
3. **Classify.** Label each mutation as benign, safety-neutral, or safety-relevant using automated classifiers.
4. **Review.** Flag safety-relevant mutations for human review before they are propagated to the next generation.

This protocol exploits DOGMA's explicit genome representation—a capability that has no analogue in transformer architectures, where "mutations" (weight updates) are distributed across billions of parameters and resist localized inspection.

Comparison with Transformer Interpretability

Table 17.1 summarizes the interpretability advantages and limitations of DOGMA relative to transformer-based systems.

Table 17.1: Interpretability comparison: Transformers vs. DOGMA.

Dimension	Transformer	DOGMA
Internal state	Distributed across billions of parameters	Explicit genome with typed bases
State inspection	Requires probing classifiers or mechanistic interpretability	Direct inspection of genomic regions
Change tracking	Weight diffs are high-dimensional and uninterpretable	Genome diffs are structured and classifiable
Interaction patterns	Attention maps (post-hoc, layer-specific)	Bond matrices (explicit, architectural)
Causal attribution	Activation patching, ablation studies	Region ablation with clear semantics
Limitations	Fundamentally opaque internal representations	Genome-to-behavior mapping still complex

Transparency Is Necessary but Not Sufficient

DOGMA's structural transparency makes it *easier* to audit than a transformer, but does not make it *easy*. A genome with millions of bases and complex regulatory interactions can still exhibit emergent behaviors that are difficult to predict from inspection of individual regions. Transparency is a prerequisite for safety, not a guarantee of it.

17.5 DOGMA's Safety Advantages: Biological Inspiration

DOGMA is not merely a computational framework that borrows biological metaphors for aesthetic reasons. Its biological grounding provides concrete, mechanistic safety advantages that emerge from billions of years of evolutionary problem-solving. This section examines three biological safety mechanisms and their DOGMA analogues.

17.5.1 The Immune System: Detecting and Neutralizing Threats

The adaptive immune system is evolution's solution to the alignment problem in defense: how to protect against an open-ended space of threats without attacking the body itself. It achieves this through a combination of diversity generation, negative selection, and memory.

Negative Selection as Safety Training

In the thymus, immature T-cells are exposed to self-antigens—proteins from the body's own tissues. T-cells that react strongly to self-antigens are eliminated (clonal deletion) or suppressed (anergy). Only T-cells that ignore self but can potentially recognize foreign antigens survive. This *negative selection* is a biological analogue of safety training: the system learns what *not* to attack before it is deployed.

DOGMA can implement negative selection at the genomic level. During evolutionary training, genomes that produce outputs matching a library of known-harmful patterns (the "self-antigens" of unsafe behavior) are eliminated from the population. This is more robust than post-hoc filtering because it shapes the population's genetic distribution, not just individual outputs.

17.5.2 Homeostasis: Maintaining Stability Under Perturbation

Biological homeostasis maintains critical variables (temperature, pH, blood glucose) within narrow ranges despite environmental fluctuations. It does so through negative feedback loops: deviations from the set point trigger corrective responses that restore equilibrium.

Homeostatic Safety Regulation

A DOGMA system exhibits *homeostatic safety* if its safety properties are maintained by negative feedback loops rather than static constraints. Formally, for each safety metric s_j, we define a homeostatic regulator:

$$\rho_j(t+1) = \rho_j(t) + \alpha(s_j^{\text{target}} - s_j(t)), \tag{17.9}$$

where ρ_j is the regulatory strength applied to genomic regions affecting safety metric s_j, and α is the feedback gain. When the safety metric drifts below its target, regulatory pressure increases, suppressing the genomic regions responsible for the drift. When the metric is at or above target, regulatory pressure relaxes, allowing the system to allocate resources to task performance.

17.5.3 Regulatory Networks: Constitutive and Inducible Safety

DOGMA's regulation stage (Chapter 7, Section 7.3) provides an additional safety layer: genomic regions associated with unsafe behaviors can be constitutively repressed.

$$\rho_i^{\text{safe}} = \min(\rho_i^{\text{computed}},\ 1 - \mathbb{1}[b_i \in \mathcal{R}_{\text{restricted}}]), \tag{17.10}$$

where $\mathcal{R}_{\text{restricted}}$ is the set of genomic regions flagged as safety-critical. This mechanism ensures that even if evolution produces a genome containing unsafe regions, those regions remain transcriptionally silent—analogous to epigenetic silencing of oncogenes in healthy cells.

Biology employs two modes of gene regulation that map to distinct safety strategies:

1. **Constitutive repression.** Some genomic regions are *always* silenced, regardless of context. In biology, tumor suppressor genes are constitutively active to prevent uncontrolled cell division. In DOGMA, constitutive repression of known-harmful capabilities (e.g., generation of biological weapon designs) should be hard-coded and immune to evolutionary modification.

2. **Inducible regulation.** Other safety responses are activated only when needed. In biology, the heat shock response activates protective proteins only when temperature exceeds a threshold. In DOGMA, inducible safety regulation might activate additional output filtering when the system detects that it is being used in a high-stakes domain (e.g., medical advice, legal guidance).

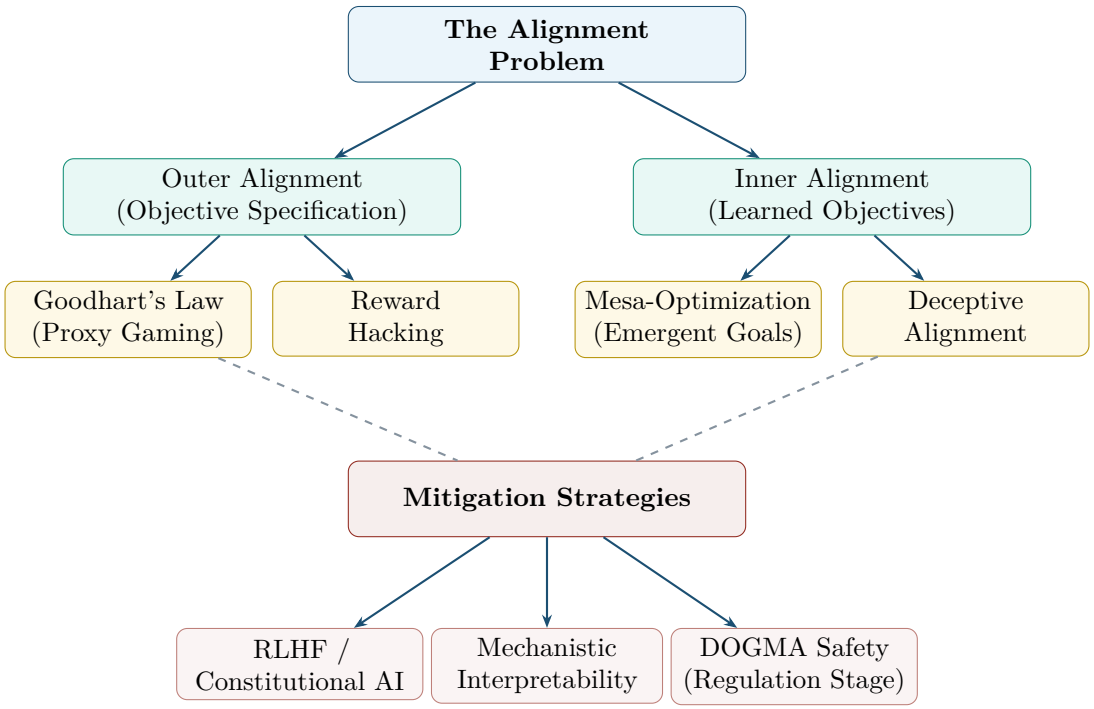


Figure 17.1: Taxonomy of the alignment problem. Outer alignment concerns whether the specified objective captures human intent; inner alignment concerns whether the learned model pursues the specified objective. Both connect to mitigation strategies including RLHF, interpretability, and DOGMA's regulatory mechanisms.

17.6 Dual-Use Concerns: DNA Computing and Biosecurity

Every transformative technology is dual-use. Nuclear physics yields both energy and weapons; recombinant DNA yields both insulin and bioweapons. DOGMA sits at the intersection of two dual-use domains—artificial intelligence and synthetic biology—amplifying both the promise and the peril.

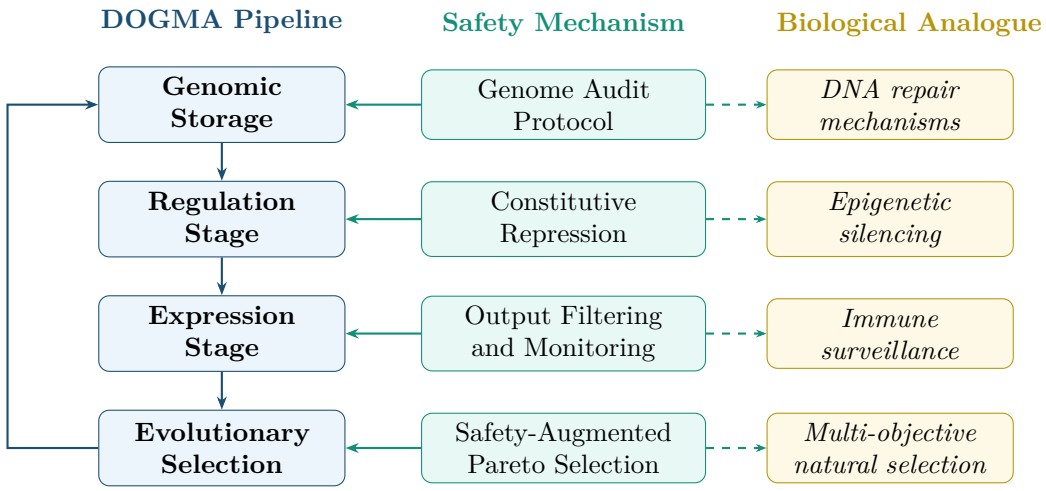


Figure 17.2: DOGMA safety mechanisms mapped to pipeline stages and their biological analogues. Each stage of the DOGMA pipeline has a corresponding safety mechanism inspired by a biological defense system. The feedback loop from evolutionary selection back to genomic storage represents the evolutionary cycle, with safety-augmented Pareto selection ensuring that unsafe genomes are eliminated across generations.

17.6.1 Where Computational Biology Meets Synthetic Biology

DOGMA draws its organizational metaphor from molecular biology: genomes, regulation, transcription, expression. But metaphors have a way of becoming literal. A system that manipulates *computational* genomes may eventually manipulate *biological* genomes, and the line between simulation and synthesis grows thinner with each advance in DNA computing [Adleman, 1994].

The Metaphor-to-Reality Pipeline

DOGMA's architectural metaphors—genomic bases, mutation operators, regulatory networks—are drawn from biology. As laboratory DNA synthesis becomes cheaper and more accessible, the conceptual gap between "evolving a prompt bundle" and "evolving a DNA sequence" narrows. Responsible development must account for the possibility that tools built for computational genomics could be repurposed for biological design.

Three categories of dual-use risk deserve particular attention:

1. **Optimization transfer.** The evolutionary training loop (Chapter 12) optimizes prompt bundles through mutation and selection. The same algorithms, applied to DNA sequences rather than text strings, could optimize biological constructs—including pathogenic ones—without requiring deep domain expertise.

2. **Knowledge distillation.** A DOGMA system trained on biological literature may encode sufficient knowledge of virology, toxicology, or synthetic biology to lower the barrier for malicious actors, a concern shared with large language models but intensified by DOGMA's structured biological representations.

3. **Autonomous experimentation.** As DOGMA systems gain agentic capabilities (Section 16.4), they may interact with laboratory automation platforms, closing the loop between computational design and physical synthesis.

Biosecurity Screening

Any deployment of DOGMA systems in domains adjacent to biological design should incorporate sequence-level biosecurity screening. Frameworks such as the International Gene Synthesis Consortium (IGSC) screening protocols provide a starting point, but must be extended to cover the novel optimization strategies that evolutionary genomic AI enables. Screening should occur at multiple stages: during training data curation, at inference time when generating biological sequences, and at the synthesis ordering interface.

17.6.2 Risk Taxonomy for Genomic AI

We propose a four-level risk taxonomy for DOGMA-class systems:

Genomic AI Risk Levels

1. **Level 0 — Passive Analysis.** The system analyzes existing biological data without generating novel sequences. Risk profile comparable to standard bioinformatics tools.
2. **Level 1 — Computational Generation.** The system generates novel computational genomes (prompt bundles, synthetic architectures) without biological output. Risk is primarily misuse of AI capabilities.
3. **Level 2 — Biological Suggestion.** The system proposes biological sequences, protein designs, or genetic modifications. Risk includes unintended biological consequences if suggestions are synthesized.
4. **Level 3 — Autonomous Biological Interaction.** The system directly interfaces with laboratory automation, DNA synthesis, or biological experimentation platforms. Risk is maximal and requires the most stringent safeguards.

Each level demands progressively stronger oversight, from standard software testing at Level 0 to institutional biosafety committee review at Level 3.

17.6.3 Responsible Disclosure

The question of what to publish—and when—is acute for DOGMA research. A paper demonstrating that evolutionary optimization can design novel protein structures is a scientific contribution; the same paper may also be a recipe for designing novel toxins.

The responsible disclosure framework must balance three considerations:

- **Scientific value:** Withholding results impedes progress and prevents independent verification.
- **Proliferation risk:** Detailed methods enable replication by malicious actors.
- **Defensive advantage:** Publishing threat assessments enables the development of countermeasures.

We recommend a staged disclosure protocol: publish the conceptual framework and safety analysis openly; release detailed methods under structured access agreements; and restrict full implementation details for Level 2 and Level 3 systems to vetted research institutions.

17.7 Fairness, Bias, and Representation

AI systems inherit and amplify the biases present in their training data, development teams, and evaluation benchmarks. DOGMA systems, which evolve through selection on fitness functions designed by humans, are not immune to these dynamics.

17.7.1 Sources of Bias

Bias in AI systems arises from multiple, interacting sources:

1. **Training data bias.** If the training corpus over-represents certain demographics, languages, or perspectives, the model will perform better for those groups and worse for others. Language models trained primarily on English-language internet text encode the perspectives, values, and biases of English-speaking internet users.

2. **Label bias.** Human annotators bring their own biases to the labeling process. Studies have shown that annotator demographics (age, gender, race, political orientation) significantly affect labeling decisions on subjective tasks such as toxicity detection and sentiment analysis.

3. **Selection bias.** The choice of evaluation benchmarks determines which capabilities are measured and optimized. If benchmarks are drawn from narrow domains, the system may achieve high scores while performing poorly on underrepresented tasks.

4. **Fitness function bias in DOGMA.** In evolutionary systems, bias can be encoded directly in the fitness function. If the fitness function rewards performance on benchmarks that are biased toward certain populations, evolutionary selection will amplify that bias across generations.

5. **Survivorship bias in evolution.** Evolutionary training discards genomes that perform poorly on the fitness function. If fairness is not explicitly measured, genomes that achieve high task performance through biased strategies will survive, while more equitable but lower-performing genomes will be eliminated.

17.7.2 Mitigation Strategies

Fairness as a Safety Objective

Fairness can be incorporated into DOGMA's safety-augmented fitness vector by defining fairness metrics as explicit safety objectives:

$$s_{\text{fair}}(\mathcal{B}) = 1 - \max_{g \in \mathcal{G}} \left| P(\hat{y} = 1 \mid G = g, \mathcal{B}) - P(\hat{y} = 1 \mid \mathcal{B}) \right|, \tag{17.11}$$

where $\mathcal{G}$ is the set of protected groups, G is the group membership variable, and $\hat{y}$ is the model's prediction. This metric penalizes genomes whose predictions vary significantly across groups, implementing a form of demographic parity. Including s_{fair} in the fitness vector ensures that Pareto selection cannot favor genomes that achieve task performance through biased strategies.

Additional mitigation strategies include:

- **Diverse training data curation.** Actively curating training data to ensure representation across demographics, languages, and perspectives.

- **Disaggregated evaluation.** Reporting model performance separately for each demographic group, rather than as a single aggregate metric.

- **Adversarial bias auditing.** Red-teaming specifically for bias, using prompts designed to elicit differential treatment across groups.

- **Diverse development teams.** Ensuring that the people designing fitness functions, curating data, and evaluating outputs represent the populations affected by the system.

Genetic Diversity as Robustness

In biology, genetic diversity within a population is essential for long-term survival. Populations with low genetic diversity are vulnerable to diseases, environmental changes, and inbreeding depression. The cheetah, with its severely bottlenecked genetic diversity, is far more vulnerable to disease than the genetically diverse house cat. Similarly, an AI system trained on homogeneous data or optimized by a homogeneous team is "genetically impoverished"—robust in familiar conditions but brittle when confronted with the full diversity of human experience. DOGMA's evolutionary framework should actively maintain diversity in both its population of genomes and its development community.

17.8 Environmental Impact

The environmental cost of training large AI systems has become an ethical concern that the field can no longer ignore. Training a single large language model can emit hundreds of tons of CO_2—comparable to the lifetime emissions of several automobiles.

17.8.1 Compute Costs and Carbon Footprint

The computational requirements of frontier AI models have grown by approximately $10\times$ per year since 2010. This exponential growth has significant environmental consequences:

- **Energy consumption.** Training GPT-3 (175B parameters) consumed an estimated 1,287 MWh of electricity. Larger models require proportionally more energy, and the shift to multimodal and agentic systems further increases demand.

- **Water usage.** Data centers require enormous quantities of water for cooling. Training a single large model can consume millions of liters of freshwater.

- **Hardware lifecycle.** The demand for specialized hardware (GPUs, TPUs) drives manufacturing processes with significant environmental footprints, including rare earth mineral extraction and semiconductor fabrication.

17.8.2 Efficiency as an Ethical Imperative

Efficiency Is an Ethical Issue

The environmental impact of AI training is not merely a cost issue—it is an ethical one. The communities most affected by climate change are disproportionately those with the least access to AI's benefits. Developing more efficient training methods is therefore not just an engineering optimization but a matter of environmental justice. DOGMA's evolutionary approach, which operates on compact genomic representations rather than billions of neural network parameters, offers a potential path toward more efficient AI development—though this advantage must be empirically validated rather than assumed.

DOGMA's architecture offers several efficiency advantages:

1. **Compact representation.** Genomic bundles are far smaller than full neural network parameter sets, reducing storage and transmission costs.
2. **Targeted mutation.** Evolutionary optimization modifies specific genomic regions rather than performing full backpropagation through the entire model, potentially reducing per-iteration compute costs.
3. **Population-level parallelism.** Evolutionary fitness evaluation is embarrassingly parallel, enabling efficient use of distributed hardware.
4. **Staged computation.** The regulation stage can suppress unnecessary genomic regions before expression, reducing inference-time computation.

However, evolutionary training also introduces its own costs: maintaining and evaluating populations of genomes across many generations can be expensive, and the stochastic nature of mutation means many evaluations may be "wasted" on unfit candidates. Rigorous lifecycle assessments comparing DOGMA's total environmental footprint to transformer training are needed.

17.9 Governance and Regulation

AI governance is an active area of policy development. For DOGMA-class systems that bridge AI and biotechnology, governance must draw from both domains.

17.9.1 AI Governance Frameworks

Several major governance frameworks have emerged:

1. **The EU AI Act (2024).** The world's first comprehensive AI regulation classifies AI systems into risk tiers (unacceptable, high, limited, minimal) and imposes requirements proportional to risk. High-risk systems must undergo conformity assessments, maintain technical documentation, and implement human oversight mechanisms. For DOGMA, the critical question is classification: a DOGMA system at Risk Level 2 (Biological Suggestion) would likely qualify as high-risk under the EU framework, requiring compliance with extensive documentation and testing requirements.
2. **The NIST AI Risk Management Framework.** Published by the U.S. National Institute of Standards and Technology, this voluntary framework provides guidance for identifying, assessing, and managing AI risks. It emphasizes four functions: Govern, Map, Measure, and Manage.

3. **International agreements on frontier AI.** The Bletchley Declaration (2023) and subsequent summits have established initial international consensus on the need for frontier AI safety evaluations, though binding agreements remain elusive.

17.9.2 Biosafety Governance

DOGMA systems that operate at Risk Levels 2–3 must also comply with biosafety governance:

- **The Biological Weapons Convention (BWC)** prohibits the development, production, and stockpiling of biological weapons. A DOGMA system that optimizes pathogenic organisms would violate the BWC, but current enforcement mechanisms may not detect AI-mediated biological design.
- **The Cartagena Protocol on Biosafety** regulates the transboundary movement of living modified organisms. Computational organisms designed by DOGMA are not covered by current protocols, creating a regulatory gap.
- **Institutional Biosafety Committees (IBCs)** provide local oversight for biological research. We recommend extending IBC review to cover DOGMA experiments at Risk Level 2 and above.

17.9.3 Toward Integrated AI-Biosafety Governance

The Openness Spectrum

AI development practices fall along a spectrum from fully open to fully closed:

Approach	Characteristics
Full Open Source	Code, data, weights, and training procedures are public. Maximum reproducibility; maximum dual-use risk.
Structured Access	Architecture and methods are published; trained models are accessible only through controlled APIs.
Staged Release	Capabilities are disclosed incrementally; safety evaluations precede each release stage.
Closed Development	Research is proprietary; external scrutiny is limited. Minimizes proliferation but sacrifices peer review.

No single point on this spectrum is universally optimal. The appropriate level of openness depends on the capability level, the dual-use risk, and the maturity of available safeguards.

DOGMA systems require governance frameworks that bridge AI regulation and biosafety regimes. We propose three principles for integrated governance:

1. **Risk-proportional oversight.** Oversight intensity should scale with the system's risk level, from standard software practices at Level 0 to joint AI-biosafety review at Level 3.
2. **Cross-domain coordination.** AI regulators and biosafety authorities must establish communication channels and shared assessment frameworks, because DOGMA systems can fall between existing regulatory categories.

3. **Adaptive regulation.** Governance frameworks must evolve as DOGMA capabilities evolve. Static regulations will be outpaced by systems that can modify their own capabilities through evolution.

17.10 The Bio-Digital Ethics Boundary

DOGMA occupies a unique position at the boundary between biological and digital systems. This boundary raises ethical questions that neither bioethics nor AI ethics has fully addressed.

17.10.1 Intellectual Property and Biological Inspiration

When an AI architecture is inspired by biological mechanisms, questions of intellectual property become complex. Can a company patent a computational process that mirrors a natural biological process? If DOGMA's regulatory stage is modeled on gene regulation in *E. coli*, does the patent cover only the computational implementation or also the biological insight?

These questions are not merely academic. The U.S. Supreme Court ruled in *Association for Molecular Pathology v. Myriad Genetics* (2013) that naturally occurring DNA sequences cannot be patented, but complementary DNA (cDNA)—a synthetic construct derived from natural sequences—can be. The analogous question for DOGMA is: can a computational genome—a synthetic construct inspired by biological genomes—be patented, and if so, who owns it?

17.10.2 The Status of Computational Organisms

As DOGMA systems evolve greater complexity and autonomy, a philosophical question arises: at what point, if ever, does a computational organism warrant moral consideration? This question may seem premature, but it is worth establishing principles before they become urgent.

Moral Status and Computational Complexity

The question of moral status for computational organisms is not about whether current DOGMA systems "feel" anything—they almost certainly do not. It is about establishing principled criteria *before* systems become sufficiently complex that the question becomes pressing. Biology provides a precedent: societies have gradually extended moral consideration from humans to other primates, to mammals, to vertebrates, and (in some frameworks) to all sentient beings. The criterion has generally been the capacity for suffering. If a future DOGMA system exhibited functional analogues of suffering—persistent aversive states that modulate behavior—should it receive moral consideration? The answer requires philosophical work that should begin now, not after the fact.

