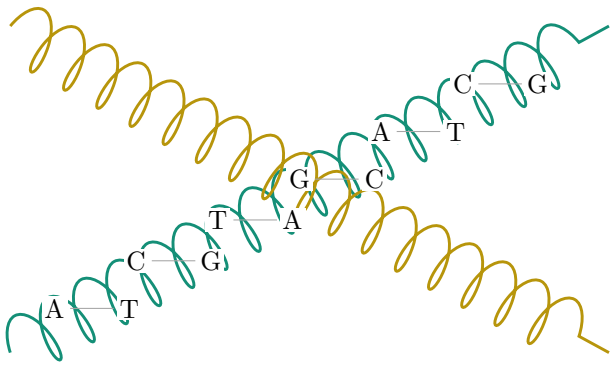
DNA COMPUTING FOUNDATIONS

From Molecular Information to Learning Systems

An Undergraduate Introduction to Biological Information, Molecular Algorithms, and Genomic Models

First Edition — Open Textbook

Wenyan Qin



July 2026

First Edition, v1.0, July 2026.

This volume separates established DNA-computing foundations from the experimental DOGMA research program. Claims are labeled by evidence level, and biological analogy is never treated as proof of computational advantage.

Preface

DNA is simultaneously a molecule, a durable information medium, and the substrate of a distributed chemical control system. Those roles make it a remarkable subject for computer scientists, but they also create a common source of confusion: biological information processing, wet-lab DNA computation, and machine learning on genomic sequences are related fields, not interchangeable names for the same technology.

This book builds a careful bridge among them. It begins with nucleotides and the central dogma of molecular biology, develops strand displacement and chemical reaction networks as algorithms, studies gene regulation as conditional computation, and then supplies the machine-learning background needed to understand modern genomic foundation models. The aim is not to claim that biology has already solved artificial intelligence. The aim is to give students enough biology, computation, and mathematics to ask better questions and design reproducible experiments.

Audience. The text is written for undergraduate students in computer science, biology, engineering, and data science. Introductory programming, calculus, linear algebra, and probability are helpful. No prior molecular biology is assumed.

Evidence discipline. Established biological mechanisms are identified as biological facts and cited to the literature. Results from other laboratories are external results. Proposed computational mechanisms remain hypotheses until an implementation, a frozen evaluation, and a reproducible artifact support a stronger label.

Relationship to the companion volume. The companion research monograph, *DOGMA: DNA-Organized Genomic Model Architecture*, explores a specific non-transformer architecture inspired by persistent state, regulation, local interaction, expression, and population selection. Readers can study this foundations volume independently.

Wenyan Qin
July 2026

Reader's Roadmap

Ch.	Title	Learning focus
1	DNA as an Information System	Nucleotides, base pairing, information content, replication, transcription, translation, and biological error correction
2	DNA Computing	Adleman's experiment, molecular search, strand displacement, chemical reaction networks, logic gates, complexity, and laboratory constraints
3	Gene Regulation as Computation	Promoters, repressors, enhancers, feedback, epigenetic state, and context-dependent expression
4	Machine-Learning Foundations	Representation, optimization, generalization, sequence models, evaluation, and experimental controls
5	Transformers and Large Language Models	Attention, tokenization, scaling, strengths, limitations, and why comparison architectures matter
6	Biological Foundation Models	Genomic and protein language models, structure prediction, benchmarks, interpretability, and responsible use

Fast route for computer scientists: Chapters 1, 2, 4, 5, and 6.
Fast route for life-science students: Chapters 1, 3, 4, and 6.
Full course: All chapters in order, with Appendix A used as a just-in-time mathematics reference.

Contents

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.

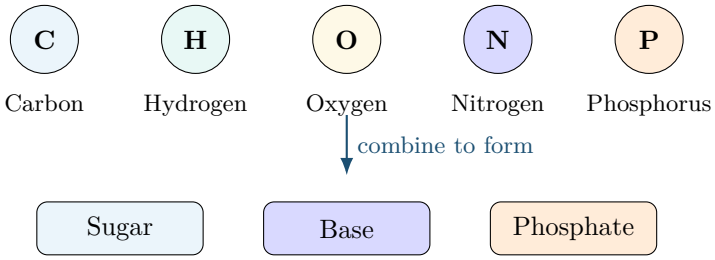


Figure 1.1: The five elements of DNA combine to form three molecular building blocks: a sugar, a nitrogenous base, and a phosphate group.

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.

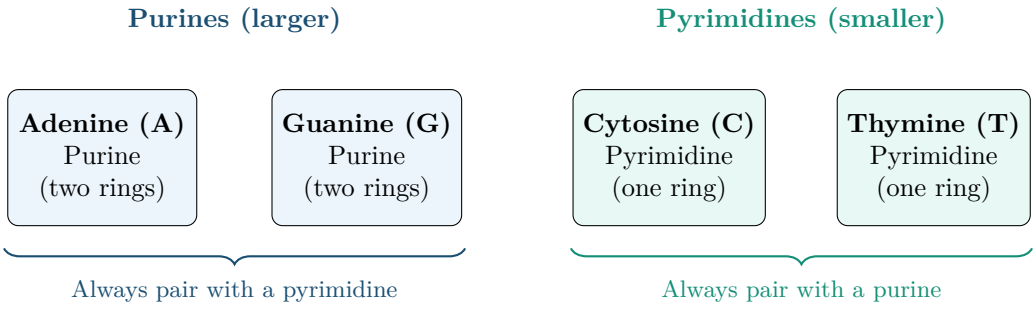


Figure 1.2: The four DNA bases grouped by chemical family. A purine always pairs with a pyrimidine, keeping the helix diameter constant.