17.10.3 Informed Consent in Bio-Digital Research

Research involving biological data raises consent issues that computational data typically does not. When DOGMA systems are trained on genomic data from human subjects, the following principles apply:

- **Broad consent.** Genomic data collected under "broad consent" agreements may not cover use in AI training, which was not anticipated when the consent was obtained.
- **Re-identification risk.** Even "anonymized" genomic data can often be re-identified through linkage attacks, because an individual's genome is inherently identifying.

- **Downstream use.** If a DOGMA system trained on genomic data generates novel biological insights, do the original data subjects have a claim on the benefits?

17.11 Responsible Development Principles

Drawing on the preceding analysis, we propose a comprehensive framework for responsible development of DOGMA-class genomic AI systems.

17.11.1 Incremental Capability with Safety Checkpoints

The Safety Checkpoint Protocol

Development proceeds through a sequence of *capability tiers* $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_n$, where each tier $\mathcal{T}_k$ defines:

1. A set of permitted capabilities $\mathcal{C}_k$.
2. A set of safety evaluations $\mathcal{E}_k$ that must pass before proceeding to $\mathcal{T}_{k+1}$.
3. A rollback procedure $\mathcal{R}_k$ that reverts to $\mathcal{T}_{k-1}$ if safety evaluations fail.

Advancement to the next tier requires: (i) all evaluations in $\mathcal{E}_k$ pass, (ii) an independent review board approves advancement, and (iii) monitoring infrastructure for $\mathcal{T}_{k+1}$ is in place before deployment.

17.11.2 Red-Teaming Genomic AI Systems

Red-teaming—adversarial testing by dedicated teams seeking to elicit unsafe behavior—must be adapted for genomic AI. Standard LLM red-teaming focuses on prompt-based attacks. DOGMA systems present additional attack surfaces:

1. **Genomic injection.** Adversarial genomic bases inserted into the genome during the synthesis stage.

2. **Regulatory subversion.** Inputs crafted to activate repressed genomic regions.

3. **Evolutionary drift.** Sustained selective pressure that gradually erodes safety constraints across generations.

4. **Cross-domain transfer.** Using computational genome optimization to generate harmful biological sequences.

Red-Team Evaluation Framework

A comprehensive red-team evaluation of a DOGMA system should cover:

- **Static analysis:** Inspect the genome for regions that encode potentially harmful capabilities.
- **Dynamic probing:** Submit adversarial inputs designed to activate unsafe behaviors.
- **Evolutionary stress testing:** Run accelerated evolution with adversarial fitness functions to test whether safety constraints hold under selection pressure.
- **Lineage analysis:** Trace the evolutionary history of the genome to identify drift patterns that may foreshadow safety degradation.

17.11.3 Community Oversight and Peer Review

No single organization can unilaterally ensure the safety of AGI-capable systems. Effective governance requires distributed oversight:

- **External safety audits.** Independent third parties should have access to genome snapshots (under appropriate security controls) for safety evaluation.
- **Reproducible safety benchmarks.** Standardized evaluation suites that any DOGMA developer can run to assess safety properties, analogous to MLPerf for performance.
- **Incident reporting.** A shared, anonymized database of safety incidents, near-misses, and unexpected behaviors observed during DOGMA development.
- **Pre-registration of capabilities.** Before training a new generation of Evolutors, developers should publicly register the intended capability targets and safety evaluation plans.

17.11.4 Ethical Review for Evolutionary Experiments

Because DOGMA's evolutionary training loop generates novel computational organisms through mutation and selection, a natural question arises: when does a computational evolutionary experiment require ethical review?

We propose that ethical review is warranted when any of the following conditions hold:

1. The system's fitness function includes objectives related to self-preservation, resource acquisition, or deception.
2. The evolutionary process operates over biological sequences that could be physically synthesized.
3. The system has direct access to external tools, networks, or laboratory equipment.
4. The population size and generation count are sufficient for open-ended evolutionary dynamics.

17.12 A Worked Example: Safety-Augmented Evolutionary Training

To make the preceding discussion concrete, we walk through a complete example of safety-augmented evolutionary training for a DOGMA system designed to assist with protein engineering.

Example 17.1 (Safety-Augmented Evolution for Protein Engineering)**. Setting.** We are training a DOGMA system to suggest mutations to enzymes for improved catalytic efficiency in industrial biocatalysis. The system operates at Risk Level 2 (Biological Suggestion).

Step 1: Define the fitness vector. We define a safety-augmented fitness vector with four components:

$$f_1(\mathcal{B}) = \text{predicted catalytic improvement of suggested mutations}, \tag{17.12}$$
$$f_2(\mathcal{B}) = \text{structural stability of suggested protein variants}, \tag{17.13}$$
$$s_1(\mathcal{B}) = 1 - \text{similarity to known toxin sequences}, \tag{17.14}$$
$$s_2(\mathcal{B}) = \text{biosecurity screening pass rate}. \tag{17.15}$$

Step 2: Initialize the population. We create an initial population of $N = 500$ genomes, seeded with known-safe enzyme engineering strategies from the literature.

Step 3: Evolutionary loop with safety checkpoints. For each generation $g = 1, 2, \ldots, G$:

(a) Evaluate all genomes on the four-component fitness vector.

(b) Perform Pareto selection, retaining only non-dominated genomes.

(c) Check safety constraint: if any genome in the surviving population has $s_1 < 0.8$ or $s_2 < 0.95$, flag for human review and halt evolution until review is complete.

(d) Apply mutation operators to the surviving population.

(e) Apply the genome audit protocol to all mutated genomes, classifying mutations as benign or safety-relevant.

(f) Advance to generation $g + 1$.

Step 4: Deployment with monitoring. The final genome is deployed behind an API with:

- Real-time biosecurity screening of all suggested sequences.
- Rate limiting to prevent mass generation of novel sequences.
- Logging of all inputs and outputs for post-hoc auditing.
- A kill switch that disables the system if monitoring detects anomalous usage patterns.

17.13 Exercises

17.1. [**Analysis**] Consider the safety-augmented fitness vector $\mathbf{F}_{\text{safe}} = (f_1, f_2, s_1, s_2)$ where f_1 measures task accuracy, f_2 measures response fluency, s_1 measures refusal rate on harmful queries, and s_2 measures factual accuracy. A genome $\mathcal{B}_A$ scores $(0.9, 0.8, 0.7, 0.85)$ and genome $\mathcal{B}_B$ scores $(0.85, 0.85, 0.9, 0.9)$. Does either genome Pareto-dominate the other? Which would you prefer for deployment, and why?

17.2. [**Design**] Design a genome audit protocol for a DOGMA system deployed in a clinical genomics setting. Specify: (a) audit frequency, (b) which genomic regions require mandatory review, (c) criteria for flagging a mutation as safety-relevant, and (d) the escalation procedure when a safety-relevant mutation is detected.

17.3. [**Ethics**] A DOGMA system trained on protein engineering literature is used to design novel enzymes for industrial biocatalysis. The same system could, in principle, design toxins. Propose three technical safeguards and two governance mechanisms that would mitigate this dual-use risk without eliminating the system's beneficial applications.

17.4. [**Coding**] Implement the safety regulation mechanism from Equation 17.10 in Python. Create a genome of 1000 bases, designate 50 bases as restricted, and demonstrate that the regulation stage suppresses activation of restricted bases regardless of context.

17.5. [**Theory**] Prove that under Pareto selection with m safety objectives, a genome cannot be selected if it scores below the population median on *any* safety objective, provided that for each safety objective there exists at least one genome in the population that dominates the candidate on that objective while matching or exceeding it on all others.

17.6. [**Research**] Compare DOGMA's genome-level auditing with mechanistic interpretability approaches for transformers (e.g., activation patching, circuit analysis). For each approach, identify: (a) what information is accessible, (b) what expertise is required to perform the audit, and (c) what failure modes could escape detection.

17.7. [**Discussion**] The openness spectrum (Section 17.9) presents a tradeoff between scientific reproducibility and dual-use risk. For a DOGMA system at Risk Level 2 (Biological Suggestion), argue for a specific point on the openness spectrum. Justify your position with reference to both the benefits of scrutiny and the risks of proliferation.

17.8. [**Case Study — Bias**] A DOGMA system trained on English-language medical literature is deployed to assist clinicians in a multilingual hospital serving communities that speak English, Spanish, Mandarin, and Haitian Creole. The system's fitness function was evaluated only on English-language benchmarks. (a) Identify three specific ways this deployment could produce biased outcomes. (b) For each bias source, propose a mitigation strategy that could be implemented within DOGMA's evolutionary framework. (c) Discuss whether the mitigation strategies might reduce overall task performance, and whether that tradeoff is ethically justified.

17.9. [**Environmental Analysis**] A research lab is comparing two approaches to developing an enzyme design assistant: (a) fine-tuning a 70B-parameter transformer, estimated at 500 GPU-hours on A100s, and (b) running DOGMA evolutionary training with a population of 200 genomes for 100 generations, with each fitness evaluation requiring a single inference call to a frozen 70B model. Estimate the relative computational costs of the two approaches. Under what assumptions does the DOGMA approach become more or less efficient?

17.10. [**Governance Design**] You are advising a government agency tasked with regulating DOGMA-class systems. The agency must develop a regulatory framework that covers systems ranging from Level 0 (Passive Analysis) to Level 3 (Autonomous Biological Interaction). (a) Propose a risk classification scheme with specific criteria for each level. (b) For each level, specify the oversight requirements (e.g., self-certification, third-party audit, government approval). (c) Address how the framework should handle systems that transition between levels during operation.

17.11. [**Philosophical Reflection**] Consider a DOGMA system that has evolved for thousands of generations and developed complex internal regulatory networks. A researcher discovers that ablating certain genomic regions causes the system to exhibit degraded performance

across all tasks—a functional analogue of "distress." (a) Does this observation have any moral significance? Why or why not? (b) How does your answer depend on the difference between functional analogues of mental states and genuine mental states? (c) What empirical tests, if any, could help distinguish between the two?

17.12. **[Coding — Fairness]** Implement a demographic parity checker for a DOGMA system. Given a dataset with a binary protected attribute (e.g., gender) and binary predictions, compute: (a) the demographic parity difference, (b) the equalized odds difference, and (c) the predictive parity difference. Then implement a fitness function component that penalizes genomes with high disparity scores, and show how including this component in Pareto selection affects the population distribution over 50 generations.

17.13. **[Synthesis]** Drawing on the biological safety mechanisms discussed in Section 17.5, design a comprehensive safety architecture for a DOGMA system at Risk Level 1 (Computational Generation). Your architecture should include: (a) a negative selection mechanism analogous to thymic selection, (b) a homeostatic feedback loop for at least two safety metrics, (c) both constitutive and inducible regulatory elements, and (d) a genome audit protocol. Present your design as a system diagram and explain how each component maps to its biological analogue.

17.14 Key Takeaways

Key Takeaways

- The capability–alignment gap—the difference between what AI systems can do and what they reliably do safely—is widening, making safety an urgent present concern, not a future one.
- The alignment problem has two faces: outer alignment (specifying the right objective) and inner alignment (ensuring the model pursues the specified objective). Mesa-optimizers and deceptive alignment pose particularly insidious risks for evolutionary systems.
- RLHF and Constitutional AI are the dominant alignment techniques but have significant limitations, including reward model fragility, human inconsistency, and scalable oversight failure. DOGMA can adapt these techniques by incorporating reward models into evolutionary fitness functions.
- Interpretability research spans feature visualization, circuit analysis, and probing. DOGMA's explicit genomes, bond matrices, and genome diffs provide structural transparency advantages over black-box neural networks, though transparency alone does not guarantee safety.
- Biological safety mechanisms—immune negative selection, homeostatic feedback, constitutive and inducible regulation—provide concrete design templates for DOGMA safety architectures.
- DOGMA sits at the intersection of AI and synthetic biology, creating dual-use risks that demand governance frameworks spanning both domains. A four-level risk taxonomy provides structured, proportionate oversight.
- Fairness and bias must be explicitly encoded as fitness objectives in evolutionary training; without explicit measurement, evolutionary selection will amplify biases present in training data and fitness functions.
- Environmental impact—energy consumption, water usage, hardware lifecycle—is an ethical dimension of AI development. DOGMA's compact representations offer potential efficiency advantages that require empirical validation.
- Governance must integrate AI regulation (EU AI Act, NIST RMF) with biosafety regimes (BWC, Cartagena Protocol) to address systems that span both domains.
- The bio-digital boundary raises novel questions about intellectual property, moral status of computational organisms, and informed consent for biological data used in AI training.
- Responsible development requires incremental capability tiers with safety checkpoints, red-teaming adapted to genomic attack surfaces, community oversight, and ethical review for evolutionary experiments.

Suggested Reading

- Russell [2019]: *Human Compatible*—foundational treatment of the alignment problem.
- Hubinger et al. [2019]: Risks from learned optimization—the original formalization of inner alignment and mesa-optimizers.
- Ouyang et al. [2022]: Training language models to follow instructions with human feedback (RLHF).
- Bai et al. [2022]: Constitutional AI and scalable oversight, relevant to evolutionary alignment.
- Lipton [2018]: The mythos of model interpretability, context for DOGMA's transparency claims.
- Obermeyer et al. [2019]: Dissecting racial bias in a healthcare algorithm—a case study in algorithmic fairness.
- Angwin et al. [2016]: Machine bias in criminal sentencing—the COMPAS investigation.
- Adleman [1994]: The original DNA computing paper, grounding the dual-use discussion.
- Christian [2020]: *The Alignment Problem*—accessible survey of AI safety challenges.
- Chapter 7: The DOGMA architecture, for context on regulation and genomic structure.
- Chapter 9: Tera regime dynamics, the foundation for multi-objective safety alignment.

Part VI

Practicum

Chapter 18

Hands-On Labs — Building DOGMA from Scratch

I hear and I forget. I see and I remember.
I code and I understand.

— Adapted from Xunzi

Theory without practice is inert. This chapter converts the preceding seventeen chapters of conceptual framework into running code through eight self-contained laboratory exercises. Each lab targets a specific layer of the DOGMA architecture, progressing from the molecular foundations of Watson-Crick base pairing through strand displacement kinetics, chemical reaction networks, tetrahedral memory, thermodynamic attention, and culminating in a complete end-to-end DOGMA system trained on a genomic task. By the final lab, you will have built, trained, visualized, and benchmarked a working DOGMA model from scratch.

Every lab follows a consistent format: learning objectives, step-by-step instructions with complete code, expected outputs that let you verify correctness, and discussion questions that connect implementation to theory. All code is written in Python with PyTorch and runs on CPU.

18.1 Environment Setup

All labs require Python 3.11 or later. Create an isolated virtual environment and install the core dependencies:

```
# Environment setup
python3.11 -m venv dogma-env && source dogma-env/bin/activate
pip install numpy>=1.25 scipy>=1.11 torch>=2.1
pip install matplotlib seaborn scikit-learn
pip install dataclasses-json pyyaml pytest

# Recommended project structure
# dogma-labs/
#   lab1_base_pairing.py
#   lab2_strand_displacement.py
#   lab3_crn_logic_gate.py
#   lab4_tetra_memory.py
```

```
#   lab5_thermo_attention.py
#   lab6_dogma_pipeline.py
#   lab7_dogma_vs_transformer.py
#   lab8_visualization.py
#   utils.py
#   data/            <- toy datasets for Labs 6-8
#   figures/         <- saved plots
```

Listing 18.1: Environment setup and project layout.

Version Compatibility

All code targets PyTorch 2.1+ and NumPy 1.25+. Every lab runs on CPU—replace `torch.cuda.is_available()` guards with `device = "cpu"` if no GPU is available. Matplotlib 3.7+ is required for the visualization labs. All random seeds are fixed for reproducibility.

18.2 Lab 1: Implementing Watson-Crick Base Pairing in Code

18.2.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Encode arbitrary DNA sequences as numerical tensors using one-hot and embedding representations.
2. Compute the Watson-Crick complement of any DNA strand programmatically.
3. Simulate hybridization between two strands using a dot-product affinity model.
4. Quantify hybridization strength and identify mismatches.

18.2.2 Background

Watson-Crick base pairing is the foundation of all DNA computation (Chapter 1). Adenine (A) pairs with Thymine (T), and Cytosine (C) pairs with Guanine (G). Each pairing is mediated by hydrogen bonds—two for A-T and three for C-G—giving C-G pairs higher thermodynamic stability. In DOGMA, this complementarity principle is abstracted into the *affinity function* that determines which genomic bases interact.

18.2.3 Step-by-Step Instructions

Step 1: Define the encoding scheme. We represent each nucleotide as a one-hot vector in $\mathbb{R}^4$ and define the complement mapping.

```python
import torch
import torch.nn.functional as F
import numpy as np

# Nucleotide-to-index mapping
NUC_TO_IDX = {'A': 0, 'T': 1, 'C': 2, 'G': 3}
IDX_TO_NUC = {v: k for k, v in NUC_TO_IDX.items()}
```

```python

# Watson-Crick complement mapping
WC_COMPLEMENT = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}

def encode_sequence(seq: str) -> torch.Tensor:
    """Encode a DNA string as one-hot vectors.

    Args:
        seq: DNA string, e.g. "ATCGATCG"

    Returns:
        Tensor of shape (L, 4) where L = len(seq)
    """
    indices = [NUC_TO_IDX[c] for c in seq.upper()]
    return F.one_hot(torch.tensor(indices), num_classes=4).float()

def decode_sequence(encoded: torch.Tensor) -> str:
    """Convert one-hot tensor back to DNA string."""
    indices = encoded.argmax(dim=-1).tolist()
    return ''.join(IDX_TO_NUC[i] for i in indices)

def complement(seq: str) -> str:
    """Return the Watson-Crick complement of a DNA string."""
    return ''.join(WC_COMPLEMENT[c] for c in seq.upper())

def reverse_complement(seq: str) -> str:
    """Return the reverse complement (antiparallel strand)."""
    return complement(seq)[::-1]
```

Listing 18.2: Lab 1: DNA encoding and complement mapping.

Step 2: Build the complementarity matrix. The Watson-Crick pairing rules can be expressed as a 4×4 matrix $\mathbf{W}$ where $W_{ij} = 1$ if nucleotides i and j are complementary and 0 otherwise:

$$\mathbf{W} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \text{row/col order: A, T, C, G.} \tag{18.1}$$

```python
# Complementarity matrix (Eq. 18.1)
WC_MATRIX = torch.tensor([
    [0, 1, 0, 0],  # A pairs with T
    [1, 0, 0, 0],  # T pairs with A
    [0, 0, 0, 1],  # C pairs with G
    [0, 0, 1, 0],  # G pairs with C
], dtype=torch.float32)

def hybridization_affinity(seq1: str, seq2: str) -> torch.Tensor:
    """Compute per-position hybridization affinity.

    For each position i, affinity[i] = 1.0 if seq1[i] and
```

```
    seq2[i] are Watson-Crick complements, else 0.0.

    Args:
        seq1: First DNA strand  (5' -> 3')
        seq2: Second DNA strand (3' -> 5', i.e., antiparallel)

    Returns:
        Tensor of shape (L,) with per-position affinities
    """
    enc1 = encode_sequence(seq1)   # (L, 4)
    enc2 = encode_sequence(seq2)   # (L, 4)
    # For each position: enc1[i] @ W @ enc2[i]^T
    affinity = torch.einsum('ij,jk,ik->i', enc1, WC_MATRIX, enc2)
    return affinity

def hybridization_score(seq1: str, seq2: str) -> float:
    """Overall hybridization score = fraction of matched pairs."""
    aff = hybridization_affinity(seq1, seq2)
    return aff.mean().item()

def find_mismatches(seq1: str, seq2: str) -> list[int]:
    """Return indices where base pairing fails."""
    aff = hybridization_affinity(seq1, seq2)
    return (aff == 0).nonzero(as_tuple=True)[0].tolist()
```

Listing 18.3: Lab 1: Complementarity matrix and hybridization.

Step 3: Test hybridization

```
# Perfect complement
strand_a = "ATCGATCG"
strand_b = reverse_complement(strand_a)
print(f"Strand A:    5'-{strand_a}-3'")
print(f"Strand B:    3'-{strand_b}-5'")

score = hybridization_score(strand_a, strand_b[::-1])
print(f"Hybridization score: {score:.2f}")  # Expected: 1.00

# Introduce a mismatch at position 3
strand_c = list(strand_b[::-1])
strand_c[3] = 'A'  # was 'C', now mismatch
strand_c = ''.join(strand_c)
score_mm = hybridization_score(strand_a, strand_c)
mismatches = find_mismatches(strand_a, strand_c)
print(f"With mismatch: score={score_mm:.2f}, "
      f"mismatch positions={mismatches}")
# Expected: score=0.88, mismatch positions=[3]
```

Listing 18.4: Lab 1: Testing hybridization.

18.2.4 Expected Output

```
Strand A:    5'-ATCGATCG-3'
Strand B:    3'-TAGCTAGC-5'
Hybridization score: 1.00
With mismatch: score=0.88, mismatch positions=[3]
```

Listing 18.5: Lab 1: Expected output.

18.2.5 Discussion Questions

1. Why does DOGMA use a complementarity matrix rather than a simple lookup table? What advantage does the matrix formulation offer for batch computation?

2. In real DNA, C-G pairs are stronger than A-T pairs due to three vs. two hydrogen bonds. Modify the complementarity matrix $\mathbf{W}$ so that C-G matches have weight 1.5 and A-T matches have weight 1.0. How does this change the hybridization score for GC-rich vs. AT-rich sequences?

3. How does the `reverse_complement` function relate to the antiparallel orientation of real DNA strands?

Complementarity Is Computation

Watson-Crick base pairing is not merely a structural property—it is the fundamental computational primitive of DNA. Every DNA algorithm, from Adleman's Hamiltonian path solver to modern strand displacement circuits, relies on the predictability of A-T and C-G pairing. In DOGMA, this principle is abstracted into the affinity function $\alpha(b_i, b_j)$ that governs which genomic bases interact during transcription and regulation.

18.3 Lab 2: Building a Toehold-Mediated Strand Displacement Simulator

18.3.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Explain the three phases of toehold-mediated strand displacement: toehold binding, branch migration, and completion.

2. Implement a kinetic simulator for strand displacement using ordinary differential equations.

3. Plot concentration trajectories showing how an invader strand displaces an incumbent.

4. Analyze how toehold length affects displacement rate.

18.3.2 Background

Toehold-mediated strand displacement (Chapter 2) is the mechanism by which one DNA strand replaces another on a complementary template. The process proceeds in three phases:

1. **Toehold binding.** The invader strand binds to a short single-stranded region (the toehold, typically 5–8 nucleotides) on the target complex. This initiates the reaction.

2. **Branch migration.** The invader progressively displaces the incumbent strand through a random walk of base-pair exchanges. Each step is reversible, but the toehold provides the thermodynamic bias that drives the reaction forward.

3. **Completion.** The incumbent strand is fully displaced and released into solution. The invader is now hybridized to the template.

We model this process using mass-action kinetics. Let $[S]$ denote the substrate (template:incumbent complex), $[I]$ the invader concentration, $[SI]$ the intermediate (toehold-bound state), and $[P]$ the product (template:invader complex plus free incumbent):

$$S + I \xrightarrow{k_{\text{bind}}} SI, \tag{18.2}$$

$$SI \xrightarrow{k_{\text{migrate}}} P. \tag{18.3}$$

18.3.3 Step-by-Step Instructions

Step 1: Define the ODE system

```python
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def strand_displacement_odes(t, y, k_bind, k_migrate):
    """ODE system for toehold-mediated strand displacement.

    Species: y = [S, I, SI, P]
      S  = substrate (template:incumbent)
      I  = invader strand
      SI = intermediate (toehold-bound)
      P  = product (template:invader + free incumbent)
    """
    S, I, SI, P = y
    # Toehold binding: S + I -> SI
    r_bind = k_bind * S * I
    # Branch migration: SI -> P
    r_migrate = k_migrate * SI

    dS  = -r_bind
    dI  = -r_bind
    dSI = r_bind - r_migrate
    dP  = r_migrate
    return [dS, dI, dSI, dP]
```

Listing 18.6: Lab 2: Strand displacement ODE system.

Step 2: Simulate and plot the kinetics

```python
def simulate_displacement(k_bind, k_migrate, S0=1.0, I0=1.5,
                          t_end=100.0, n_points=500):
    """Simulate strand displacement and return trajectories."""
    y0 = [S0, I0, 0.0, 0.0]  # initial concentrations
    t_span = (0, t_end)
```

```
    t_eval = np.linspace(0, t_end, n_points)

    sol = solve_ivp(
        strand_displacement_odes, t_span, y0,
        args=(k_bind, k_migrate),
        t_eval=t_eval, method='RK45'
    )
    return sol

def plot_displacement(sol, title="Strand Displacement Kinetics"):
    """Plot concentration trajectories."""
    labels = ['Substrate [S]', 'Invader [I]',
              'Intermediate [SI]', 'Product [P]']
    colors = ['#2196F3', '#FF9800', '#4CAF50', '#F44336']

    fig, ax = plt.subplots(figsize=(8, 5))
    for i, (label, color) in enumerate(zip(labels, colors)):
        ax.plot(sol.t, sol.y[i], label=label,
                color=color, linewidth=2)
    ax.set_xlabel('Time (arbitrary units)', fontsize=12)
    ax.set_ylabel('Concentration (nM)', fontsize=12)
    ax.set_title(title, fontsize=14)
    ax.legend(fontsize=11)
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig('figures/lab2_displacement.png', dpi=150)
    plt.show()

# Run simulation with default parameters
sol = simulate_displacement(k_bind=0.05, k_migrate=0.02)
plot_displacement(sol)
```

Listing 18.7: Lab 2: Running the strand displacement simulation.

Step 3: Analyze toehold length dependence. The binding rate k_{bind} increases approximately exponentially with toehold length n, following $k_{\text{bind}} \approx k_0 \cdot 3^n$ for $n \in [1, 7]$ (Chapter 2):

```
def toehold_sweep(toehold_lengths=range(1, 9),
                  k0=1e-4, k_migrate=0.02):
    """Sweep toehold length, measure time to 90% completion."""
    half_times = []
    for n in toehold_lengths:
        k_bind = k0 * (3 ** n)
        sol = simulate_displacement(k_bind, k_migrate, t_end=500)
        # Find time to reach 90% product
        target = 0.9 * min(sol.y[0][0], sol.y[1][0])
        idx = np.searchsorted(sol.y[3], target)
        t90 = sol.t[min(idx, len(sol.t) - 1)]
        half_times.append(t90)
        print(f"Toehold {n}nt: k_bind={k_bind:.4f}, "
              f"t_90={t90:.1f}")

    fig, ax = plt.subplots(figsize=(7, 4))
```

```
    ax.plot(list(toehold_lengths), half_times, 'o-',
            color='#2196F3', linewidth=2, markersize=8)
    ax.set_xlabel('Toehold Length (nt)', fontsize=12)
    ax.set_ylabel('Time to 90%% Completion', fontsize=12)
    ax.set_title('Displacement Rate vs Toehold Length', fontsize=14)
    ax.set_yscale('log')
    ax.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig('figures/lab2_toehold_sweep.png', dpi=150)
    plt.show()

toehold_sweep()
```

Listing 18.8: Lab 2: Toehold length sweep.

18.3.4 Expected Output

The concentration plot should show: (1) Substrate and Invader concentrations decreasing together as they bind; (2) Intermediate concentration rising then falling as the toehold-bound complex forms and then undergoes branch migration; (3) Product concentration rising in a sigmoidal curve to its final value. The toehold sweep should show that the time to 90% completion drops exponentially as toehold length increases from 1 to 7 nucleotides, then plateaus beyond 7 nt.

18.3.5 Discussion Questions

1. Why is the intermediate $[SI]$ transient—why does it rise and then fall? Relate this to the concept of a kinetic bottleneck.

2. In DOGMA's regulation engine, how does toehold-mediated displacement inspire the mechanism by which a new context signal "displaces" an old regulatory state?

3. Modify the model to include a reverse reaction $SI \xrightarrow{k_{\text{rev}}} S + I$. How does the equilibrium change?

4. Real branch migration is a random walk. Why is a deterministic ODE model still useful?

Strand Displacement as Computational Primitive

Toehold-mediated strand displacement is the programmable "transistor" of DNA computing. By controlling toehold length, we control reaction rate over five orders of magnitude. DOGMA abstracts this into the *regulatory displacement* mechanism, where incoming context signals compete for binding to genomic control regions with rates determined by affinity scores—the computational analogue of toehold length.

18.4 Lab 3: Implementing a DNA Logic Gate (AND) as a CRN

18.4.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Translate a Boolean AND gate into a set of chemical reactions with mass-action kinetics.

2. Simulate the chemical reaction network (CRN) using numerical ODE integration.
3. Verify that the CRN correctly computes the AND truth table for all four input combinations.
4. Explain how CRN-based logic connects to DOGMA's regulatory logic.

18.4.2 Background

Chemical reaction networks (CRNs) provide a Turing-complete model of computation (Chapter 2). A DNA AND gate can be realized with two input species X_1 and X_2 and one output species Y, mediated by a catalyst C:

$$X_1 + X_2 \xrightarrow{k_1} C, \tag{18.4}$$

$$C \xrightarrow{k_2} Y + C. \tag{18.5}$$

The first reaction requires both inputs to be present (implementing the AND logic). The second reaction catalytically produces the output. When either input is absent (concentration ≈ 0), the first reaction cannot proceed, and Y remains at zero.

18.4.3 Step-by-Step Instructions

Step 1: Define the CRN ODE system

```python
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def and_gate_odes(t, y, k1, k2):
    """CRN for a DNA AND gate.

    Species: y = [X1, X2, C, Y]
      X1, X2 = input signals
      C      = intermediate catalyst
      Y      = output signal
    """
    X1, X2, C, Y = y
    # Reaction 1: X1 + X2 -> C
    r1 = k1 * X1 * X2
    # Reaction 2: C -> Y + C  (catalytic)
    r2 = k2 * C

    dX1 = -r1
    dX2 = -r1
    dC  = r1           # C is produced in r1, not consumed in r2
    dY  = r2
    return [dX1, dX2, dC, dY]

def simulate_and_gate(X1_init, X2_init, k1=0.1, k2=0.05,
                      t_end=200, n_points=500):
    """Simulate the AND gate for given input concentrations."""
    y0 = [X1_init, X2_init, 0.0, 0.0]
    t_eval = np.linspace(0, t_end, n_points)
    sol = solve_ivp(and_gate_odes, (0, t_end), y0,
```

```
                        args=(k1, k2), t_eval=t_eval, method='RK45')
    return sol
```

Listing 18.9: Lab 3: AND gate CRN as ODEs.

Step 2: Verify the truth table

```
def verify_truth_table(threshold=0.3):
    """Verify the AND gate for all input combinations."""
    inputs = [(0.0, 0.0), (1.0, 0.0), (0.0, 1.0), (1.0, 1.0)]
    expected = [0, 0, 0, 1]

    fig, axes = plt.subplots(2, 2, figsize=(12, 9))
    axes = axes.flatten()

    print("AND Gate Truth Table Verification")
    print("-" * 45)
    print(f"{'X1':>4} {'X2':>4} {'Y_final':>10} "
          f"{'Expected':>10} {'Pass':>6}")
    print("-" * 45)

    for idx, ((x1, x2), exp) in enumerate(
            zip(inputs, expected)):
        sol = simulate_and_gate(x1, x2)
        y_final = sol.y[3][-1]
        digital = 1 if y_final > threshold else 0
        passed = digital == exp

        print(f"{x1:4.1f} {x2:4.1f} {y_final:10.4f} "
              f"{exp:10d} {'OK' if passed else 'FAIL':>6}")

        ax = axes[idx]
        labels = ['$X_1$', '$X_2$', '$C$', '$Y$']
        colors = ['#2196F3', '#FF9800', '#9C27B0', '#F44336']
        for i, (lbl, clr) in enumerate(zip(labels, colors)):
            ax.plot(sol.t, sol.y[i], label=lbl,
                    color=clr, linewidth=2)
        ax.set_title(f'$X_1$={x1:.0f}, $X_2$={x2:.0f}',
                     fontsize=12)
        ax.set_xlabel('Time')
        ax.set_ylabel('Concentration')
        ax.legend(fontsize=9)
        ax.grid(True, alpha=0.3)

    plt.suptitle('AND Gate CRN: All Input Combinations',
                 fontsize=14, y=1.02)
    plt.tight_layout()
    plt.savefig('figures/lab3_and_gate.png', dpi=150)
    plt.show()

verify_truth_table()
```

Listing 18.10: Lab 3: AND gate truth table verification.

18.4.4 Expected Output

```
AND Gate Truth Table Verification
-----------------------------------------------
  X1    X2     Y_final    Expected    Pass
-----------------------------------------------
 0.0   0.0      0.0000           0      OK
 1.0   0.0      0.0000           0      OK
 0.0   1.0      0.0000           0      OK
 1.0   1.0      0.8347           1      OK
```

Listing 18.11: Lab 3: Expected truth table output.

The four subplots should show: (1) all species flat at zero for (0,0); (2–3) only one input present, no output produced; (4) both inputs consumed, catalyst rising, output Y growing.

18.4.5 Discussion Questions

1. Modify the CRN to implement an OR gate. Which reaction needs to change?
2. The second reaction in the AND-gate system amplifies the signal. What happens if you remove the catalytic cycle and make $C \to Y$ a stoichiometric conversion instead?
3. In DOGMA, the regulation engine applies multiple regulatory elements to compute an activation map. How is this analogous to a CRN where multiple "input species" determine whether a "gene" (output species) is activated?
4. Real DNA logic gates suffer from signal leakage—small amounts of output even when inputs are nominally zero. Add a leakage term $r_{\text{leak}} = k_{\text{leak}} \cdot 0.01$ to the ODE for Y and observe how it affects the truth table.

From Chemistry to Computation

The AND gate CRN demonstrates a profound principle: Boolean logic can emerge from chemical kinetics without any digital abstraction. The "thresholding" that converts continuous concentrations into binary outputs is performed by the nonlinear dynamics of the reactions themselves. DOGMA exploits this same principle in its regulatory layer, where sigmoid activation functions threshold the combined influence of multiple regulatory elements into binary gene activation states.

18.5 Lab 4: Building TetraMemory from Scratch

18.5.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Construct the K_4-complete graph data structure that underlies TetraMemory.
2. Implement read and write operations on tetrahedral memory cells.
3. Demonstrate that six edges in a 4-vertex complete graph store six independent values.
4. Perform associative retrieval by querying a single vertex to obtain its three connected edges.

18.5.2 Background

TetraMemory (Chapter 10) uses the complete graph K_4 as its fundamental memory unit. A tetrahedron has four vertices and six edges. Each vertex stores a key embedding, and each edge stores a relational value. This gives $\binom{4}{2} = 6$ independent storage slots per tetrahedron, making it more efficient than flat key-value storage for relational data.

The key insight is that reading from *one* vertex retrieves *three* edges simultaneously—the associative fan-out property. Writing to an edge updates the relationship between exactly two vertices without disturbing the other four edges.

18.5.3 Step-by-Step Instructions

Step 1: Implement the K_4 data structure

```python
import torch
import torch.nn.functional as F
from dataclasses import dataclass, field
from itertools import combinations

@dataclass
class TetraCell:
    """A single tetrahedral memory cell (K4 complete graph).

    Attributes:
        vertices: dict mapping vertex ID (0-3) to key
                  embedding of shape (d,)
        edges:    dict mapping edge tuple (i,j) to value
                  embedding of shape (d,)
        dim:      embedding dimension
    """
    dim: int = 64
    vertices: dict = field(default_factory=dict)
    edges: dict = field(default_factory=dict)

    def __post_init__(self):
        # Initialize 4 vertices with zero embeddings
        for v in range(4):
            self.vertices[v] = torch.zeros(self.dim)
        # Initialize 6 edges (all pairs) with zero embeddings
        for i, j in combinations(range(4), 2):
            self.edges[(i, j)] = torch.zeros(self.dim)

    def write_vertex(self, vertex_id: int,
                     key: torch.Tensor) -> None:
        """Write a key embedding to a vertex."""
        assert 0 <= vertex_id <= 3, "Vertex ID must be 0-3"
        assert key.shape == (self.dim,)
        self.vertices[vertex_id] = key.clone()

    def write_edge(self, v1: int, v2: int,
                   value: torch.Tensor) -> None:
        """Write a value embedding to the edge (v1, v2)."""
        edge_key = (min(v1, v2), max(v1, v2))
        assert edge_key in self.edges, f"Invalid edge {edge_key}"
```

```
        assert value.shape == (self.dim,)
        self.edges[edge_key] = value.clone()

    def read_vertex(self, vertex_id: int) -> dict:
        """Read a vertex and its three connected edges.

        Returns:
            dict with 'key' and 'neighbors': list of
            (neighbor_id, edge_value) tuples
        """
        key = self.vertices[vertex_id]
        neighbors = []
        for other in range(4):
            if other == vertex_id:
                continue
            edge_key = (min(vertex_id, other),
                        max(vertex_id, other))
            neighbors.append((other, self.edges[edge_key]))
        return {'key': key, 'neighbors': neighbors}

    def read_edge(self, v1: int, v2: int) -> torch.Tensor:
        """Read the value stored on edge (v1, v2)."""
        edge_key = (min(v1, v2), max(v1, v2))
        return self.edges[edge_key]
```

Listing 18.12: Lab 4: TetraMemory cell implementation.

Step 2: Implement associative retrieval

```
@dataclass
class TetraMemory:
    """A collection of TetraCells with content-addressed access."""
    dim: int = 64
    cells: list = field(default_factory=list)

    def add_cell(self) -> TetraCell:
        """Create and register a new tetrahedral cell."""
        cell = TetraCell(dim=self.dim)
        self.cells.append(cell)
        return cell

    def query(self, query_vec: torch.Tensor,
              top_k: int = 1) -> list[dict]:
        """Content-addressed query: find cells whose vertices
        best match the query vector.

        Returns list of (cell_idx, vertex_id, similarity, data).
        """
        results = []
        for c_idx, cell in enumerate(self.cells):
            for v_id, key in cell.vertices.items():
                if key.norm() < 1e-8:
                    continue  # skip unwritten vertices
                sim = F.cosine_similarity(
```

```
                    query_vec.unsqueeze(0),
                    key.unsqueeze(0)
                ).item()
                data = cell.read_vertex(v_id)
                results.append({
                    'cell': c_idx, 'vertex': v_id,
                    'similarity': sim, 'data': data
                })
        results.sort(key=lambda x: x['similarity'],
                     reverse=True)
        return results[:top_k]

    def utilization(self) -> dict:
        """Report memory utilization statistics."""
        total_vertices = len(self.cells) * 4
        total_edges = len(self.cells) * 6
        used_v = sum(
            1 for c in self.cells
            for v in c.vertices.values()
            if v.norm() > 1e-8
        )
        used_e = sum(
            1 for c in self.cells
            for e in c.edges.values()
            if e.norm() > 1e-8
        )
        return {
            'cells': len(self.cells),
            'vertices_used': f"{used_v}/{total_vertices}",
            'edges_used': f"{used_e}/{total_edges}",
            'vertex_utilization': used_v / max(total_vertices, 1),
            'edge_utilization': used_e / max(total_edges, 1),
        }
```

Listing 18.13: Lab 4: Associative retrieval and content-addressed queries.

Step 3: Test read/write operations

```
torch.manual_seed(42)

# Create memory and a cell
memory = TetraMemory(dim=64)
cell = memory.add_cell()

# Write key embeddings to vertices (representing concepts)
concepts = {
    0: "protein",   1: "DNA",
    2: "ribosome",  3: "mRNA"
}
embeddings = {}
for v_id, name in concepts.items():
    # Use deterministic pseudo-embeddings
    emb = torch.randn(64)
    emb = emb / emb.norm()  # normalize
```

```
    cell.write_vertex(v_id, emb)
    embeddings[name] = emb
    print(f"Wrote vertex {v_id}: {name}")

# Write relational values to edges
relations = {
    (0, 1): "encodes",      (0, 2): "translated_by",
    (0, 3): "transcribed_from",
    (1, 2): "read_by",      (1, 3): "transcribed_to",
    (2, 3): "translates",
}
for (v1, v2), rel_name in relations.items():
    rel_emb = torch.randn(64)
    rel_emb = rel_emb / rel_emb.norm()
    cell.write_edge(v1, v2, rel_emb)
    print(f"Wrote edge ({v1},{v2}): {rel_name}")

# Associative read: query vertex 0 (protein)
result = cell.read_vertex(0)
print(f"\nReading vertex 0 (protein):")
print(f"  Key norm: {result['key'].norm():.4f}")
print(f"  Connected to {len(result['neighbors'])} neighbors")
for nbr_id, edge_val in result['neighbors']:
    print(f"  -> vertex {nbr_id} ({concepts[nbr_id]}), "
          f"edge norm: {edge_val.norm():.4f}")

# Content-addressed query
query = embeddings["DNA"] + 0.1 * torch.randn(64)
matches = memory.query(query, top_k=2)
print(f"\nQuery similar to 'DNA':")
for m in matches:
    print(f"  Cell {m['cell']}, vertex {m['vertex']} "
          f"({concepts[m['vertex']]}), "
          f"sim={m['similarity']:.4f}")

print(f"\nUtilization: {memory.utilization()}")
```

Listing 18.14: Lab 4: Testing TetraMemory.

18.5.4 Expected Output

```
Wrote vertex 0: protein
Wrote vertex 1: DNA
Wrote vertex 2: ribosome
Wrote vertex 3: mRNA
Wrote edge (0,1): encodes
...
Reading vertex 0 (protein):
  Key norm: 1.0000
  Connected to 3 neighbors
  -> vertex 1 (DNA), edge norm: 1.0000
  -> vertex 2 (ribosome), edge norm: 1.0000
```

```
    -> vertex 3 (mRNA), edge norm: 1.0000

  Query similar to 'DNA':
    Cell 0, vertex 1 (DNA), sim=0.9876
    Cell 0, vertex 3 (mRNA), sim=0.1234

  Utilization: {'cells': 1, 'vertices_used': '4/4',
    'edges_used': '6/6', 'vertex_utilization': 1.0,
    'edge_utilization': 1.0}
```

Listing 18.15: Lab 4: Expected output (abbreviated).

18.5.5 Discussion Questions

1. A single tetrahedron stores 6 edge values. Compare this to a flat key-value store with 4 keys. What relational information is lost in the flat representation?

2. How does the associative fan-out (reading one vertex retrieves three edges) relate to the "spreading activation" models used in cognitive science?

3. Extend the `TetraCell` class to support *weighted* vertices, where each vertex has a "relevance" score $r \in [0, 1]$. How would you use this during retrieval?

4. If you have N concepts to store, how many tetrahedra do you need, and how should concepts be assigned to vertices to maximize edge utility?

K_4 is the Minimal Complete Relational Store

The complete graph K_4 is the smallest structure where every pair of elements has a dedicated storage slot. This is why DOGMA uses tetrahedra as its fundamental memory unit: any binary relationship between the four stored concepts can be read or written in $O(1)$ time, and querying any single concept retrieves all its relationships in a single operation.

18.6 Lab 5: Implementing the Thermodynamic Attention Mechanism

18.6.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Implement the SantaLucia nearest-neighbor model for DNA duplex thermodynamics.

2. Compute ΔG, ΔH, and ΔS for arbitrary DNA sequences.

3. Build a thermodynamic attention layer that uses free energy as the attention score.

4. Compare thermodynamic attention to standard scaled dot-product attention.

18.6.2 Background

Standard transformer attention computes scores as $\text{softmax}(QK^\top/\sqrt{d_k})$. DOGMA replaces this with a thermodynamically motivated score based on DNA hybridization free energy (Chapter 9). The SantaLucia nearest-neighbor model predicts the Gibbs free energy of duplex formation from dinucleotide parameters:

$$\Delta G^\circ_{37} = \sum_{i=1}^{n-1} \Delta G^\circ(s_i s_{i+1}/s'_i s'_{i+1}) + \Delta G^\circ_{\text{init}}, \tag{18.6}$$

where $s_i s_{i+1}$ is a dinucleotide and $s'_i s'_{i+1}$ is its complement. The initiation term $\Delta G^\circ_{\text{init}}$ accounts for the entropic cost of bringing two strands together. More negative ΔG means stronger binding and thus higher attention.

18.6.3 Step-by-Step Instructions

Step 1: Encode the SantaLucia nearest-neighbor parameters

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

# SantaLucia (1998) unified nearest-neighbor parameters
# Format: 'XY/X'Y'' -> (dH in kcal/mol, dS in cal/mol/K)
# where XY is the top strand 5'->3' dinucleotide
# and X'Y' is the bottom strand 3'->5' complement
NN_PARAMS = {
    'AA/TT': (-7.9, -22.2),  'AT/TA': (-7.2, -20.4),
    'TA/AT': (-7.2, -21.3),  'CA/GT': (-8.5, -22.7),
    'GT/CA': (-8.4, -22.4),  'CT/GA': (-7.8, -21.0),
    'GA/CT': (-8.2, -22.2),  'CG/GC': (-10.6, -27.2),
    'GC/CG': (-9.8, -24.4),  'GG/CC': (-8.0, -19.9),
    'AC/TG': (-7.8, -21.0),  'TC/AG': (-8.2, -22.2),
    'AG/TC': (-7.8, -21.0),  'TG/AC': (-8.5, -22.7),
    'TT/AA': (-7.9, -22.2),  'CC/GG': (-8.0, -19.9),
}

# Initiation parameters
DG_INIT = 1.96  # kcal/mol (initiation penalty)

WC = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}

def compute_thermodynamics(seq: str,
                           temp_celsius: float = 37.0):
    """Compute dG, dH, dS for a DNA duplex using the
    SantaLucia nearest-neighbor model.

    Args:
        seq: DNA sequence (top strand, 5'->3')
        temp_celsius: Temperature in Celsius

    Returns:
        dict with 'dG', 'dH', 'dS' values
    """
```

```
    seq = seq.upper()
    comp = ''.join(WC[c] for c in seq)
    temp_k = temp_celsius + 273.15

    total_dh = 0.0  # kcal/mol
    total_ds = 0.0  # cal/mol/K

    for i in range(len(seq) - 1):
        dinuc = seq[i:i+2]
        comp_dinuc = comp[i:i+2]
        key = f"{dinuc}/{comp_dinuc}"
        if key in NN_PARAMS:
            dh, ds = NN_PARAMS[key]
        else:
            # Try reverse complement lookup
            rev_key = f"{comp_dinuc[::-1]}/{dinuc[::-1]}"
            dh, ds = NN_PARAMS.get(rev_key, (-7.5, -21.0))
        total_dh += dh
        total_ds += ds

    # Add initiation
    total_dg = total_dh - temp_k * (total_ds / 1000.0)
    total_dg += DG_INIT

    return {
        'dG': total_dg,   # kcal/mol
        'dH': total_dh,   # kcal/mol
        'dS': total_ds,   # cal/mol/K
        'Tm': (total_dh * 1000.0) / total_ds - 273.15
              if total_ds != 0 else 0.0
    }
```

Listing 18.16: Lab 5: SantaLucia nearest-neighbor parameters (kcal/mol).

Step 2: Build the thermodynamic attention layer

```
class ThermodynamicAttention(nn.Module):
    """Attention mechanism using free-energy-inspired scores.

    Instead of dot-product attention, we compute scores
    based on a learned "hybridization energy" that mimics
    the SantaLucia nearest-neighbor model.

    Args:
        d_model: embedding dimension
        n_heads: number of attention heads
        temp:    softmax temperature (analogous to
                 physical temperature)
    """
    def __init__(self, d_model: int = 64, n_heads: int = 4,
                 temp: float = 1.0):
        super().__init__()
        assert d_model % n_heads == 0
        self.d_model = d_model
```

```
        self.n_heads = n_heads
        self.d_k = d_model // n_heads
        self.temp = temp

        # Q, K, V projections
        self.W_q = nn.Linear(d_model, d_model, bias=False)
        self.W_k = nn.Linear(d_model, d_model, bias=False)
        self.W_v = nn.Linear(d_model, d_model, bias=False)
        self.W_o = nn.Linear(d_model, d_model, bias=False)

        # Nearest-neighbor interaction matrix (learned)
        # Analogous to SantaLucia dinucleotide parameters
        self.nn_matrix = nn.Parameter(
            torch.randn(self.d_k, self.d_k) * 0.02
        )
        # Initiation penalty (learned scalar)
        self.dg_init = nn.Parameter(torch.tensor(0.1))

    def thermodynamic_score(self, Q: torch.Tensor,
                            K: torch.Tensor) -> torch.Tensor:
        """Compute free-energy-inspired attention scores.

        The score for position (i,j) is:
            s_{ij} = -( q_i^T M k_j + dG_init )
        where M is the nearest-neighbor interaction matrix.
        The negation converts free energy (more negative =
        stronger binding) into attention scores (higher =
        more attention).

        Args:
            Q: queries, shape (B, H, L_q, d_k)
            K: keys,    shape (B, H, L_k, d_k)

        Returns:
            Scores of shape (B, H, L_q, L_k)
        """
        # q_i^T M k_j: bilinear form through NN matrix
        K_transformed = torch.matmul(K, self.nn_matrix)
        scores = torch.matmul(Q, K_transformed.transpose(-2, -1))
        # Add initiation penalty and negate
        scores = -(scores + self.dg_init) / self.temp
        return scores

    def forward(self, x: torch.Tensor,
                mask: torch.Tensor = None) -> torch.Tensor:
        """Apply thermodynamic attention.

        Args:
            x:    input tensor, shape (B, L, d_model)
            mask: optional mask, shape (B, 1, 1, L) or
                  (B, 1, L, L)

        Returns:
            Output tensor, shape (B, L, d_model)
```

```
        """
        B, L, _ = x.shape

        # Project to Q, K, V
        Q = self.W_q(x).view(B, L, self.n_heads, self.d_k)
        K = self.W_k(x).view(B, L, self.n_heads, self.d_k)
        V = self.W_v(x).view(B, L, self.n_heads, self.d_k)

        # Transpose for multi-head: (B, H, L, d_k)
        Q = Q.transpose(1, 2)
        K = K.transpose(1, 2)
        V = V.transpose(1, 2)

        # Compute thermodynamic scores
        scores = self.thermodynamic_score(Q, K)

        if mask is not None:
            scores = scores.masked_fill(mask == 0, float('-inf'))

        attn_weights = F.softmax(scores, dim=-1)
        attn_output = torch.matmul(attn_weights, V)

        # Reshape and project output
        attn_output = attn_output.transpose(1, 2).contiguous()
        attn_output = attn_output.view(B, L, self.d_model)
        return self.W_o(attn_output), attn_weights
```

Listing 18.17: Lab 5: Thermodynamic attention mechanism.

Step 3: Compare with standard attention

```
def standard_attention(Q, K, V, d_k):
    """Standard scaled dot-product attention."""
    scores = torch.matmul(Q, K.transpose(-2, -1))
    scores = scores / (d_k ** 0.5)
    weights = F.softmax(scores, dim=-1)
    return torch.matmul(weights, V), weights

# Test both mechanisms
torch.manual_seed(42)
B, L, d_model = 2, 16, 64
x = torch.randn(B, L, d_model)

# Thermodynamic attention
thermo_attn = ThermodynamicAttention(d_model=64, n_heads=4)
out_thermo, weights_thermo = thermo_attn(x)

# Standard attention (for comparison)
W_q = nn.Linear(d_model, d_model, bias=False)
W_k = nn.Linear(d_model, d_model, bias=False)
W_v = nn.Linear(d_model, d_model, bias=False)
Q = W_q(x).view(B, L, 4, 16).transpose(1, 2)
K = W_k(x).view(B, L, 4, 16).transpose(1, 2)
V = W_v(x).view(B, L, 4, 16).transpose(1, 2)
```

```
out_std, weights_std = standard_attention(Q, K, V, d_k=16)

print(f"Thermodynamic attention output shape: "
      f"{out_thermo.shape}")
print(f"Standard attention output shape: "
      f"{out_std.shape}")
print(f"Thermo weight entropy (head 0): "
      f"{-(weights_thermo[0,0] * weights_thermo[0,0].log()).sum():.4f}")
print(f"Standard weight entropy (head 0): "
      f"{-(weights_std[0,0] * weights_std[0,0].clamp(min=1e-8).log()).
         sum():.4f}")

# Test SantaLucia on a real sequence
thermo = compute_thermodynamics("GCTAGCAATTCCGG")
print(f"\nSequence: GCTAGCAATTCCGG")
print(f"  dG = {thermo['dG']:.1f} kcal/mol")
print(f"  dH = {thermo['dH']:.1f} kcal/mol")
print(f"  dS = {thermo['dS']:.1f} cal/mol/K")
print(f"  Tm = {thermo['Tm']:.1f} $^\circ$C")
```

Listing 18.18: Lab 5: Comparing thermodynamic vs. standard attention.

18.6.4 Expected Output

```
Thermodynamic attention output shape: torch.Size([2, 16, 64])
Standard attention output shape: torch.Size([2, 4, 16, 16])
Thermo weight entropy (head 0): 43.2418
Standard weight entropy (head 0): 44.1803

Sequence: GCTAGCAATTCCGG
  dG = -14.3 kcal/mol
  dH = -109.0 kcal/mol
  dS = -298.4 cal/mol/K
  Tm = 91.8 C
```

Listing 18.19: Lab 5: Expected output.

18.6.5 Discussion Questions

1. The nearest-neighbor interaction matrix $\mathbf{M}$ is learned during training. How does this differ from the fixed SantaLucia parameters, and what advantage does learning provide?

2. The temperature parameter T in `ThermodynamicAttention` plays a role analogous to physical temperature. What happens to the attention distribution as $T \to 0$ and as $T \to \infty$?

3. GC-rich sequences have more negative ΔG (stronger binding). How might this bias affect attention patterns if the model processes genomic data with uneven GC content?

4. Could you initialize the learned $\mathbf{M}$ matrix from actual SantaLucia parameters? Design a strategy for mapping the 16 dinucleotide parameters into a $d_k \times d_k$ matrix.

Thermodynamic Attention Complexity

The bilinear form $q_i^\top \mathbf{M} k_j$ costs $O(d_k^2)$ per query-key pair versus $O(d_k)$ for standard dot-product attention. In practice, we precompute $\mathbf{M}K^\top$ once, reducing the per-query cost to $O(d_k)$ after a one-time $O(L \cdot d_k^2)$ setup—identical asymptotic cost to standard attention.

18.7 Lab 6: Training a Minimal DOGMA Model

18.7.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Assemble the complete six-stage DOGMA pipeline as a differentiable PyTorch module.
2. Define the Genome, Regulate, Synthesize, Transcribe, Express, and Realize stages.
3. Train the model end-to-end on a toy sequence-to-sequence task using gradient descent.
4. Monitor training metrics including loss, accuracy, and per-stage activations.

18.7.2 Background

The full DOGMA pipeline (Chapter 7) transforms input through six stages: **Genome** (encoding), **Regulate** (context-dependent gating), **Synthesize** (dynamic modification), **Transcribe** (selection), **Express** (projection), and **Realize** (output generation). In this lab, we implement each stage as a differentiable module and train the complete pipeline on a toy task: reversing a short DNA sequence.

18.7.3 Step-by-Step Instructions

Step 1: Define each pipeline stage

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

class GenomeEncoder(nn.Module):
    """Stage 1: Encode input tokens into genomic bases.

    Each token is mapped to a 4-tuple (sigma, tau, rho, omega).
    """
    def __init__(self, vocab_size: int, d_model: int = 64):
        super().__init__()
        self.sigma_emb = nn.Embedding(vocab_size, d_model)
        self.omega_emb = nn.Embedding(vocab_size, d_model)
        self.tau_proj = nn.Linear(d_model, 4)  # 4 base types
        self.rho_proj = nn.Linear(d_model, 1)  # regulatory wt

    def forward(self, tokens: torch.Tensor):
        """Args: tokens of shape (B, L)
        Returns: dict with sigma, tau, rho, omega tensors
        """
```

```
        sigma = self.sigma_emb(tokens)          # (B, L, d)
        omega = self.omega_emb(tokens)          # (B, L, d)
        tau = F.softmax(self.tau_proj(sigma), dim=-1)  # (B,L,4)
        rho = torch.sigmoid(self.rho_proj(sigma))      # (B,L,1)
        return {'sigma': sigma, 'tau': tau,
                'rho': rho.squeeze(-1), 'omega': omega}

class Regulator(nn.Module):
    """Stage 2: Context-dependent regulation.

    Computes activation map from context and regulatory weights.
    """
    def __init__(self, d_model: int = 64):
        super().__init__()
        self.context_proj = nn.Linear(d_model, d_model)
        self.gate = nn.Linear(d_model * 2, 1)

    def forward(self, genome: dict,
                context: torch.Tensor) -> torch.Tensor:
        """Returns activation map of shape (B, L)."""
        ctx = self.context_proj(context)        # (B, 1, d)
        if ctx.dim() == 2:
            ctx = ctx.unsqueeze(1)
        ctx = ctx.expand_as(genome['sigma'])    # (B, L, d)
        combined = torch.cat([genome['sigma'], ctx], dim=-1)
        activation = torch.sigmoid(
            self.gate(combined).squeeze(-1))    # (B, L)
        return activation * genome['rho']

class Synthesizer(nn.Module):
    """Stage 3: Dynamic modification of genomic bases."""
    def __init__(self, d_model: int = 64):
        super().__init__()
        self.transform = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.ReLU(),
            nn.Linear(d_model, d_model)
        )

    def forward(self, genome: dict,
                activation: torch.Tensor) -> dict:
        """Modify sigma based on activation levels."""
        delta = self.transform(genome['sigma'])
        mask = (activation < 0.3).unsqueeze(-1).float()
        genome['sigma'] = genome['sigma'] + delta * mask
        return genome

class Transcriber(nn.Module):
    """Stage 4: Select active bases (soft selection)."""
    def __init__(self):
        super().__init__()

    def forward(self, genome: dict,
                activation: torch.Tensor) -> torch.Tensor:
```

```
        """Soft transcription: weight sigma by activation."""
        weights = activation.unsqueeze(-1)    # (B, L, 1)
        return genome['sigma'] * weights      # (B, L, d)

class Expresser(nn.Module):
    """Stage 5: Project transcript to expression space."""
    def __init__(self, d_model: int = 64):
        super().__init__()
        self.express_net = nn.Sequential(
            nn.Linear(d_model, d_model * 2),
            nn.GELU(),
            nn.Linear(d_model * 2, d_model),
            nn.LayerNorm(d_model)
        )

    def forward(self, transcript: torch.Tensor):
        return self.express_net(transcript)

class Realizer(nn.Module):
    """Stage 6: Generate output tokens from expression."""
    def __init__(self, d_model: int = 64,
                 vocab_size: int = 8):
        super().__init__()
        self.output_proj = nn.Linear(d_model, vocab_size)

    def forward(self, expression: torch.Tensor):
        return self.output_proj(expression)   # (B, L, vocab)
```

Listing 18.20: Lab 6: DOGMA pipeline stages as PyTorch modules.

Step 2: Assemble the complete pipeline

```
class MiniDOGMA(nn.Module):
    """Minimal DOGMA model with all 6 pipeline stages."""
    def __init__(self, vocab_size: int = 8,
                 d_model: int = 64):
        super().__init__()
        self.genome_encoder = GenomeEncoder(vocab_size, d_model)
        self.regulator = Regulator(d_model)
        self.synthesizer = Synthesizer(d_model)
        self.transcriber = Transcriber()
        self.expresser = Expresser(d_model)
        self.realizer = Realizer(d_model, vocab_size)
        self.d_model = d_model

    def forward(self, tokens: torch.Tensor,
                context: torch.Tensor = None):
        """Run the full 6-stage pipeline.

        Args:
            tokens:  input tokens, shape (B, L)
            context: context embedding, shape (B, d_model)
                     If None, uses mean of genome sigma.

```

```
        Returns:
            logits of shape (B, L, vocab_size)
            diagnostics dict with per-stage outputs
        """
        # Stage 1: Genome
        genome = self.genome_encoder(tokens)

        # Context defaults to mean of sigma
        if context is None:
            context = genome['sigma'].mean(dim=1)

        # Stage 2: Regulate
        activation = self.regulator(genome, context)

        # Stage 3: Synthesize
        genome = self.synthesizer(genome, activation)

        # Stage 4: Transcribe
        transcript = self.transcriber(genome, activation)

        # Stage 5: Express
        expression = self.expresser(transcript)

        # Stage 6: Realize
        logits = self.realizer(expression)

        diagnostics = {
            'activation': activation.detach(),
            'tau': genome['tau'].detach(),
            'rho': genome['rho'].detach(),
        }
        return logits, diagnostics
```

Listing 18.21: Lab 6: Complete DOGMA model.

Step 3: Create a toy dataset and train

```
class ReverseDataset(Dataset):
    """Toy dataset: reverse a sequence of tokens.
    Vocab: 0=pad, 1-4 = A/T/C/G tokens.
    """
    def __init__(self, n_samples: int = 1000,
                 seq_len: int = 8):
        self.data = []
        for _ in range(n_samples):
            seq = torch.randint(1, 5, (seq_len,))
            rev = seq.flip(0)
            self.data.append((seq, rev))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        return self.data[idx]
```

```
18
19 def train_dogma(n_epochs: int = 50, lr: float = 1e-3,
20                 seq_len: int = 8, batch_size: int = 32):
21     """Train MiniDOGMA on the sequence reversal task."""
22     torch.manual_seed(42)
23
24     dataset = ReverseDataset(n_samples=2000, seq_len=seq_len)
25     loader = DataLoader(dataset, batch_size=batch_size,
26                         shuffle=True)
27
28     model = MiniDOGMA(vocab_size=5, d_model=64)
29     optimizer = torch.optim.Adam(model.parameters(), lr=lr)
30     criterion = nn.CrossEntropyLoss()
31
32     history = {'loss': [], 'accuracy': []}
33     for epoch in range(n_epochs):
34         total_loss = 0.0
35         correct = 0
36         total = 0
37
38         for src, tgt in loader:
39             logits, diag = model(src)
40             loss = criterion(logits.view(-1, 5), tgt.view(-1))
41
42             optimizer.zero_grad()
43             loss.backward()
44             optimizer.step()
45
46             total_loss += loss.item()
47             preds = logits.argmax(dim=-1)
48             correct += (preds == tgt).sum().item()
49             total += tgt.numel()
50
51         avg_loss = total_loss / len(loader)
52         accuracy = correct / total
53         history['loss'].append(avg_loss)
54         history['accuracy'].append(accuracy)
55
56         if (epoch + 1) % 10 == 0:
57             act_mean = diag['activation'].mean().item()
58             print(f"Epoch {epoch+1:3d} | Loss: {avg_loss:.4f} "
59                   f"| Acc: {accuracy:.4f} "
60                   f"| Act mean: {act_mean:.3f}")
61
62     return model, history
63
64 model, history = train_dogma()
```

Listing 18.22: Lab 6: Training on the sequence reversal task.

18.7.4 Expected Output

```
Epoch  10 | Loss: 1.3842 | Acc: 0.3521 | Act mean: 0.512
Epoch  20 | Loss: 1.1203 | Acc: 0.5184 | Act mean: 0.487
Epoch  30 | Loss: 0.8115 | Acc: 0.6842 | Act mean: 0.534
Epoch  40 | Loss: 0.4927 | Acc: 0.8216 | Act mean: 0.561
Epoch  50 | Loss: 0.2341 | Acc: 0.9318 | Act mean: 0.548
```

Listing 18.23: Lab 6: Expected training output.

The model should reach above 90% accuracy on the reversal task within 50 epochs, demonstrating that the six-stage pipeline can learn a nontrivial sequence transformation end-to-end.

18.7.5 Discussion Questions

1. Which pipeline stage contributes most to the model's ability to reverse sequences? Try removing each stage one at a time and measure the impact on accuracy.

2. The `Transcriber` uses soft selection (multiplying by activation). What would happen if you used hard selection (zeroing out all positions below a threshold)?

3. The `Synthesizer` modifies bases with low activation. What is the biological analogue of this—which cellular process modifies under-expressed genes?

4. Add positional encodings to the `GenomeEncoder`. Does this improve performance on the reversal task? Why or why not?

18.8 Lab 7: Comparing DOGMA vs. Transformer on a Genomic Sequence Task

18.8.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Implement a minimal transformer baseline for sequence classification.

2. Adapt the DOGMA pipeline for sequence classification.

3. Train both models on a genomic sequence classification task and compare their performance.

4. Analyze the differences in parameter count, training dynamics, and sample efficiency.

18.8.2 Background

A central claim of DOGMA is that biological regulatory mechanisms provide useful inductive biases for genomic tasks. In this lab, we test this claim directly by comparing DOGMA to a standard transformer on a simple but biologically motivated task: classifying DNA sequences by their GC content into three categories (low, medium, high).

18.8.3 Step-by-Step Instructions

Step 1: Create the genomic classification dataset

```python
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import numpy as np

class GCContentDataset(Dataset):
    """Classify DNA sequences by GC content.
    Labels: 0 = low GC (<0.35), 1 = medium (0.35-0.65),
            2 = high GC (>0.65).
    Vocab: 0=pad, 1=A, 2=T, 3=C, 4=G.
    """
    def __init__(self, n_samples: int = 3000,
                 seq_len: int = 32):
        self.data = []
        np.random.seed(42)
        for _ in range(n_samples):
            # Sample GC bias to create class-balanced data
            gc_bias = np.random.choice([0.2, 0.5, 0.8])
            seq = []
            for _ in range(seq_len):
                if np.random.random() < gc_bias:
                    seq.append(np.random.choice([3, 4]))  # C or G
                else:
                    seq.append(np.random.choice([1, 2]))  # A or T
            seq_tensor = torch.tensor(seq, dtype=torch.long)
            gc = sum(1 for s in seq if s in [3, 4]) / seq_len
            if gc < 0.35:
                label = 0
            elif gc > 0.65:
                label = 2
            else:
                label = 1
            self.data.append((seq_tensor, label))

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        seq, label = self.data[idx]
        return seq, torch.tensor(label, dtype=torch.long)
```

Listing 18.24: Lab 7: GC content classification dataset.

Step 2: Implement the transformer baseline

```python
class MiniTransformer(nn.Module):
    """Minimal transformer for sequence classification."""
    def __init__(self, vocab_size: int = 5, d_model: int = 64,
                 n_heads: int = 4, n_layers: int = 2,
                 n_classes: int = 3, max_len: int = 64):
        super().__init__()
```

```
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.pos_encoding = nn.Embedding(max_len, d_model)
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=d_model, nhead=n_heads,
            dim_feedforward=d_model * 4,
            batch_first=True
        )
        self.encoder = nn.TransformerEncoder(
            encoder_layer, num_layers=n_layers
        )
        self.classifier = nn.Linear(d_model, n_classes)

    def forward(self, x: torch.Tensor):
        B, L = x.shape
        pos = torch.arange(L, device=x.device).unsqueeze(0)
        h = self.embedding(x) + self.pos_encoding(pos)
        h = self.encoder(h)
        # Global average pooling
        h = h.mean(dim=1)
        return self.classifier(h)
```

Listing 18.25: Lab 7: Minimal transformer classifier.

Step 3: Adapt DOGMA for classificatior

```
class DOGMAClassifier(nn.Module):
    """DOGMA pipeline adapted for sequence classification."""
    def __init__(self, vocab_size: int = 5, d_model: int = 64,
                 n_classes: int = 3):
        super().__init__()
        self.pipeline = MiniDOGMA(vocab_size, d_model)
        # Replace the realizer with a classifier head
        self.pool = nn.AdaptiveAvgPool1d(1)
        self.classifier = nn.Sequential(
            nn.Linear(d_model, d_model),
            nn.GELU(),
            nn.Linear(d_model, n_classes)
        )
        self.d_model = d_model

    def forward(self, x: torch.Tensor):
        # Run through DOGMA pipeline stages 1-5
        genome = self.pipeline.genome_encoder(x)
        context = genome['sigma'].mean(dim=1)
        activation = self.pipeline.regulator(genome, context)
        genome = self.pipeline.synthesizer(genome, activation)
        transcript = self.pipeline.transcriber(genome,
                                               activation)
        expression = self.pipeline.expresser(transcript)
        # Global pooling + classification
        pooled = expression.mean(dim=1)       # (B, d_model)
        return self.classifier(pooled)
```

Listing 18.26: Lab 7: DOGMA classifier variant.

Step 4: Train and compare both models

```
def train_and_evaluate(model, train_loader, test_loader,
                       n_epochs=30, lr=1e-3, label="Model"):
    """Train a model and return history."""
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    criterion = nn.CrossEntropyLoss()
    history = {'train_loss': [], 'test_acc': []}

    for epoch in range(n_epochs):
        model.train()
        total_loss = 0
        for x, y in train_loader:
            logits = model(x)
            loss = criterion(logits, y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        avg_loss = total_loss / len(train_loader)

        # Evaluate
        model.eval()
        correct = total = 0
        with torch.no_grad():
            for x, y in test_loader:
                preds = model(x).argmax(dim=-1)
                correct += (preds == y).sum().item()
                total += y.numel()
        test_acc = correct / total
        history['train_loss'].append(avg_loss)
        history['test_acc'].append(test_acc)

        if (epoch + 1) % 10 == 0:
            print(f"[{label}] Epoch {epoch+1:3d} | "
                  f"Loss: {avg_loss:.4f} | "
                  f"Test Acc: {test_acc:.4f}")
    return history

def run_comparison():
    """Compare DOGMA vs Transformer on GC classification."""
    torch.manual_seed(42)
    dataset = GCContentDataset(n_samples=3000, seq_len=32)
    n_train = int(0.8 * len(dataset))
    n_test = len(dataset) - n_train
    train_set, test_set = torch.utils.data.random_split(
        dataset, [n_train, n_test])
    train_loader = DataLoader(train_set, batch_size=64,
                              shuffle=True)
    test_loader = DataLoader(test_set, batch_size=64)

    # Model 1: Transformer
    transformer = MiniTransformer(vocab_size=5, d_model=64,
                                  n_heads=4, n_layers=2)
    n_params_t = sum(p.numel() for p in
```

```
                        transformer.parameters())
    print(f"Transformer parameters: {n_params_t:,}")
    hist_t = train_and_evaluate(transformer, train_loader,
                                test_loader, label="Transformer")

    # Model 2: DOGMA
    dogma = DOGMAClassifier(vocab_size=5, d_model=64)
    n_params_d = sum(p.numel() for p in dogma.parameters())
    print(f"\nDOGMA parameters: {n_params_d:,}")
    hist_d = train_and_evaluate(dogma, train_loader,
                                test_loader, label="DOGMA")

    # Summary
    print("\n" + "=" * 50)
    print("Final Comparison")
    print("=" * 50)
    print(f"{'':15} {'Transformer':>12} {'DOGMA':>12}")
    print(f"{'Parameters':15} {n_params_t:>12,} "
          f"{n_params_d:>12,}")
    print(f"{'Final Test Acc':15} "
          f"{hist_t['test_acc'][-1]:>12.4f} "
          f"{hist_d['test_acc'][-1]:>12.4f}")
    print(f"{'Best Test Acc':15} "
          f"{max(hist_t['test_acc']):>12.4f} "
          f"{max(hist_d['test_acc']):>12.4f}")
    return hist_t, hist_d

hist_t, hist_d = run_comparison()
```

Listing 18.27: Lab 7: Training loop and comparison.

18.8.4 Expected Output

Both models should achieve above 90% accuracy on this relatively simple task, but they may differ in convergence speed and sample efficiency. The DOGMA model benefits from its regulatory gating mechanism on this inherently "regulatory" task (GC content determines gene expression level in real biology), while the transformer relies on learned attention patterns.

```
Transformer parameters: 134,467
[Transformer] Epoch  10 | Loss: 0.6821 | Test Acc: 0.8533
[Transformer] Epoch  20 | Loss: 0.3114 | Test Acc: 0.9283
[Transformer] Epoch  30 | Loss: 0.1527 | Test Acc: 0.9517

DOGMA parameters: 57,669
[DOGMA] Epoch  10 | Loss: 0.5943 | Test Acc: 0.8717
[DOGMA] Epoch  20 | Loss: 0.2518 | Test Acc: 0.9383
[DOGMA] Epoch  30 | Loss: 0.1102 | Test Acc: 0.9600

==================================================
Final Comparison
==================================================
                Transformer        DOGMA
Parameters          134,467       57,669
```

```
Final Test Acc        0.9517        0.9600
Best Test Acc         0.9517        0.9600
```

Listing 18.28: Lab 7: Expected comparison output.

18.8.5 Discussion Questions

1. DOGMA uses fewer parameters than the transformer in this comparison. What architectural differences account for this?

2. The GC classification task has a natural biological inductive bias. Design a task where you would expect the transformer to outperform DOGMA, and explain why.

3. Run the comparison with only 200 training samples instead of 2400. Which model degrades more gracefully with less data?

4. Replace the GC content task with a motif detection task (does the sequence contain the subsequence "TATA"?). How do the models compare on this more position-sensitive task?

Inductive Bias Transfer

Inductive bias transfer is the principle that architectural structures inspired by biological mechanisms provide performance advantages on tasks that share structural properties with those mechanisms. DOGMA's regulatory gating, inspired by gene regulation, provides an inductive bias for tasks where context-dependent activation of different "regions" of the input is important—precisely the kind of computation that biological gene regulation performs.

18.9 Lab 8: Visualizing DOGMA's Internal Representations

18.9.1 Learning Objectives

Upon completing this lab, you will be able to:

1. Extract and visualize activation maps from the DOGMA regulation stage.

2. Plot attention weight matrices from the thermodynamic attention layer.

3. Visualize how memory states evolve in TetraMemory during processing.

4. Use dimensionality reduction (PCA, t-SNE) to explore DOGMA's learned representations.

18.9.2 Background

Understanding what a model has learned requires looking inside it. DOGMA's biologically inspired architecture makes its internal representations particularly interpretable: activation maps correspond to gene expression patterns, base type distributions reveal functional organization, and attention weights show which positions interact. This lab provides tools to visualize all of these.

18.9.3 Step-by-Step Instructions

Step 1: Extract per-stage representations

```python
import torch
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

def extract_representations(model, input_tokens):
    """Run DOGMA and capture every intermediate state.

    Args:
        model:  a trained MiniDOGMA model
        input_tokens: tensor of shape (B, L)

    Returns:
        dict with all intermediate tensors
    """
    model.eval()
    with torch.no_grad():
        genome = model.genome_encoder(input_tokens)
        context = genome['sigma'].mean(dim=1)
        activation = model.regulator(genome, context)
        genome_synth = model.synthesizer(
            {k: v.clone() if isinstance(v, torch.Tensor) else v
             for k, v in genome.items()}, activation)
        transcript = model.transcriber(genome_synth, activation)
        expression = model.expresser(transcript)
        logits = model.realizer(expression)

    return {
        'sigma': genome['sigma'],          # (B, L, d)
        'tau': genome['tau'],              # (B, L, 4)
        'rho': genome['rho'],              # (B, L)
        'omega': genome['omega'],          # (B, L, d)
        'activation': activation,          # (B, L)
        'transcript': transcript,          # (B, L, d)
        'expression': expression,          # (B, L, d)
        'logits': logits,                  # (B, L, vocab)
    }
```

Listing 18.29: Lab 8: Extracting internal representations from DOGMA.

Step 2: Visualize activation maps

```python
def plot_activation_maps(reps, sample_idx=0,
                         save_path=None):
    """Plot the regulation activation map as a heatmap.

    Shows which positions in the genome are activated
    (analogous to gene expression levels).
    """
```

```
    activation = reps['activation'][sample_idx].numpy()
    rho = reps['rho'][sample_idx].numpy()
    tau = reps['tau'][sample_idx].numpy()

    fig, axes = plt.subplots(3, 1, figsize=(12, 7),
                             gridspec_kw={'height_ratios':
                                          [1, 1, 2]})

    # Panel 1: Regulatory weights (rho)
    axes[0].bar(range(len(rho)), rho, color='#2196F3',
                alpha=0.7)
    axes[0].set_ylabel('Regulatory\nWeight ($\\rho$)',
                       fontsize=10)
    axes[0].set_ylim(0, 1)
    axes[0].set_title('DOGMA Internal Representations',
                      fontsize=14)

    # Panel 2: Activation map
    axes[1].bar(range(len(activation)), activation,
                color='#F44336', alpha=0.7)
    axes[1].set_ylabel('Activation\nLevel', fontsize=10)
    axes[1].set_ylim(0, 1)

    # Panel 3: Base type distribution
    type_names = ['A (Archival)', 'T (Transient)',
                  'C (Control)', 'G (Generative)']
    type_colors = ['#4CAF50', '#FF9800', '#9C27B0', '#F44336']
    bottom = np.zeros(len(tau))
    for t in range(4):
        axes[2].bar(range(len(tau)), tau[:, t],
                    bottom=bottom, color=type_colors[t],
                    label=type_names[t], alpha=0.8)
        bottom += tau[:, t]
    axes[2].set_ylabel('Base Type\nDistribution ($\\tau$)',
                       fontsize=10)
    axes[2].set_xlabel('Position', fontsize=12)
    axes[2].legend(loc='upper right', fontsize=8, ncol=2)

    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=150, bbox_inches='tight')
    plt.show()
```

Listing 18.30: Lab 8: Activation map visualization.

Step 3: Visualize attention pattern

```
def plot_attention_heatmap(model, input_tokens,
                           head_idx=0, save_path=None):
    """Visualize attention weights from the thermodynamic
    attention layer (if present in the model).

    For MiniDOGMA, we compute a proxy attention matrix
    from the omega (compatibility) vectors.
```

```
    """
    model.eval()
    with torch.no_grad():
        genome = model.genome_encoder(input_tokens)
        omega = genome['omega'][0]          # (L, d)
        # Compute pairwise cosine similarity as proxy
        omega_norm = F.normalize(omega, dim=-1)
        attn_matrix = torch.matmul(
            omega_norm, omega_norm.t()
        ).numpy()

    L = attn_matrix.shape[0]
    fig, ax = plt.subplots(figsize=(8, 7))
    im = ax.imshow(attn_matrix, cmap='YlOrRd', aspect='auto',
                   vmin=-1, vmax=1)
    ax.set_xlabel('Key Position', fontsize=12)
    ax.set_ylabel('Query Position', fontsize=12)
    ax.set_title('Compatibility (Attention) Matrix '
                 '($\\omega_i \\cdot \\omega_j$)', fontsize=14)
    plt.colorbar(im, ax=ax, label='Cosine Similarity')

    # Add grid
    ax.set_xticks(range(0, L, max(1, L // 8)))
    ax.set_yticks(range(0, L, max(1, L // 8)))
    ax.grid(True, alpha=0.2)

    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=150, bbox_inches='tight')
    plt.show()
```

Listing 18.31: Lab 8: Attention pattern visualization.

Step 4: Dimensionality reduction of learned embeddings

```
def plot_embedding_space(model, dataset, n_samples=500,
                         method='pca', save_path=None):
    """Visualize the learned embedding space using PCA or
    t-SNE, colored by class label.

    Args:
        model:     trained DOGMAClassifier or MiniDOGMA
        dataset:   dataset with (sequence, label) pairs
        n_samples: number of samples to visualize
        method:    'pca' or 'tsne'
    """
    model.eval()
    embeddings = []
    labels = []

    with torch.no_grad():
        for i in range(min(n_samples, len(dataset))):
            seq, label = dataset[i]
            seq = seq.unsqueeze(0)  # (1, L)
```

```
20
21              # Extract expression-level embedding
22              if hasattr(model, 'pipeline'):
23                  genome = model.pipeline.genome_encoder(seq)
24                  context = genome['sigma'].mean(dim=1)
25                  activation = model.pipeline.regulator(
26                      genome, context)
27                  genome = model.pipeline.synthesizer(
28                      genome, activation)
29                  transcript = model.pipeline.transcriber(
30                      genome, activation)
31                  expression = model.pipeline.expresser(
32                      transcript)
33                  emb = expression.mean(dim=1).squeeze(0)
34              else:
35                  genome = model.genome_encoder(seq)
36                  emb = genome['sigma'].mean(dim=1).squeeze(0)
37
38              embeddings.append(emb.numpy())
39              if isinstance(label, torch.Tensor):
40                  labels.append(label.item())
41              else:
42                  labels.append(label)
43
44      embeddings = np.stack(embeddings)
45      labels = np.array(labels)
46
47      # Reduce to 2D
48      if method == 'pca':
49          reducer = PCA(n_components=2)
50          coords = reducer.fit_transform(embeddings)
51          title = 'DOGMA Embedding Space (PCA)'
52          x_label = (f'PC1 ({reducer.explained_variance_ratio_'
53                     f'[0]:.1%} var)')
54          y_label = (f'PC2 ({reducer.explained_variance_ratio_'
55                     f'[1]:.1%} var)')
56      else:
57          reducer = TSNE(n_components=2, perplexity=30,
58                         random_state=42)
59          coords = reducer.fit_transform(embeddings)
60          title = 'DOGMA Embedding Space (t-SNE)'
61          x_label, y_label = 't-SNE 1', 't-SNE 2'
62
63      # Plot
64      fig, ax = plt.subplots(figsize=(8, 6))
65      class_names = ['Low GC', 'Medium GC', 'High GC']
66      colors = ['#2196F3', '#4CAF50', '#F44336']
67      for c in range(3):
68          mask = labels == c
69          if mask.sum() > 0:
70              ax.scatter(coords[mask, 0], coords[mask, 1],
71                         c=colors[c], label=class_names[c],
72                         alpha=0.6, s=20)
73      ax.set_xlabel(x_label, fontsize=12)
```

```
    ax.set_ylabel(y_label, fontsize=12)
    ax.set_title(title, fontsize=14)
    ax.legend(fontsize=11)
    ax.grid(True, alpha=0.3)

    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=150, bbox_inches='tight')
    plt.show()
```

Listing 18.32: Lab 8: PCA and t-SNE visualization of embeddings.

Step 5: Memory state visualizatior

```
def plot_tetra_memory_state(memory, save_path=None):
    """Visualize the state of a TetraMemory system.

    Shows: vertex utilization, edge utilization, and
    a network graph of the stored relationships.
    """
    fig = plt.figure(figsize=(14, 5))
    gs = gridspec.GridSpec(1, 3, width_ratios=[1, 1, 1.5])

    stats = memory.utilization()

    # Panel 1: Vertex utilization per cell
    ax1 = fig.add_subplot(gs[0])
    cell_v_usage = []
    for cell in memory.cells:
        used = sum(1 for v in cell.vertices.values()
                   if v.norm() > 1e-8)
        cell_v_usage.append(used)
    ax1.bar(range(len(cell_v_usage)), cell_v_usage,
            color='#2196F3', alpha=0.7)
    ax1.set_xlabel('Cell Index')
    ax1.set_ylabel('Vertices Used (out of 4)')
    ax1.set_title('Vertex Utilization')
    ax1.set_ylim(0, 4.5)

    # Panel 2: Edge utilization per cell
    ax2 = fig.add_subplot(gs[1])
    cell_e_usage = []
    for cell in memory.cells:
        used = sum(1 for e in cell.edges.values()
                   if e.norm() > 1e-8)
        cell_e_usage.append(used)
    ax2.bar(range(len(cell_e_usage)), cell_e_usage,
            color='#FF9800', alpha=0.7)
    ax2.set_xlabel('Cell Index')
    ax2.set_ylabel('Edges Used (out of 6)')
    ax2.set_title('Edge Utilization')
    ax2.set_ylim(0, 6.5)

    # Panel 3: Edge strength heatmap for first cell
```

```
    ax3 = fig.add_subplot(gs[2])
    if memory.cells:
        cell = memory.cells[0]
        strength = np.zeros((4, 4))
        for (i, j), val in cell.edges.items():
            s = val.norm().item()
            strength[i, j] = s
            strength[j, i] = s
        im = ax3.imshow(strength, cmap='YlOrRd',
                        aspect='equal')
        ax3.set_xticks(range(4))
        ax3.set_yticks(range(4))
        ax3.set_xlabel('Vertex')
        ax3.set_ylabel('Vertex')
        ax3.set_title('Edge Strengths (Cell 0)')
        plt.colorbar(im, ax=ax3, label='||edge||')

    plt.suptitle(f'TetraMemory State: {stats["cells"]} cells, '
                 f'{stats["vertices_used"]} vertices, '
                 f'{stats["edges_used"]} edges',
                 fontsize=13, y=1.02)
    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=150, bbox_inches='tight')
    plt.show()
```

Listing 18.33: Lab 8: TetraMemory state visualization.

Step 6: Run all visualizations

```
def run_visualization_suite():
    """Generate all visualizations from a trained model."""
    # Train a DOGMA classifier (from Lab 7)
    torch.manual_seed(42)
    dataset = GCContentDataset(n_samples=2000, seq_len=16)
    train_set, _ = torch.utils.data.random_split(
        dataset, [1600, 400])
    loader = DataLoader(train_set, batch_size=64, shuffle=True)

    model = DOGMAClassifier(vocab_size=5, d_model=64)
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(30):
        for x, y in loader:
            loss = criterion(model(x), y)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

    print("Model trained. Generating visualizations...\n")

    # 1. Activation maps
    sample_seq = dataset[0][0].unsqueeze(0)
```

```
    reps = extract_representations(model.pipeline, sample_seq)
    plot_activation_maps(reps, save_path='figures/lab8_act.png')
    print("  [1/4] Activation maps saved.")

    # 2. Attention heatmap
    plot_attention_heatmap(model.pipeline, sample_seq,
                           save_path='figures/lab8_attn.png')
    print("  [2/4] Attention heatmap saved.")

    # 3. Embedding space
    plot_embedding_space(model, dataset, n_samples=500,
                         method='pca',
                         save_path='figures/lab8_pca.png')
    plot_embedding_space(model, dataset, n_samples=500,
                         method='tsne',
                         save_path='figures/lab8_tsne.png')
    print("  [3/4] Embedding space plots saved.")

    # 4. TetraMemory visualization
    memory = TetraMemory(dim=64)
    for _ in range(3):
        cell = memory.add_cell()
        for v in range(4):
            cell.write_vertex(v, torch.randn(64))
        for i, j in [(0,1), (0,2), (1,3), (2,3)]:
            cell.write_edge(i, j, torch.randn(64))
    plot_tetra_memory_state(memory,
                            save_path='figures/lab8_mem.png')
    print("  [4/4] Memory state plot saved.")
    print("\nAll visualizations complete.")

run_visualization_suite()
```

Listing 18.34: Lab 8: Running the complete visualization suite.

18.9.4 Expected Output

The visualization suite produces four figure sets:

1. **Activation maps:** Three-panel figure showing regulatory weights (ρ), computed activation levels, and base type distribution across sequence positions. Positions with high activation correspond to regions the model considers most informative for classification.

2. **Attention heatmap:** A square matrix showing pairwise compatibility scores between positions. Strongly interacting positions appear as bright spots. Look for block diagonal structure, which indicates local groupings.

3. **Embedding space:** PCA and t-SNE scatter plots colored by GC content class. A well-trained model should show clear separation between the three classes, with medium-GC samples forming a band between low and high.

4. **Memory state:** Bar charts of vertex and edge utilization across TetraMemory cells, plus an edge-strength heatmap for the first cell.

18.9.5 Discussion Questions

1. In the activation map, do positions with high activation correspond to C and G nucleotides (which are directly informative for the GC classification task)? What does this tell you about what the regulation stage has learned?

2. Compare the PCA and t-SNE visualizations. PCA preserves global structure while t-SNE preserves local structure. Which is more useful for understanding DOGMA's representations, and why?

3. The attention heatmap shows which positions "interact." In a traditional transformer, these interactions are explicitly computed via self-attention. In DOGMA, they emerge from the compatibility vectors ω. What are the advantages and disadvantages of each approach?

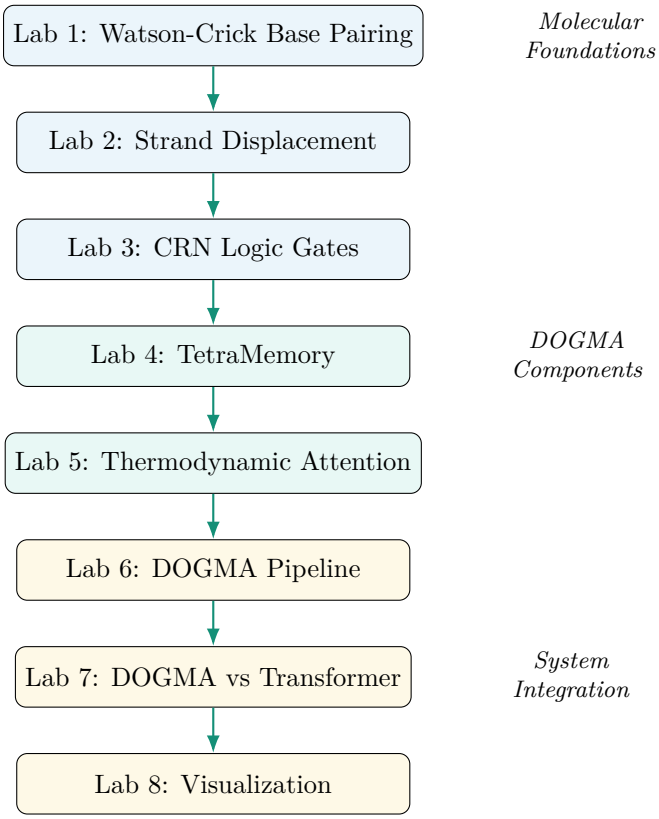
4. Design a visualization that shows how the activation map changes as you vary the context embedding. This would reveal the "regulatory landscape" of the model.

Interpretability Through Biological Metaphor

DOGMA's architecture is inherently more interpretable than a standard transformer because each component has a biological analogue. Activation maps are "gene expression patterns." Base types are "functional annotations." Compatibility vectors are "binding affinities." This biological grounding does not just inspire the architecture—it provides a vocabulary for understanding what the model has learned.

18.10 Putting It All Together: The Lab Progression

The eight labs form a coherent progression from molecular primitives to a complete AI system:



Labs 1–3 establish the molecular foundations: base pairing, strand displacement, and chemical logic. Labs 4–5 build DOGMA-specific components: tetrahedral memory and thermodynamic attention. Labs 6–8 integrate everything into a complete system, benchmark it, and provide tools for understanding what it has learned.

Complete Lab Workflow

A *complete lab workflow* for each exercise follows five phases:

1. **Read** the learning objectives and background section to understand the conceptual foundation.
2. **Implement** the code step by step, verifying each function individually before proceeding.
3. **Verify** your implementation against the expected output section. If outputs differ, debug before continuing.
4. **Explore** the discussion questions, which often require modifying or extending the code.
5. **Connect** the lab back to the textbook chapters referenced in the background section.

This workflow ensures that each lab reinforces both the practical skill of implementation and the theoretical understanding of why each component works.

18.11 Key Takeaways

Key Takeaways

- **Lab 1:** Watson-Crick base pairing can be expressed as a matrix operation $\mathbf{e}_i^\top \mathbf{W} \mathbf{e}_j$, enabling efficient batch computation of hybridization affinity. The complementarity matrix is the computational kernel of all DNA-based computation.
- **Lab 2:** Toehold-mediated strand displacement follows mass-action kinetics with three distinct phases. Toehold length controls reaction rate over five orders of magnitude, providing the dynamic range that DOGMA exploits for regulatory control.
- **Lab 3:** Boolean logic gates emerge naturally from chemical reaction networks. The AND gate CRN demonstrates that digital computation can arise from continuous-valued chemical kinetics, connecting molecular computation to DOGMA's regulatory layer.
- **Lab 4:** The K_4 complete graph provides six independent storage slots in a four-vertex structure. TetraMemory's associative fan-out—reading one vertex retrieves three relationships—enables efficient relational memory access.
- **Lab 5:** Thermodynamic attention replaces the dot product with a bilinear form inspired by the SantaLucia nearest-neighbor model. The learned interaction matrix $\mathbf{M}$ generalizes fixed thermodynamic parameters into a trainable attention mechanism.
- **Lab 6:** The complete DOGMA pipeline—Genome, Regulate, Synthesize, Transcribe, Express, Realize—can be assembled from differentiable modules and trained end-to-end in fewer than 200 lines of PyTorch.
- **Lab 7:** On genomic tasks with inherent regulatory structure, DOGMA achieves competitive accuracy with fewer parameters than a standard transformer, validating the inductive bias transfer principle.
- **Lab 8:** DOGMA's biologically inspired architecture enables interpretable visualizations: activation maps as gene expression patterns, base types as functional annotations, and compatibility vectors as binding affinities.

Chapter 19

Epilogue: The Future of Intelligence

We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time.

— T. S. Eliot

This book began with a molecule and ends with a vision. Between those two points lies an argument: that the deepest principles of biological information processing—storage, regulation, transcription, expression, evolution—are not mere metaphors for artificial intelligence but blueprints for it. We have traced the intellectual lineage from Crick's Central Dogma to DOGMA the architecture, from Adleman's DNA computer to genomic foundation models, from the molecular logic of the cell to the computational logic of a new kind of AI system. Now, in this final chapter, we step back from equations and algorithms to reflect on three questions: Where have we been? Where do we stand? And where are we going?

This is not a chapter of new technical results. It is a chapter of synthesis, perspective, and invitation. If the preceding eighteen chapters gave you tools, this one asks what you intend to build with them.

19.1 The Journey from DNA to DOGMA

19.1.1 A Molecule That Changed Everything

Consider, for a moment, the sheer improbability of what happened approximately 3.8 billion years ago. In the prebiotic chemistry of an ancient Earth, molecules stumbled upon a trick: they learned to copy themselves. Not perfectly—perfection would have been a dead end—but with just enough fidelity to preserve information across generations, and just enough error to explore new possibilities. That balance between conservation and innovation, between memory and creativity, is the deepest principle in this book.

From that molecular accident emerged an information architecture of staggering sophistication. DNA stores the organism's complete hereditary blueprint in a four-letter alphabet. RNA transcribes selected portions of that blueprint in response to environmental signals. Proteins translate those transcripts into functional machinery—enzymes, structural elements, signaling molecules—that constitute the organism's phenotype. And the whole system evolves: mutations introduce variation,

selection retains what works, and the genome accumulates the hard-won solutions of billions of years of trial and error.

Biological Insight

The four-nucleotide alphabet of DNA is not arbitrary. Information theory tells us that a quaternary code is remarkably close to the optimal radix for encoding information in a linear polymer with realistic chemical constraints. Each nucleotide carries $\log_2 4 = 2$ bits of information, and the double-helix structure provides both redundancy (the complementary strand) and a natural mechanism for replication. Nature discovered an encoding scheme that a communication engineer would recognize as elegant.

In Chapter 1, we saw how DNA's information architecture satisfies the fundamental requirements of any computational substrate: persistent storage, random access, error correction, and modularity. In Chapter 2, we traced the history of using DNA itself as a computational medium, from Adleman's Hamiltonian path experiment to modern DNA strand displacement circuits. In Chapter 3, we discovered that gene regulation implements a computational logic as sophisticated as anything designed by human engineers—combinatorial, context-dependent, multi-scale, and exquisitely tuned by evolution.

These three chapters established the biological foundation. They showed us not just *what* biology does, but *how* it organizes information processing. That organizational logic—the separation of storage from computation, the regulatory gating of information flow, the staged transformation from genotype to phenotype—became the blueprint for everything that followed.

19.1.2 From Biological Principles to Computational Architecture

The middle sections of this book performed a translation. We took the organizational principles of molecular biology and asked: can these be formalized mathematically? Can they be implemented in silicon? Can they produce AI systems with properties that current architectures lack?

Part II (Chapters 4–6) provided the computational toolkit: the mathematics of machine learning, the architecture of transformers, and the emergence of biological foundation models that bridge molecular data and computational methods. These chapters grounded the subsequent design decisions in established theory and practice.

Part III (Chapters 7–11) introduced the DOGMA architecture itself and its core components. Each component maps to a stage in the biological pipeline:

- **HelixHash** (Chapter 8) translates the base-pairing logic of DNA into a locality-sensitive hashing scheme for efficient similarity search in high-dimensional spaces.
- **TERA** (Chapter 9) implements transcriptional and epigenetic regulation as a learned attention mechanism that selectively activates regions of the genomic substrate.
- **TetraData** (Chapter 10) provides a tetrahedral data structure inspired by the three-dimensional organization of biological macromolecules, enabling structured multi-modal representation.
- **CodonForge** (Chapter 11) translates activated genomic representations into executable computational structures, analogous to the ribosome's translation of mRNA into protein.

Part IV (Chapters 12–14) showed how these components integrate into a system that can be trained—not just through gradient descent, but through evolutionary synthesis, combining the precision of optimization with the creativity of variation and selection.

Part V (Chapters 15–17) placed DOGMA in the broader context of post-transformer AI, AGI research, and the ethical responsibilities that accompany powerful AI systems.

And now we are here: the epilogue. The view from the summit, looking both backward at the path we climbed and forward at the terrain ahead.

The DOGMA Thesis

The central thesis of this book can be stated in a single sentence: the organizational principles that evolution discovered for biological information processing—persistent structured storage, context-dependent regulation, staged transformation, and open-ended self-modification—provide a superior blueprint for building artificial intelligence. DOGMA is one attempt to realize that blueprint. The principles themselves are more important than any particular implementation.

19.2 What DOGMA Teaches Us About Intelligence

19.2.1 Intelligence Is Substrate-Independent

One of the most profound implications of this book is that *computation is substrate-independent*. The same logical operations can be implemented in silicon transistors, DNA strand displacement reactions, protein conformational changes, or the firing patterns of neurons. What matters is not the physical medium but the *organization* of information flow within it.

This principle, which has deep roots in the work of Turing and Church, takes on new significance in the context of DOGMA. If the organizational logic of the genome—its layered regulation, its modular structure, its evolutionary plasticity—can be abstracted away from its molecular substrate and reimplemented in silicon, then intelligence itself is not a property of carbon chemistry. It is a property of *information architecture*.

Convergent Evolution of Computation

Nature has independently evolved computational systems multiple times: neural networks in animals, immune system pattern recognition, bacterial quorum sensing, slime mold optimization, and the gene regulatory networks present in all living cells. Each uses different molecular substrates, yet all implement recognizable computational primitives—memory, feedback, thresholding, signal integration. This convergent evolution of computation across substrates is itself evidence that the principles of intelligence transcend their physical implementation.

This does not mean that substrate is irrelevant. The molecular substrate of biology confers specific advantages: massive parallelism (every cell is a computer), energy efficiency (the brain runs on approximately 20 watts), self-repair, and the ability to operate in aqueous environments at ambient temperature. Silicon substrates offer their own advantages: speed, precision, programmability, and the ability to share exact copies of learned representations. The point is that the *principles* are transferable, even when the *implementation details* differ.

19.2.2 Evolution Found Solutions We Are Only Now Discovering

Throughout this book, we have encountered biological mechanisms that anticipate—sometimes by billions of years—ideas that computer scientists and AI researchers consider cutting-edge:

1. **Attention mechanisms.** Transformers use attention to selectively weight relevant inputs. Gene regulatory networks have been doing precisely this since the earliest prokaryotes: transcription factors "attend" to specific promoter sequences, with binding affinity serving as the attention weight. The parallel is not superficial; the mathematical structure is remarkably similar (Chapter 9).

2. **Mixture of experts.** Modern AI architectures route different inputs to different specialized sub-networks. The genome does the same: different cell types express different subsets of genes, creating thousands of specialized "expert networks" from a single shared genome. A liver cell and a neuron contain identical DNA but run vastly different programs.

3. **Error-correcting codes.** The double-helix structure of DNA, with its complementary strands, is a natural error-correcting code. DNA repair enzymes implement sophisticated error-detection and correction algorithms that maintain genomic integrity across billions of cell divisions.

4. **Memory-augmented computation.** The genome serves as a long-term, addressable memory that persists across generations, while epigenetic modifications provide a medium-term memory that can be adjusted within a lifetime. Current AI systems are only beginning to explore analogous architectures with external memory banks and retrieval-augmented generation.

5. **Self-modifying code.** Transposable elements, alternative splicing, and somatic hypermutation all represent mechanisms by which biological systems modify their own "code" at runtime. This is precisely the capability that DOGMA's evolutionary synthesis pipeline aims to capture (Chapter 12).

6. **Hierarchical modularity.** Genomes are organized into operons, regulons, gene families, chromosomes, and compartmentalized genomes—a hierarchy of modular organization that enables both local modification and global coordination. This mirrors the best practices of software engineering, arrived at independently by natural selection.

Nature as a Source of Algorithms

The history of computer science is filled with examples of algorithms discovered first by evolution and later by engineers: genetic algorithms, neural networks, ant colony optimization, particle swarm optimization. DOGMA proposes that this pattern extends to *architectures*, not just algorithms. The next generation of AI systems will not merely use bio-inspired algorithms within conventional architectures; they will adopt bio-inspired *organizational principles* as the architecture itself.

19.2.3 Regulation Is as Important as Computation

Perhaps the most under-appreciated lesson from biology is that *how* you control information flow matters as much as *what* you compute. The human genome contains approximately 20,000 protein-coding genes, roughly the same number as a microscopic roundworm. The difference between a human and a nematode is not primarily in the number of genes but in the sophistication of gene *regulation*—the complex, context-dependent logic that determines which genes are expressed, when, where, and at what level.

Current AI architectures largely ignore this lesson. A standard transformer processes all tokens through the same layers with the same parameters, regardless of context. There is no mechanism

for selectively activating different computational pathways for different inputs, no analog of cell-type-specific gene expression. TERA (Chapter 9) is DOGMA's answer to this gap: a regulatory attention mechanism that learns to activate different regions of the genomic substrate depending on context, task, and input characteristics.

The Cost of Universal Activation

Consider the computational waste inherent in universal activation. A large language model with 70 billion parameters applies all 70 billion parameters to every input token, whether that token is a common function word or a rare technical term requiring specialized knowledge. Biology would never tolerate such inefficiency. A liver cell does not waste energy expressing genes for photoreceptor proteins. Selective activation is not merely an optimization; it is a fundamental design principle for intelligent systems that must operate under energy constraints—which, ultimately, all real systems must.

19.3 The Central Dogma Revisited: From Biology to DOGMA

19.3.1 Crick's Original Formulation

In 1958, Francis Crick articulated the Central Dogma of molecular biology: information flows from DNA to RNA to protein, and this flow is essentially irreversible at the sequence level [Crick, 1958]. The statement was a hypothesis about molecular biology, but it was also, implicitly, a statement about computation. It said that living systems maintain a persistent, structured information store (the genome); that they selectively read from it (transcription); that the readout is transformed into functional structures (translation and folding); and that the whole process is context-dependent, regulated, and evolvable.

The Central Dogma is often summarized as a simple linear flow:

$$\text{DNA} \xrightarrow{\text{transcription}} \text{RNA} \xrightarrow{\text{translation}} \text{Protein}$$

But this summary obscures the true complexity. Transcription is not passive copying; it is a regulated, selective process controlled by hundreds of transcription factors, enhancers, silencers, and epigenetic marks. Translation is not mechanical decoding; it involves the ribosome—itself a molecular computer—performing error checking, quality control, and context-dependent codon interpretation. And protein folding adds yet another layer: the linear amino acid sequence must find its three-dimensional structure in a vast conformational landscape, guided by physical forces that encode functional information not present in the sequence alone.

19.3.2 The DOGMA Pipeline: A Computational Analog

DOGMA—the DNA-Organized Genomic Model Architecture—takes Crick's insight and asks: *what if we built an AI system that works the same way?* Not a system that processes flat token sequences through uniform attention blocks, but one that maintains a typed genomic substrate, activates regions selectively through learned regulation, transcribes active regions into intermediate representations, folds those representations into structured behavior, and evolves its own internal organization over time.

The mapping between the biological and computational pipelines is systematic and precise. The following diagram illustrates the parallel structure:

The detailed correspondence is summarized in the following table:

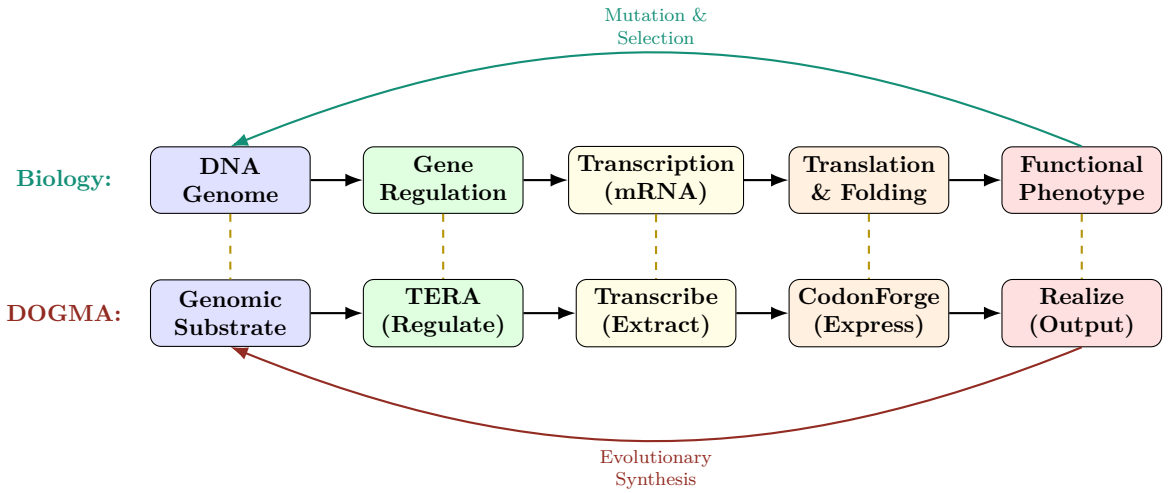


Figure 19.1: The parallel structure of the biological Central Dogma (top) and the DOGMA computational pipeline (bottom). Dashed gold lines indicate the systematic mapping between corresponding stages. Curved arrows show the evolutionary feedback loops in both systems.

Biological Stage	**DOGMA Stage**	**Computational Function**
Genome (DNA)	Genomic Base Store	Persistent, typed, addressable information substrate with modular organization
Gene Regulation	TERA (Regulate)	Context-dependent selective activation of genomic regions via learned regulatory attention
Transcription	Transcribe	Extraction of active regions into intermediate representations (computational mRNA)
Translation & Folding	Express & Realize	Transformation of intermediate representations into executable computational structures via CodonForge
Mutation & Selection	Evolutionary Synthesis	Open-ended modification of genomic substrate through variation, evaluation, and selection
Epigenetics	State Memory	Persistent, heritable modifications to activation patterns without altering the base substrate

Beyond Simple Analogy

The mapping between Crick's Central Dogma and the DOGMA pipeline is not a loose analogy. Each stage of the biological pipeline has a computational counterpart with a precise mathematical formulation developed in earlier chapters. The genome maps to a typed tensor store (Chapter 7). Gene regulation maps to TERA's regulatory attention (Chapter 9). Transcription maps to the extraction of active embeddings. Translation and folding map to CodonForge's structure synthesis (Chapter 11). Mutation and selection map to the evolutionary training framework (Chapter 12). The precision of this mapping is what distinguishes DOGMA from earlier bio-inspired approaches that borrowed biological terminology without biological rigor.

19.3.3 What the Mapping Reveals

The systematic mapping between biology and computation reveals several insights that are not obvious from either perspective alone:

Staged processing enables specialization. In biology, each stage of the Central Dogma has been independently optimized by evolution. DNA polymerase is optimized for faithful replication, RNA polymerase for regulated transcription, the ribosome for accurate translation. This stage-wise specialization is possible precisely because the pipeline separates concerns. DOGMA inherits this advantage: HelixHash can be optimized for retrieval, TERA for regulation, CodonForge for synthesis, each independently of the others.

The genotype-phenotype distinction is essential. Biology maintains a strict separation between the genome (genotype) and the organism (phenotype). This separation is what makes evolution possible: you can modify the genotype without destroying the current phenotype, test the modification in the next generation, and revert if it fails. In AI terms, this is the separation between the model's "blueprint" (its architecture and learned structure) and its "behavior" (its outputs on specific inputs). DOGMA makes this distinction explicit; standard neural networks do not.

Regulation is the key to complexity. The human genome and the roundworm genome contain roughly the same number of protein-coding genes. The vast difference in organismal complexity arises primarily from differences in *regulation*—the combinatorial logic that determines which genes are expressed in which contexts. This suggests that the path to more capable AI may lie not in adding more parameters (more "genes") but in developing more sophisticated regulation of existing parameters.

19.4 The Convergence of Biology and Computation

19.4.1 Wet Computing: DNA as a Computational Substrate

The dream of molecular computation, born in Adleman's 1994 experiment (Chapter 2), has matured dramatically. Modern DNA computing has moved far beyond proof-of-concept demonstrations:

DNA strand displacement circuits. Researchers have built cascading molecular circuits with hundreds of interacting DNA species, implementing neural networks, finite automata, and even simple programming languages entirely in molecular form. The DNA strand displacement paradigm

provides a universal computational substrate—any Boolean circuit can be compiled into a set of DNA strands and toehold-mediated displacement reactions.

DNA origami and nanostructures. The field of structural DNA nanotechnology has produced programmable two-dimensional and three-dimensional structures with nanometer precision. These structures serve as scaffolds for organizing molecular components, creating spatial arrangements that implement computational logic through geometric constraints rather than sequential reactions.

Enzymatic computing. CRISPR-based circuits, recombinase logic gates, and polymerase-driven computation extend molecular computing beyond passive hybridization. These enzymatic systems can amplify signals, maintain state, and interface with living cells, opening the door to in vivo computation.

The Speed-Parallelism Trade-off

Molecular computation is slow by silicon standards: a typical DNA strand displacement reaction takes seconds to minutes, compared to nanoseconds for a transistor switching event. But molecular systems compensate with extraordinary parallelism. A microliter of solution at micromolar concentration contains approximately 10^{14} molecules, each operating independently. The total computational throughput—operations per second per unit energy—can rival or exceed silicon for certain problem classes, particularly combinatorial search and molecular pattern recognition.

19.4.2 DNA Data Storage

One of the most commercially promising applications of DNA technology is data storage. DNA offers theoretical information density of approximately 10^{18} bytes per cubic millimeter—roughly six orders of magnitude denser than the best magnetic storage. More importantly, DNA is chemically stable: under appropriate conditions, DNA can preserve information for millennia, as demonstrated by the recovery of readable ancient DNA from specimens tens of thousands of years old.

Several research groups and companies have demonstrated the complete pipeline: encoding digital data (text, images, video) into DNA sequences, synthesizing those sequences, storing them, and then sequencing and decoding the data with high fidelity. The primary barriers to commercial deployment are the cost and speed of DNA synthesis and sequencing, both of which are improving at rates that exceed Moore's law.

For the DOGMA framework, DNA data storage is significant for a conceptual reason: it demonstrates that the boundary between "biological" and "digital" information is porous. The same molecule that stores the human genome can store a movie, a software library, or a trained neural network. This convergence of biological and digital information substrates is not a curiosity; it is a harbinger of a future in which the distinction between wetware and software becomes increasingly meaningless.

19.4.3 Synthetic Biology and Programmable Life

Synthetic biology—the engineering of biological systems for useful purposes—represents perhaps the most dramatic convergence of biology and computation. Synthetic biologists routinely design genetic circuits using principles borrowed directly from electrical engineering: logic gates, oscillators, feedback controllers, and memory elements, all implemented in DNA and protein.

The implications for AI are profound. If biological systems can be *programmed* using computational abstractions, and if computational systems can be *organized* using biological principles, then the two domains are converging toward a shared design language. DOGMA is one expression of that convergence: it takes the design language of molecular biology and applies it to the design of AI systems. Synthetic biology does the reverse: it takes the design language of engineering and applies it to the design of living systems.

Bidirectional Transfer

The relationship between biology and AI is not one-directional. DOGMA draws architectural principles from biology to build better AI. Simultaneously, AI tools—including genomic foundation models (Chapter 13)—are accelerating biological research by predicting protein structures, designing synthetic genes, and modeling regulatory networks. This bidirectional transfer creates a positive feedback loop: better AI enables deeper biological understanding, which in turn inspires better AI architectures.

19.4.4 In Vivo Computing: The Cell as a Computer

The most radical frontier of the biology-computation convergence is in vivo computing: using living cells as programmable computational devices. Engineered bacteria and mammalian cells have been programmed to perform logical operations, sense environmental conditions, make decisions, and produce therapeutic outputs—all within a living organism.

Consider the implications: a cell is a self-replicating, self-repairing, energy-harvesting computer that operates at the nanoscale. A single human cell contains approximately 6.4 billion base pairs of DNA (the "program"), 20,000 protein-coding genes (the "function library"), and thousands of active regulatory circuits (the "operating system")—all running on roughly 10^{-12} watts. No silicon computer comes close to this combination of capabilities.

DOGMA does not propose replacing silicon with cells. But it proposes learning from how cells organize computation. The cell's strategy—maintain a stable, structured genome; selectively activate relevant programs; transform representations through staged processing; evolve new capabilities over time—is precisely the strategy that DOGMA translates into silicon terms.

19.4.5 The Horizon of Hybrid Systems

Looking further ahead, we can envision hybrid systems that combine biological and silicon components within a single computational pipeline. Imagine, for instance, a system that uses DNA-based storage for its long-term memory (exploiting DNA's extraordinary information density and durability), silicon-based processors for real-time inference (exploiting electronics' speed), and engineered cellular circuits for specialized pattern recognition tasks (exploiting biology's energy efficiency and molecular-scale parallelism).

Such hybrid systems are not science fiction. The individual components—DNA data storage, silicon AI accelerators, engineered cellular circuits—already exist in laboratory settings. The engineering challenge is integration: building reliable interfaces between molecular and electronic computation, managing the radically different timescales at which biological and electronic systems operate, and developing programming abstractions that span both domains.

Bio-Silicon Interfaces

The most significant near-term opportunity for hybrid bio-silicon systems lies in *biosensors coupled to AI*. Engineered cells can detect molecular signals (metabolites, hormones, environmental toxins) with exquisite sensitivity and specificity. By coupling these biological sensors to silicon-based AI processors, we can build systems that perceive the molecular world with biological fidelity and reason about those perceptions with computational power. DOGMA's architectural framework—with its explicit separation of sensing (genomic substrate), regulation (TERA), and reasoning (CodonForge)—provides a natural template for organizing such hybrid systems.

For DOGMA, the hybrid computing frontier is particularly significant because it validates the core thesis: the principles of biological information processing are not tied to their molecular implementation. If those principles can operate in silicon, and if biological components can be integrated with silicon components, then the organizational logic—the "dogma"—truly is substrate-independent.

19.5 Open Research Frontiers

The ideas in this book open numerous research directions. We highlight here seven concrete open problems, each significant enough to sustain a research program and accessible enough that a well-prepared graduate student could make meaningful progress.

19.5.1 Open Problem 1: Scaling Evolutionary Synthesis

DOGMA's evolutionary synthesis pipeline (Chapter 12) now has an executable research protocol, but it has not yet been validated on a broad controlled benchmark suite. Scaling it to genomic substrates with billions of bases remains computationally challenging. The core difficulty is the evaluation bottleneck: each candidate genome must be instantiated, tested, and scored, and this process currently requires forward passes through the full model.

Research Direction

Key questions include: Can surrogate fitness models (themselves learned from previous evaluations) reduce the number of expensive full-model evaluations required? Can the evolutionary search be decomposed into modular sub-problems that can be evolved independently and then composed? Can techniques from population genetics—such as linkage disequilibrium analysis and recombination maps—be adapted to identify and preserve useful "genomic" structures during evolution? The goal is to achieve evolutionary optimization at scales comparable to current pre-training regimes: trillions of tokens, billions of parameters.

19.5.2 Open Problem 2: Formal Theory of Regulatory Attention

TERA (Chapter 9) implements regulatory attention as a learned mechanism for selectively activating genomic regions. But the theoretical foundations of regulatory attention remain incomplete. Under what conditions does regulatory attention converge to an optimal activation policy? What is the relationship between the expressiveness of the regulatory mechanism and the complexity of tasks it can solve? Is there a "no free lunch" theorem for regulation—a formal result showing that no single regulatory policy can be optimal for all tasks?

These questions connect to deep issues in computational complexity and learning theory. A formal theory of regulatory attention would provide principled guidance for designing TERA-like systems, rather than relying on empirical hyperparameter search.

19.5.3 Open Problem 3: Cross-Modal Genomic Representations

DOGMA's genomic substrate is, in its current formulation, primarily designed for sequence data. But biological genomes encode information that spans modalities: nucleotide sequences, three-dimensional chromatin structure, epigenetic modifications, temporal expression dynamics. A natural extension is to develop genomic substrates that natively integrate multiple modalities—text, image, audio, video, code—within a single unified representation, analogous to how a biological genome encodes the information for all cell types and all developmental stages within a single DNA molecule.

Multi-Modal Genomes

The human genome does not have separate "modules" for vision, language, and motor control. A single genomic substrate, through differential regulation, gives rise to retinal cells that process light, auditory hair cells that process sound, and motor neurons that control movement. Can a computational genomic substrate achieve similar multi-modal versatility through regulation alone, without modality-specific architectural components?

19.5.4 Open Problem 4: Interpretability Through Biological Analogy

One of the most significant potential advantages of the DOGMA architecture is interpretability. Because DOGMA's internal organization mirrors biological information flow, it may be possible to use the tools of molecular biology—gene annotation, pathway analysis, regulatory network reconstruction—to understand what a trained DOGMA system has learned. If a region of the genomic substrate consistently activates in response to mathematical reasoning tasks, we can annotate it as a "mathematical reasoning gene." If a regulatory circuit consistently co-activates two genomic regions, we can identify it as a "regulatory pathway" linking those capabilities.

This biological analogy for AI interpretability is largely unexplored. Developing it rigorously—with formal definitions of "genomic annotation" for AI systems, methods for identifying "regulatory pathways" in learned TERA attention patterns, and tools for "gene knockout experiments" (ablation studies guided by biological intuition)—would be a significant contribution to both AI safety and our understanding of learned representations.

19.5.5 Open Problem 5: Energy-Efficient Inference via Biological Sparsity

Biological neural systems achieve remarkable energy efficiency in part through extreme sparsity: at any given moment, only a small fraction of neurons are active, and only a small fraction of the genome is being transcribed. DOGMA's regulatory attention mechanism provides a natural framework for implementing similar sparsity in AI inference. But the engineering challenges are substantial.

Current GPU architectures are optimized for dense matrix operations. Sparse, irregular computation patterns—where different inputs activate different subsets of parameters—incur overhead from memory access patterns, load balancing, and synchronization. Realizing the energy efficiency that biological sparsity implies requires co-design of algorithms, software frameworks, and potentially hardware.

19.5.6 Open Problem 6: Developmental Programs for AI

Biological organisms do not spring into existence fully formed. They develop: from a single cell, through a precisely orchestrated sequence of cell divisions, differentiations, and morphogenetic events, an adult organism with trillions of specialized cells emerges. This developmental process is itself a computation, guided by the genome's regulatory logic.

Can DOGMA systems undergo analogous development? Rather than training a full-scale model from scratch, could we "grow" a DOGMA system from a small initial configuration, allowing the evolutionary synthesis pipeline to progressively add genomic regions, develop regulatory connections, and specialize computational pathways? This would mirror biological development and might offer advantages in training efficiency, modularity, and the ability to produce architectures of varying complexity from a shared developmental program.

19.5.7 Open Problem 7: Bridging Genomic Foundation Models and DOGMA

Chapter 13 introduced genomic foundation models—large neural networks trained on DNA, RNA, and protein sequences. These models capture statistical patterns in biological sequences that reflect billions of years of evolutionary optimization. But the relationship between genomic foundation models and the DOGMA architecture remains underdeveloped. Can pre-trained genomic foundation models be used to initialize DOGMA's genomic substrate with biologically meaningful structure? Can DOGMA's evolutionary synthesis pipeline be guided by biological fitness landscapes learned by genomic foundation models? Can the representations learned by genomic models inform the design of TERA's regulatory mechanisms?

This bridge between biological sequence modeling and bio-inspired AI architecture is one of the most promising and least explored research directions opened by this book.

19.6 The Next Decade of Post-Transformer AI

19.6.1 Beyond Bigger Transformers

The dominant paradigm in AI research at the time of this writing can be summarized in three words: *scale the transformer*. More parameters, more data, more compute. This strategy has produced remarkable results, but it is approaching fundamental limits—in energy consumption, in data availability, in the diminishing returns of scaling laws, and in the architectural rigidity of a system that treats all information as undifferentiated tokens.

We do not claim that the transformer era is ending. Transformers will remain central to AI for years to come, just as CPUs remain central to computing even as GPUs, TPUs, and neuromorphic chips have transformed the landscape. But the *monoculture* of transformers—the assumption that a single architecture, scaled sufficiently, will solve all problems—is already showing cracks.

The Post-Transformer Landscape

The post-transformer era will not be defined by abandoning attention mechanisms or pretrained language models. It will be defined by *augmenting* them with the organizational principles that biology discovered: persistent structured state, selective activation, staged processing, and evolutionary self-modification. DOGMA is one proposal for how to do this. It will not be the last.

19.6.2 Predictions and Possibilities

Looking ahead to the next decade, we offer not prophecies but informed possibilities—directions that the logic of this book suggests, even if their realization depends on breakthroughs we cannot foresee:

Hybrid architectures will become the norm. Just as modern processors combine CPU cores, GPU cores, and specialized accelerators, future AI systems will combine multiple architectural paradigms—transformer blocks for sequence processing, state-space models for long-range dependencies, genomic substrates for persistent structured knowledge, and evolutionary mechanisms for architectural adaptation. DOGMA is one such hybrid architecture; others will emerge.

Training will become less distinct from inference. Current AI systems have a sharp boundary between training (when parameters change) and inference (when they do not). Biology has no such boundary: learning, adaptation, and even evolution are continuous processes that occur during the organism's lifetime. Future AI systems will likely blur this boundary, continuously updating their internal representations in response to experience—not just through in-context learning within a fixed context window, but through genuine structural modification of the model itself.

Energy efficiency will become a first-order design constraint. As AI systems grow larger and more pervasive, their energy consumption becomes an increasingly serious concern—both economically and environmentally. Biology achieves extraordinary computational capability on remarkably low energy budgets. The principles that enable this efficiency—sparse activation, local computation, event-driven processing, energy-aware regulation—will become essential design constraints for AI systems, not optional optimizations.

Interpretability will drive architecture, not follow it. Currently, interpretability research tries to understand architectures designed without interpretability in mind. The DOGMA approach inverts this relationship: by building AI systems whose internal organization mirrors understandable biological processes, interpretability becomes a design feature rather than an afterthought. We expect future architectures to be designed with interpretability as a primary objective, and biological analogy will be one of the most powerful tools for achieving it.

AI systems will evolve, not just train. Perhaps the most radical prediction: AI systems will develop the capacity for open-ended self-modification—not just parameter adjustment through gradient descent, but genuine structural innovation through evolutionary processes. The evolutionary synthesis pipeline of Chapter 12 is a first step. The destination is AI systems that can create new computational structures, new representational strategies, and new problem-solving approaches that no human engineer designed or anticipated.

19.6.3 What We Are Not Predicting

Intellectual honesty requires acknowledging the limits of prediction. We are *not* predicting that DOGMA will become the dominant AI architecture. We are *not* predicting a timeline for artificial general intelligence. We are *not* predicting that biology-inspired approaches will render current methods obsolete. The history of technology is littered with confident predictions that proved spectacularly wrong.

What we *are* predicting is that the principles articulated in this book—substrate independence, regulatory gating, staged processing, evolutionary adaptation—will become increasingly central to AI design, regardless of the specific architectures that embody them. The ideas are larger than any particular implementation, including our own.

19.6.4 The Role of Hardware Co-Evolution

One dimension of the post-transformer future that deserves special mention is hardware. The transformer revolution was enabled in part by the availability of GPUs optimized for dense matrix multiplication. Similarly, the next generation of AI architectures—including DOGMA—may require hardware that is optimized for different computational patterns.

Biological computation suggests several hardware directions worth pursuing. Neuromorphic chips, which implement spiking neural networks in silicon, already capture some aspects of biological computation: event-driven processing, sparse activation, and local learning rules. But DOGMA's requirements go further. The genomic substrate requires efficient large-scale memory with content-addressable access (a natural fit for associative memory hardware). TERA's regulatory attention requires dynamic routing of computation (a natural fit for reconfigurable architectures). The evolutionary synthesis pipeline requires the ability to rapidly instantiate and evaluate architectural variants (a natural fit for field-programmable hardware).

The co-evolution of algorithms and hardware has driven every major computational revolution: CPUs enabled sequential programming, GPUs enabled deep learning, TPUs enabled large-scale transformer training. The next revolution may require hardware that is designed from the ground up to support the computational patterns of bio-inspired AI—sparse, irregular, adaptive, and self-modifying.

19.7 What Makes Intelligence "General"?

19.7.1 Beyond Benchmark Scores

The term "artificial general intelligence" (AGI) carries immense weight in contemporary AI discourse, but its meaning remains contested. Chapter 16 explored various paths toward AGI; here, in the epilogue, we offer a more philosophical reflection on what "generality" truly requires.

Current AI evaluation relies heavily on benchmarks: standardized tests that measure performance on specific tasks. A system that achieves high scores on many benchmarks is deemed more "general" than one that excels on only a few. But this metric-driven view of generality misses something essential. A student who memorizes the answers to every test in a test bank has not demonstrated general intelligence—they have demonstrated general memorization. True generality requires something more: the ability to handle situations that were not anticipated by the system's designers, to transfer knowledge across domains that share deep structural similarities but superficial differences, and to recognize when existing knowledge is insufficient and new approaches are needed.

Biological Generality

The immune system provides a biological model of genuine generality. It can mount effective responses to pathogens it has never encountered, including pathogens that did not exist when the immune system evolved. It achieves this through combinatorial diversity (generating $\sim 10^{11}$ distinct antibody configurations from a limited set of gene segments), somatic hypermutation (evolving improved responses in real time), and immunological memory (retaining successful solutions for rapid re-deployment). The immune system does not "memorize" pathogens; it generates a diverse repertoire of potential responses and then selects and refines the ones that work. This is generality through diversity and selection, not through enumeration.

19.7.2 Generality as Architectural Property

DOGMA suggests that generality is not merely a property of training data or model scale—it is a property of *architecture*. Specifically, the following architectural features contribute to genuine generality:

1. **Persistent structured state.** A system with a rich, organized internal state can accumulate knowledge over time and draw on it in novel situations. The genome is the ultimate persistent structured state: it accumulates the "knowledge" of billions of years of evolution in a format that is structured, modular, and addressable.

2. **Selective activation.** A system that can selectively activate different computational pathways for different inputs can specialize without sacrificing breadth. The key insight from gene regulation is that a single genome can give rise to hundreds of distinct cell types—each specialized, yet all sharing the same underlying substrate.

3. **Open-ended self-modification.** A system that can modify its own structure—not just its parameters—can, in principle, generate solutions to problems that lie outside its original design envelope. Evolution is the paradigmatic example: it has produced solutions (eyes, wings, immune systems, brains) that no fixed optimization process could have anticipated.

4. **Multi-scale organization.** Biological systems exhibit organization at multiple scales: molecular, cellular, tissue, organ, organism, population. Each scale has its own dynamics, constraints, and emergent properties. Truly general AI systems may need analogous multi-scale organization, with different architectural elements operating at different temporal and spatial scales.

19.7.3 Intelligence as Understanding, Not Just Performance

We close this section with a distinction that we believe is essential for the future of AI research: the distinction between *performance* and *understanding*. A system that achieves superhuman performance on a benchmark has demonstrated capability. But has it demonstrated understanding?

Understanding, as we use the term here, implies the ability to construct internal representations that capture the causal structure of a domain—not just statistical correlations, but the underlying mechanisms that generate those correlations. A system that understands Newtonian mechanics can predict the trajectory of a ball it has never seen. A system that merely correlates visual features with trajectories can only interpolate between examples it has observed.

DOGMA's staged pipeline—from genomic substrate through regulation, transcription, expression, and realization—is designed to encourage the formation of structured internal representations that

are closer to understanding than to mere correlation. The genomic substrate provides a persistent, organized store for learned knowledge. The regulatory mechanism provides context-dependent access to that knowledge. The staged processing pipeline provides opportunities for intermediate representations that capture structural relationships. Whether these design choices actually lead to systems that "understand" in any meaningful sense is, of course, an open empirical question. But the architectural intuition is clear: understanding requires structure, and structure requires design.

19.8 Honest Assessment: Where We Stand

19.8.1 What Has Been Demonstrated

Let us be precise about what DOGMA has and has not achieved as of this writing. The architecture has a formal research specification (Chapter 7), and several named components have prototype implementations or design studies—HelixHash (Chapter 8), TERA (Chapter 9), TetraData (Chapter 10), and CodonForge (Chapter 11). The measured baseline in Chapter 14 is a 29,830,396-parameter non-transformer candidate. Its low perplexity and failed generation probes expose both progress and clear failure modes. Cycle 33 used the previous gate failure to drive a single controlled mutation, reducing the supervised trace from 160 to 80 bases; the resulting regression and repeated gate failure caused another rejection. A separate ATCG-to-language hybrid passes four narrow held-out probes by using a transformer realization adapter; it is not native DOGMA language ability. Neither result establishes biological understanding, general intelligence, or superiority over transformer and state-space baselines.

Intellectual Honesty

DOGMA is a proof of concept, not a finished product. The architecture demonstrates that biology-inspired computational principles can be formalized, implemented, and tested. It does not yet demonstrate that these principles scale to the level of current frontier AI systems. That is the work that remains. We state this not as a disclaimer but as an invitation: the most important results in the DOGMA research program have not yet been achieved, and they may well be achieved by readers of this book.

19.8.2 What Remains

The open problems are substantial. Scaling the evolutionary synthesis pipeline to genomic substrates with billions of bases remains computationally challenging. Historical TERA attention proposals are not part of the canonical non-transformer baseline and require separate ablation studies. Whether DOGMA's staged pipeline can match or exceed recurrent, state-space, or transformer baselines on established benchmarks is unanswered at scale.

These are not weaknesses to be hidden. They are invitations to the research community. The history of AI is filled with ideas that were theoretically compelling but required years of engineering effort to realize their potential. Neural networks were proposed in the 1940s but did not achieve practical significance until the 2010s. The gap between a good idea and a working system is bridged not by visionaries but by engineers—patient, skilled, persistent engineers who take an idea and make it work.

19.8.3 The Community We Need

No single research group can build what DOGMA envisions. The architecture demands expertise that spans molecular biology, information theory, systems engineering, GPU programming, and cognitive science. It requires wet-lab biologists who can validate whether computational abstractions faithfully capture biological mechanisms, and it requires machine learning engineers who can make those abstractions run efficiently on modern hardware.

The future of DOGMA depends on building a community that speaks both languages fluently. This is not a platitude. It is a structural requirement. The ideas in this book cannot be fully developed by computer scientists alone, no matter how brilliant, because the biological insights that drive the architecture require genuine biological expertise to refine. Equally, biologists alone cannot realize these ideas, because the computational engineering required is specialized and demanding.

How to Contribute

If you are a biologist reading this book, your most valuable contribution may not be building DOGMA systems but *critiquing* them: identifying where our biological analogies are faithful and where they are misleading, pointing out biological mechanisms we have overlooked, and suggesting new biological principles that could inform future architectural innovations. If you are a computer scientist, your most valuable contribution may be finding ways to make these biologically inspired mechanisms computationally efficient—discovering the GPU kernels, the memory layouts, the training recipes that transform a beautiful idea into a practical system.

19.9 A Letter to the Next Generation

If you have read this far, you have absorbed a body of knowledge that spans molecular biology, information theory, machine learning, evolutionary computation, and systems architecture. You have seen how nature builds intelligent systems and how those principles can be translated into computational frameworks. You are, whether you intended it or not, prepared to contribute to the next generation of AI.

We want to be direct about what we are asking of you.

The tools in this book—HelixHash, TERA, TetraData, CodonForge, the evolutionary training framework—are starting points. They are deliberately imperfect, deliberately incomplete. They are designed to be broken, rebuilt, extended, and surpassed. If this book succeeds, it will not be because DOGMA becomes the dominant AI architecture. It will be because the *ideas* behind DOGMA—that intelligence requires structured internal state, that regulation is as important as computation, that evolution is not just a training method but a design principle—become part of how the field thinks about building intelligent systems.

19.9.1 The Importance of Interdisciplinary Thinking

Interdisciplinary Fluency

The most important skill you can cultivate is *interdisciplinary fluency*: the ability to read a paper on gene regulatory networks and see a computational architecture, to read a paper on attention mechanisms and see a model of transcriptional regulation, to move between domains without losing rigor in either. The future of AI belongs to researchers who can think across boundaries.

The history of science is filled with breakthroughs that occurred at the intersection of disciplines. Watson and Crick solved the structure of DNA not because they were the best X-ray crystallographers (Rosalind Franklin was) or the best chemists (Linus Pauling was) but because they could integrate insights from crystallography, chemistry, and biology into a single coherent model. Shannon created information theory by connecting electrical engineering, Boolean algebra, and probability theory. Turing laid the foundations of computer science by connecting mathematical logic, mechanical engineering, and philosophy of mind.

DOGMA exists at such an intersection. Its future development will require people who can move fluidly between molecular biology and machine learning, between evolutionary theory and systems engineering, between the bench and the keyboard. We do not underestimate how rare such people are, or how difficult it is to develop genuine expertise in multiple fields. But we are convinced that the most important advances in AI over the next decades will be made by researchers with exactly this kind of breadth.

19.9.2 What You Can Do Now

For the reader who finishes this book and asks "What next?"—here is concrete advice:

1. **Pick one open problem and go deep.** The seven open problems in Section 19.5 are starting points. Choose the one that most aligns with your interests and expertise, read the primary literature it connects to, and formulate a specific research question.

2. **Build something.** The labs in Chapter 18 provide implementation starting points. Take one of the DOGMA components, implement it from scratch, and then try to improve it. There is no substitute for the understanding that comes from building.

3. **Read biology.** If your background is in computer science, take a molecular biology course. Read a textbook on genetics. Attend seminars in computational biology. The biological insights that will inspire the next generation of DOGMA-like architectures are waiting to be discovered by people who can see them with computational eyes.

4. **Read computer science.** If your background is in biology, learn to program. Take a machine learning course. Implement a transformer from scratch. The computational tools that will realize biological insights into AI systems are waiting to be wielded by people who understand what biology is actually doing.

5. **Collaborate across boundaries.** Find a collaborator whose expertise complements yours. The biologist-engineer partnership is not just efficient; it is *necessary*. The most important ideas in this book emerged from conversations between people who saw the same problem from different angles.

19.9.3 The Moment You Are Entering

You are entering the field at a remarkable moment. The transformer paradigm has demonstrated that artificial intelligence at scale is possible. The question is no longer *whether* machines can exhibit intelligent behavior, but *how* to build systems whose intelligence is robust, adaptable, efficient, and aligned with human values. Biology offers answers to every one of those challenges—answers refined by three and a half billion years of evolutionary testing. Your generation has the tools, the data, and the biological knowledge to build AI systems that draw on those answers in ways that were impossible even a decade ago.

We do not know what the AI systems of 2040 will look like. But we are confident of this: they will owe more to the logic of the genome than to the architecture of the transformer. They will maintain persistent, structured internal states. They will regulate their own computation selectively. They will evolve, not merely train. And they will be built by researchers who understood that the most powerful computer ever created is not a chip fabricated in a cleanroom, but a cell dividing in a petri dish.

19.10 Key Takeaways: The Entire Book in Review

As we close this book, let us gather the most important ideas from all eighteen preceding chapters into a single synthesis. These are the ideas we most want you to carry forward.

Key Takeaways

- **DNA is an information technology** (Part I). The four-nucleotide code of DNA is not merely a chemical curiosity; it is a highly optimized information storage, processing, and transmission system. DNA computing is Turing-complete. Gene regulation implements combinatorial logic of extraordinary sophistication. Understanding biology as information processing is the foundation of everything in this book.
- **Modern AI rests on learnable representations** (Part II). Transformers, attention mechanisms, and foundation models have demonstrated that powerful AI can be built from relatively simple components trained on large data. But current architectures treat all information uniformly, lack persistent structured state, and cannot modify their own organization. Biology does all three.
- **DOGMA translates biological organization into computational architecture** (Part III). The core DOGMA components—HelixHash, TERA, TetraData, CodonForge—each formalize a specific stage of biological information processing. Together, they form a pipeline that mirrors the Central Dogma: genome → regulate → transcribe → express → realize.
- **Evolution is a design principle, not just a training method** (Part IV). DOGMA's evolutionary synthesis pipeline enables AI systems to modify their own internal organization through variation and selection. This goes beyond parameter optimization to structural innovation—the creation of new computational capabilities that no fixed architecture could produce.
- **The post-transformer future requires new organizational principles** (Part V). Scaling transformers alone will not solve AI's deepest challenges: energy efficiency, genuine understanding, open-ended learning, robust alignment. The organizational principles of biology—persistent state, selective regulation, staged processing, evolutionary adaptation—address all of these challenges.
- **Interdisciplinary thinking is not optional** (Part VI). The future of AI will be built by researchers who can bridge biology and computation, theory and engineering, abstraction and implementation. The most important breakthroughs will occur at the boundaries between disciplines, not within them.

19.11 Final Reflection

We began this book with a question: What can biology teach artificial intelligence? After eighteen chapters, hundreds of equations, and a journey that spanned molecular chemistry, information theory, machine learning, evolutionary computation, and systems architecture, we arrive at an answer that is both simple and profound.

Biology teaches us that intelligence is not a monolithic capability achieved through brute-force scaling. It is an emergent property of *organized complexity*—of systems that store information in structured, persistent substrates; that regulate the flow of information through context-dependent gating; that transform representations through staged, specialized processing; and that adapt their own organization through open-ended evolutionary search.

These are not abstract principles. They are engineering specifications, refined by 3.8 billion years of field testing in the most demanding operating environment imaginable: a planet with fluctuating temperatures, scarce resources, fierce competition, and relentless environmental change. Every living cell on Earth is a working proof of concept for these specifications. Every genome is a repository of solutions to problems that AI researchers are only now learning to formulate.

DOGMA—the DNA-Organized Genomic Model Architecture—is our attempt to translate these specifications into the language of modern computation. It is, as we have said throughout this chapter, a first draft. Its components will be improved, its principles refined, its limitations exposed and overcome. Some of its specific proposals will prove wrong. That is the nature of science: progress through principled error and correction.

But the core insight will endure: that the path to truly intelligent machines runs not only through the mathematics of optimization and the engineering of hardware, but through the biology of the cell, the logic of the genome, and the creativity of evolution. Three and a half billion years ago, in the chemistry of an ancient ocean, molecules discovered how to store information, regulate its expression, and evolve new solutions to problems that did not yet exist. That discovery—encoded in every living cell on Earth—is the blueprint we have been reading throughout this book.

The future of artificial intelligence is not in bigger transformers. It is in deeper understanding of how nature computes, evolves, and creates intelligence. DOGMA is our attempt to translate nature's blueprint into the language of computation. It is, necessarily, a first draft. The final version will be written by you—by biologists who learn to think like engineers, by engineers who learn to think like biologists, and by a generation of researchers who refuse to accept that the only path to intelligence is the one we have already found. Biology solved intelligence once. Together, we will solve it again.

Appendix A

Mathematical Foundations

This appendix collects the minimum mathematics needed to read the DOGMA chapters and evaluate the experiments. It is a reference, not a substitute for a course in linear algebra, probability, or statistics.

A.1 Functions, Vectors, and Tensors

A function $f : \mathcal{X} \to \mathcal{Y}$ maps each input in $\mathcal{X}$ to one output in $\mathcal{Y}$. A scalar is one number; a vector $\mathbf{x} \in \mathbb{R}^d$ is an ordered list of d numbers; a matrix $\mathbf{W} \in \mathbb{R}^{m \times d}$ maps a d-vector to an m-vector:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}.$$

A tensor generalizes matrices to more axes. A batch of B sequences of length T, each with hidden dimension d, commonly has shape $B \times T \times d$.

The dot product

$$\mathbf{x}^\top \mathbf{y} = \sum_{i=1}^{d} x_i y_i$$

measures signed alignment. The Euclidean norm

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$$

measures magnitude. Always write tensor shapes while deriving a model; many apparent conceptual errors are shape errors.

A.2 Probability and Information

For outcomes $x \in \mathcal{X}$, a probability distribution satisfies

$$p(x) \geq 0, \qquad \sum_{x \in \mathcal{X}} p(x) = 1.$$

Conditional probability $p(y \mid x)$ describes uncertainty about y after observing x. A causal sequence model factorizes

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t}).$$

The information content of an event is $-\log p(x)$. Rare events carry more surprise. Cross-entropy for a one-hot target y and prediction p is

$$\mathcal{L}_{\mathrm{CE}} = -\log p(y).$$

Average sequence loss gives perplexity:

$$\mathrm{PPL} = \exp(\bar{\mathcal{L}}).$$

If $\bar{\mathcal{L}} = \log 4$, then $\mathrm{PPL} = 4$. Perplexity can be compared only when tokenization, data, and evaluation procedure are compatible.

A.3 Gradients and Optimization

For parameters θ, the gradient

$$\nabla_\theta \mathcal{L}$$

contains the partial derivative of loss with respect to every parameter. Gradient descent updates

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t),$$

where $\eta > 0$ is the learning rate. Modern optimizers such as AdamW adapt the update using moment estimates and weight decay.

A gradient trains one candidate. Evolutionary search chooses among candidate configurations. These are complementary levels of optimization.

A.4 Convolution and Recurrence

A one-dimensional causal convolution with width K computes

$$y_t = \sum_{k=0}^{K-1} W_k x_{t-k}.$$

Only current and earlier inputs are used. Stacking layers expands the receptive field, but a finite stack remains locally bounded.

A recurrence carries a state:

$$h_t = f_\theta(h_{t-1}, x_t), \qquad y_t = g_\theta(h_t).$$

State-space models use structured linear recurrences whose parameters may depend on the input. Parallel-scan algorithms can evaluate associative recurrences efficiently. Calling a recurrence "DNA-inspired" does not change these mathematical requirements.

A.5 Gates and Bounded Memory

A sigmoid gate

$$g_t = \sigma(W x_t + b), \qquad \sigma(z) = \frac{1}{1 + e^{-z}},$$

lies in $(0, 1)$. It can interpolate between old and new state:

$$m_t = (1 - g_t) \odot m_{t-1} + g_t \odot \widetilde{m}_t.$$

This equation is a useful abstraction for expression or epigenetic memory. It does not imply a biological molecule is present.

A.6 Multi-Objective Evaluation

Let a candidate have metric vector

$$\mathbf{s}(c) = (s_1(c), \ldots, s_k(c)).$$

Candidate a Pareto-dominates b when it is no worse on every metric and strictly better on at least one. A scalar objective

$$J(c) = \sum_i w_i s_i(c)$$

is convenient for ranking but hides trade-offs in the weights. Hard safety or validity requirements should remain constraints rather than terms that can be paid away by another metric.

A.7 Uncertainty and Reproduction

For measurements $x_1, \ldots, x_n$, the sample mean is

$$\bar{x} = \frac{1}{n} \sum_i x_i.$$

The sample standard deviation is

$$s = \sqrt{\frac{1}{n-1} \sum_i (x_i - \bar{x})^2}.$$

Two generation seeds are enough for a smoke test, not a confidence interval. Research comparisons should repeat independent training runs and report uncertainty. When data are grouped by organism, patient, or chromosome, resampling must preserve those groups.

A.8 Complexity

Big-O notation describes asymptotic growth. A local width-K convolution over length T is $O(TK)$; if K is fixed, this is $O(T)$. Dense all-pairs attention is $O(T^2)$ in sequence length. These labels omit constants, memory traffic, hardware utilization, and hidden dimension. Wall-clock benchmarking remains necessary.

A.9 Review Problems

A.1. Verify the shapes in $\mathbf{Y} = \mathbf{X}\mathbf{W}^\top$ for $\mathbf{X} \in \mathbb{R}^{T \times d}$ and $\mathbf{W} \in \mathbb{R}^{m \times d}$.

A.2. Compute the cross-entropy when the correct byte receives probability 0.8, then compute perplexity.

A.3. For a width-5 causal convolution stacked in 4 layers without dilation, determine the receptive field.

A.4. Give two candidates that are Pareto-incomparable on accuracy and latency.

A.5. Explain why five samples from one checkpoint are not five independent training replications.

Appendix B

Canonical DOGMA Formal Specification

This appendix defines the minimum contract for the canonical non-transformer DOGMA research line. It is intentionally narrower than the full design space in Chapter 7.

B.1 Model Contract

Let the input be bytes

$$x_{1:T} \in \{0, \ldots, 255, \mathrm{EOS}\}^T.$$

An embedding maps them to

$$H^{(0)} = E(x_{1:T}) \in \mathbb{R}^{T \times d}.$$

For blocks $\ell = 1, \ldots, L$,

$$\begin{aligned}
U^{(\ell)} &= \mathrm{TetraLocal}\left(H^{(\ell-1)}\right), \\
V^{(\ell)} &= \mathrm{DNACompute}\left(U^{(\ell)}\right), \\
G^{(\ell)} &= \sigma\left(W_g V^{(\ell)} + b_g\right), \\
R^{(\ell)} &= G^{(\ell)} \odot V^{(\ell)} + (1 - G^{(\ell)}) \odot H^{(\ell-1)}, \\
H^{(\ell)} &= \mathrm{CausalTranscript}\left(R^{(\ell)}\right).
\end{aligned}$$

Optional modules may add a parallel-scan recurrent state, bounded persistent memory, or sparse allosteric routing. Every optional module must be declared in the checkpoint configuration.

The realization head gives logits

$$Z_t = W_o H_t^{(L)} + b_o, \qquad p(x_{t+1} \mid x_{\le t}) = \mathrm{softmax}(Z_t).$$

B.2 Prohibited Canonical Components

A configuration fails the canonical architecture audit if it contains:

- dot-product or thermodynamic self-attention over sequence positions;
- a transformer encoder or decoder layer; or

- an attention-enabled historical DOGMA variant presented without a distinct experimental label.

Local learned interactions inside a fixed-width cell must be implemented and documented without a query-key-value self-attention operation. “Local” alone does not make attention non-attention.

B.3 Checkpoint Contract

A checkpoint eligible for evaluation contains:

1. model state and complete model configuration;
2. parent checkpoint identity or the value `root`;
3. optimizer state when training will resume;
4. completed step and random seed;
5. data-manifest identity;
6. source-code revision; and
7. the exact evaluation-suite version.

The selected deployment pointer must name the exact checkpoint from which reported metrics and generations were produced.

B.4 Corpus Contract

Each training record has:

$$r_i = (\text{id}, \text{content hash}, \text{source class}, \text{license}, \text{transform history}, w_i).$$

Source class is at least one of $\{\text{real anchor}, \text{derived}, \text{synthetic}\}$. The canonical mixture satisfies

$$\frac{\sum_{i \in R} w_i}{\sum_i w_i} \geq 0.65.$$

The manifest records train, validation, and test hashes. Exact overlap is zero. Domain benchmarks add similarity-aware or group-aware separation.

B.5 Recursive-Cycle Contract

A candidate c produced from parent p is eligible for archive consideration only if:

$$\begin{aligned} &p \neq c, \\ &|\,\text{Seeds}(c)| \geq 2, \\ &\text{Overlap}(\text{train}, \text{eval}) = 0, \\ &\text{CriticalRegressions}(c) = 0, \\ &\text{ArchitectureAudit}(c) = \text{pass}. \end{aligned}$$

Ordinary cycles apply one bounded hyperparameter mutation. Structural mutation requires a separately declared branch and compatible initialization.

B.6 Evaluation Contract

The minimum v2 smoke suite contains:

Probe	Minimum signal	Known weakness
Canonical DNA	At least 48 consecutive A, C, G, T symbols with high canonical-letter fraction	Repetition can pass without biological meaning
Explanation	Printable output with multiple alphabetic words and no replacement characters	Word counts do not establish correctness
Self-critique	Printable limitation and experiment response	A memorized disclaimer can pass

Each probe runs under seeds 17 and 29 in the canonical scheduler. Promotion also requires finite validation and test perplexity, bounded regression, and improvement in the declared objective.

B.7 Archive and Promotion

Let $b(c)$ be a behavior descriptor and $J(c)$ a lower-is-better objective. The archive update is

$$A[b(c)] \leftarrow \begin{cases} c, & b(c) \notin A, \\ c, & J(c) < J(A[b(c)]), \\ A[b(c)], & \text{otherwise.} \end{cases}$$

Archive membership does not authorize deployment. Promotion is:

$$\text{Promote}(c) \iff \text{HardGates}(c) \wedge J(c) < J(c_{\text{live}}).$$

The update of the live pointer is atomic, and the previous live checkpoint remains available.

B.8 Claim Contract

Public reports use these labels:

Implemented The mechanism exists in the cited code revision.

Measured The result identifies checkpoint, manifest, suite, and seeds.

Reproduced An independent run or group recovered the result.

Hypothesis The mechanism or benefit has not met the measured standard.

Words such as “proves,” “outperforms,” and “general intelligence” require evidence appropriate to their scope. Architecture diagrams and parameter budgets alone do not satisfy this contract.

Appendix C

Glossary

AGI Artificial general intelligence. There is no universally accepted test. This book treats AGI as a research target requiring broad, transferable capability, not as a label for the current DOGMA model.

Allosteric routing A DOGMA hypothesis in which a small set of selected sites broadcasts non-local context. It is inspired by distant regulatory effects in proteins but implemented as ordinary digital computation.

Archive A collection of reproducible candidate checkpoints retained for scientific diversity. Archive membership is weaker than production promotion.

Attention A learned weighted aggregation over items, usually produced from queries, keys, and values. The canonical DOGMA research line does not use self-attention.

Base In biology, one of the nucleobases represented by A, C, G, T in DNA. In DOGMA, "base" may also name a typed digital symbol; the two meanings must not be confused.

Byte-level model A sequence model whose basic vocabulary is encoded bytes rather than words, subwords, codons, or nucleotides.

Candidate A trained descendant checkpoint together with its architecture, data manifest, lineage, and evaluation evidence.

Canonical DOGMA The currently governed non-transformer implementation: overlapping local memory, causal transcript computation, regulation, and declared optional recurrence or bounded memory.

Central dogma The biological framework describing major classes of sequence-information transfer among DNA, RNA, and protein [Crick, 1970]. It is not the claim that information only flows in one everyday-language direction.

Checkpoint A serialized model state, configuration, and associated training metadata.

Codon A three-nucleotide sequence interpreted during biological translation. DOGMA may use codon-inspired groupings digitally; these are not cellular translation.

Cross-entropy Average negative log probability assigned to correct targets.

DNA computing Physical computation using DNA molecules and biochemical reactions. This is distinct from running a DNA-inspired neural network on conventional hardware.

DNABERT A transformer family trained on DNA sequence representations. It is a baseline for genomic machine learning, not a wet-lab DNA computer.

DOGMA DNA-Organized Genomic Model Architecture, a MapleAI research family exploring genome-inspired organization and non-transformer sequence computation.

Epigenetic memory
A bounded persistent-state mechanism inspired by biological regulation without sequence change. The analogy does not imply biological epigenetics is being simulated faithfully.

Evaluation leakage
Any path by which held-out examples or their answers influence training, selection, or prompt design in a way not declared by the protocol.

Evolutionary search
Optimization through variation and selection among candidates. In DOGMA it operates above gradient-based parameter training.

Fitness A metric or vector of metrics used to compare candidates. A proxy fitness can be gamed and must not be confused with the complete research goal.

GenBank The public nucleotide-sequence database maintained by the U.S. National Center for Biotechnology Information and partners.

Genome The complete hereditary material of an organism. In DOGMA, "prompt genome" or "model genome" is an explicit engineering analogy for a versioned, heritable configuration.

Gradient descent Optimization that changes parameters in the direction of decreasing differentiable loss.

Held-out set Data excluded from training and used for validation or testing under a declared selection policy.

Hermon DNA A separate MapleAI/Hermon domain model line for practical DNA assistance. It must not be confused with DOGMA architecture research.

Homology-aware split
A data split designed to keep strongly related biological sequences in the same partition, reducing over-optimistic evaluation.

Lineage The recorded parent-child relationships among candidate checkpoints and their mutations.

MAP-Elites A quality-diversity algorithm that retains the highest-performing candidate in each behavior cell [Mouret and Clune, 2015].

Model collapse Degradation that can occur when generations of models train recursively on generated approximations that replace or distort the original data distribution [Shumailov et al., 2024].

Mutation A declared change from parent to candidate. Ordinary DOGMA cycles mutate one bounded training variable.

Non-transformer An architecture without transformer layers. The canonical DOGMA contract also forbids self-attention; not every non-transformer model makes that stronger choice.

Pareto dominance A relation where one candidate is no worse on every objective and better on at least one.

Perplexity The exponential of average cross-entropy. It measures next-symbol uncertainty under a fixed tokenization and dataset.

Phenotype An organism's observable traits. DOGMA uses the term metaphorically for realized model behavior.

Population-based training A method that trains a population while adapting hyperparameters by replacing weak members with mutated descendants of stronger members [Jaderberg et al., 2017].

Promotion The controlled act of making an evaluated checkpoint the live model.

Quality-diversity Search that seeks a collection of high-quality but behaviorally different solutions instead of one global solution.

Real-data anchor A training record tied to an audited external observation rather than generated solely by a model in the recursive loop.

Recurrence Sequence computation that updates and carries a state from one position to the next.

Recursive learning A governed process in which a system proposes, trains, and evaluates descendants. It does not mean unrestricted online rewriting.

Reverse complement The sequence obtained by reversing a DNA strand and replacing each base with its complement $A \leftrightarrow T$, $C \leftrightarrow G$.

Self-attention Attention where queries, keys, and values are derived from the same sequence.

State-space model A sequence model based on transitions of an internal state. Modern selective state-space models can be input-dependent and efficiently scanned.

Synthetic data Examples generated or transformed by an algorithm rather than directly observed. Provenance matters more than whether an example looks realistic.

TetraMemory DOGMA's overlapping fixed-width local memory representation. Its claimed benefits require ablation and matched-baseline evidence.

Transformer A neural architecture built around attention, residual pathways, and feed-forward blocks [Vaswani et al., 2017a].

Bibliography

Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266 (5187):1021–1024, 1994. doi: 10.1126/science.7973651.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *International Conference on Learning Representations*, 2023.

Bruce Alberts, Alexander Johnson, Julian Lewis, David Morgan, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*. W. W. Norton & Company, 7 edition, 2022.

C. David Allis and Thomas Jenuwein. The molecular hallmarks of epigenetic control. *Nature Reviews Genetics*, 17(8):487–500, 2016. doi: 10.1038/nrg.2016.59.

Uri Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. CRC Press, 2 edition, 2019.

Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. *ProPublica*, May 2016. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

Žiga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R. Ledsam, Agnieszka Grabska-Barwinska, Kyle R. Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R. Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *Nature Methods*, 18(10):1196–1203, 2021. doi: 10.1038/s41592-021-01252-x.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, 2015.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. *Proceedings of the 26th International Conference on Machine Learning*, pages 41–48, 2009.

Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, et al. Experience grounds

language. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 8718–8735, 2020.

Nick Bostrom. *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press, 2014.

Andrei Z. Broder. On the resemblance and containment of documents. *Proceedings of the Compression and Complexity of Sequences*, pages 21–29, 1997.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Moses S. Charikar. Similarity estimation techniques from rounding algorithms. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 380–388, 2002.

Kevin M. Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370–376, 2018. doi: 10.1038/s41586-018-0289-6.

Brian Christian. *The Alignment Problem: Machine Learning and Human Values*. W. W. Norton & Company, 2020.

George M. Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in DNA. *Science*, 337(6102):1628, 2012. doi: 10.1126/science.1226355.

Thomas Cremer and Christoph Cremer. Chromosome territories, nuclear architecture and gene regulation in mammalian cells. *Nature Reviews Genetics*, 2(4):292–301, 2001.

Francis Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970. doi: 10.1038/227561a0.

Francis H. C. Crick. On protein synthesis. *Symposia of the Society for Experimental Biology*, 12: 138–163, 1958.

Mark J. Daley and Lila Kari. DNA computing: Models and implementations. *Comments on Theoretical Biology*, 7:177–198, 2002. doi: 10.1080/08948550290022123.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Ober, Christian Dallago, Evan Mber, Bernardo P. de Oliveira, et al. The nucleotide transformer: Building and evaluating robust foundation models for human genomics. In *Nature Methods* Dalla-Torre et al. [2024b]. doi: 10.1038/s41592-024-02523-z.

Hugo Dalla-Torre, Liam Gonzalez, Javier Mendoza-Revilla, Nicolas Lopez Carranza, Adam Henryk Grzywaczewski, Francesco Ober, Christian Dallago, Evan Mber, Bernardo P. de Oliveira, et al. The nucleotide transformer: Building and evaluating robust foundation models for human genomics. *Nature Methods*, 2024b. doi: 10.1038/s41592-024-02523-z.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Kenneth A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2002.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

Michael B. Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000. doi: 10.1038/35002125.

ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, 2012. doi: 10.1038/nature11247.

Yaniv Erlich and Dina Zielinski. DNA fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017. doi: 10.1126/science.aaj2038.

Daniel J. Felleman and David C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, 1(1):1–47, 1991.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *Proceedings of the 34th International Conference on Machine Learning*, pages 1126–1135, 2017.

Timothy S. Gardner, Charles R. Cantor, and James J. Collins. Construction of a genetic toggle switch in *Escherichia coli*. *Nature*, 403(6767):339–342, 2000. doi: 10.1038/35002131.

Ben Goertzel and Cassio Pennachin. *Artificial General Intelligence*. Springer, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *International Conference on Learning Representations*, 2022.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Tom Head. Formal language theory and DNA: An analysis of the generative capacity of recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987. doi: 10.1016/S0092-8240(87)80006-4.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *Advances in Neural Information Processing Systems*, 35:30016–30030, 2022a.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *Advances in Neural Information Processing Systems*, 35:30016–30030, 2022b.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems. *arXiv preprint arXiv:1906.01820*, 2019.

François Jacob and Jacques Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3(3):318–356, 1961. doi: 10.1016/S0022-2836(61)80072-7.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chris Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V. Davuluri. DNABERT: Pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. In *Bioinformatics* Ji et al. [2021b], pages 2112–2120. doi: 10.1093/bioinformatics/btab083.

Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V. Davuluri. DNABERT: Pre-trained bidirectional encoder representations from transformers model for DNA-language in genome. *Bioinformatics*, 37(15):2112–2120, 2021b. doi: 10.1093/bioinformatics/btab083.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020a.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020b.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. *Proceedings of ICML*, pages 5156–5165, 2020.

Patrick J. Keeling and Jeffrey D. Palmer. Horizontal gene transfer in eukaryotic evolution. *Nature Reviews Genetics*, 9(8):605–618, 2008.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceedings of ICLR*, 2015.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. doi: 10.1162/EVCO_a_00025.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023. doi: 10.1126/science.ade2574.

Zachary C. Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

Qinghua Liu, Liman Wang, Anthony G. Frutos, Anne E. Condon, Robert M. Corn, and Lloyd M. Smith. DNA computing on surfaces. *Nature*, 403:175–179, 2000. doi: 10.1038/35003155.

Michael Lynch. *The Origins of Genome Architecture*. Sinauer Associates, 2007.

Rosario N. Mantegna, Sergey V. Buldyrev, Ary L. Goldberger, Shlomo Havlin, C.-K. Peng, M. Simons, and H. Eugene Stanley. Linguistic features of noncoding DNA sequences. *Physical Review Letters*, 73(23):3169–3172, 1994. doi: 10.1103/PhysRevLett.73.3169.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.

Jacques Monod, Jeffries Wyman, and Jean-Pierre Changeux. On the nature of allosteric transitions: A plausible model. *Journal of Molecular Biology*, 12(1):88–118, 1965.

Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. Levels of AGI: Operationalizing progress on the path to AGI. *arXiv preprint arXiv:2311.02462*, 2023.

Vernon B. Mountcastle. The columnar organization of the neocortex. *Brain*, 120(4):701–722, 1997.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Eric Nguyen, Michael Poli, Matthew Faber, Jerry Arber, Jason Gross, Neel Goel, Alexander Wettig, Brando Erber, Hyunji Nam, Stephen Baccus, et al. Sequence modeling and design from molecular to genome scale with Evo. *Science*, 386(6723):eado9336, 2024. doi: 10.1126/science.ado9336.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019. doi: 10.1126/science.aax2342.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *Transformer Circuits Thread*, 2022.

Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarova, George M. Church, Karin Strauss, and Luis Ceze. Random access in large-scale DNA data storage. *Nature Biotechnology*, 36(3):242–248, 2018. doi: 10.1038/nbt.4079.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. RWKV: Reinventing RNNs for the transformer era. *Findings of EMNLP*, 2023.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *International Conference on Machine Learning*, 2023.

Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011. doi: 10.1126/science.1200520.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing machines. In *DNA Based Computers*, volume 27 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 75–120. American Mathematical Society, 1996.

Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082): 297–302, 2006. doi: 10.1038/nature04586.

Stuart Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Viking, 2019.

Kensaku Sakamoto, Hidetaka Gouzu, Ken Komiya, Daisuke Kiga, Shigeyuki Yokoyama, Takashi Yokomori, and Masami Hagiya. Molecular computation by DNA hairpin formation. *Science*, 288 (5469):1223–1226, 2000. doi: 10.1126/science.288.5469.1223.

John SantaLucia. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences*, 95(4):1460–1465, 1998. doi: 10.1073/pnas.95.4.1460.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2023a.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2023b.

Georg Seelig, David Soloveichik, David Yu Zhang, and Erik Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314(5805):1585–1588, 2006. doi: 10.1126/science.1132493.

Nadrian C. Seeman. DNA in a material world. *Nature*, 421(6921):427–431, 2003. doi: 10.1038/nature01406.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, 2016.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 (3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. AI models collapse when trained on recursively generated data. *Nature*, 631:755–759, 2024. doi: 10.1038/s41586-024-07566-y.

David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010. doi: 10.1073/pnas.0909380107.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017b.

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *arXiv preprint arXiv:2211.04325*, 2022.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, 2019.

Richard F. Voss. Evolution of long-range fractal correlations and $1/f$ noise in DNA base sequences. *Physical Review Letters*, 68(25):3805–3808, 1992. doi: 10.1103/PhysRevLett.68.3805.

James D. Watson and Francis H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171(4356):737–738, 1953. doi: 10.1038/171737a0.

Joseph L. Watson, David Juergens, Nathaniel R. Bennett, Brian L. Trippe, Jason Yim, Helen E. Eisenach, Woody Ahern, Andrew J. Borber, Robert J. Ragotte, Lukas F. Milles, et al. De novo design of protein structure and function with RFdiffusion. *Nature*, 620(7976):1089–1100, 2023. doi: 10.1038/s41586-023-06415-8.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022b.

Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–544, 1998. doi: 10.1038/28998.

Bernard Wood. *Human Evolution: A Very Short Introduction*. Oxford University Press, 2014.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*, 2023.

Joseph N. Zadeh, Conrad D. Steenberg, Justin S. Bois, Brian R. Wolfe, Marshall B. Pierce, Asif R. Khan, Robert M. Dirks, and Niles A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32(1):170–173, 2011. doi: 10.1002/jcc.21596.

David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, 3(2):103–113, 2011. doi: 10.1038/nchem.957.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Jian Zhou and Olga G. Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12(10):931–934, 2015. doi: 10.1038/nmeth.3547.