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.

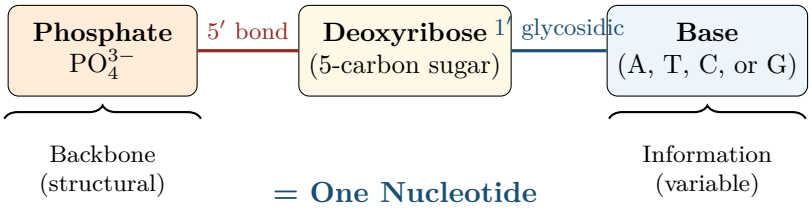


Figure 1.3: A nucleotide consists of three parts. The phosphate and sugar form the structural backbone; the base carries the genetic information.

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).

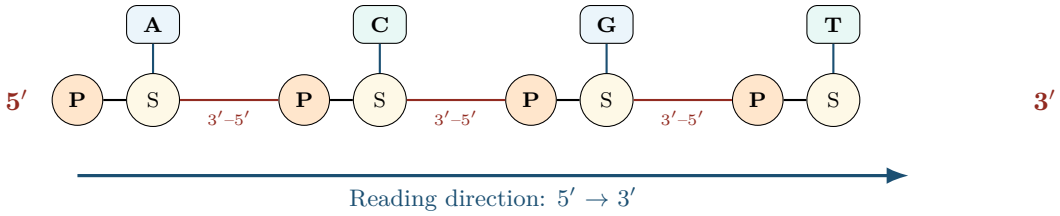
1.1.4 Step 4: The Single Strand — A Polynucleotide Chain

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

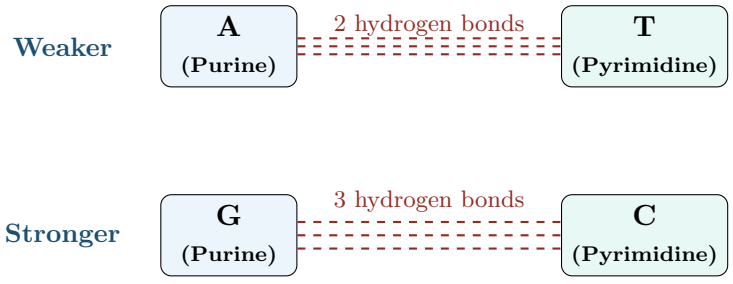
The discovery of Watson and Crick was that DNA exists as two antiparallel strands wound around each other in a right-handed helix, held together by specific hydrogen bonds between complementary bases [Watson and Crick, 1953]:

- **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.

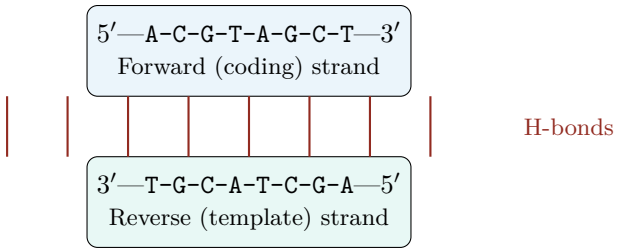
The DNA Alphabet

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.



Key properties: (1) Complementary base pairing provides built-in redundancy and error detection. (2) Antiparallel orientation enables bidirectional information encoding. (3) The sequence s and its reverse complement $\overline{s}^R$ carry equivalent information—a symmetry that DOGMA's HelixHash exploits computationally.

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 2).

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

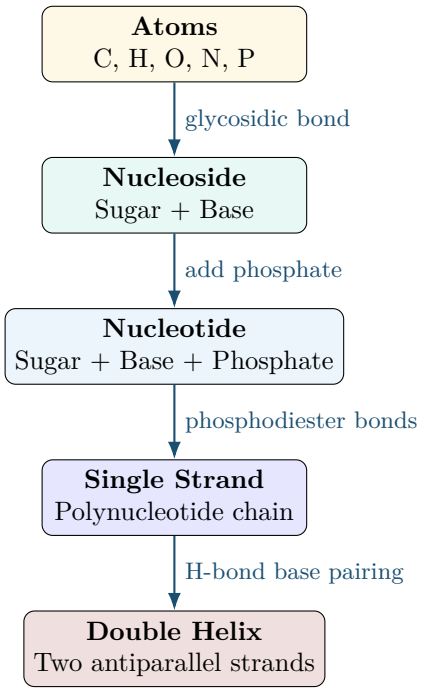


Figure 1.7: Building DNA step by step: from atoms to the double helix in five levels of organization.

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].

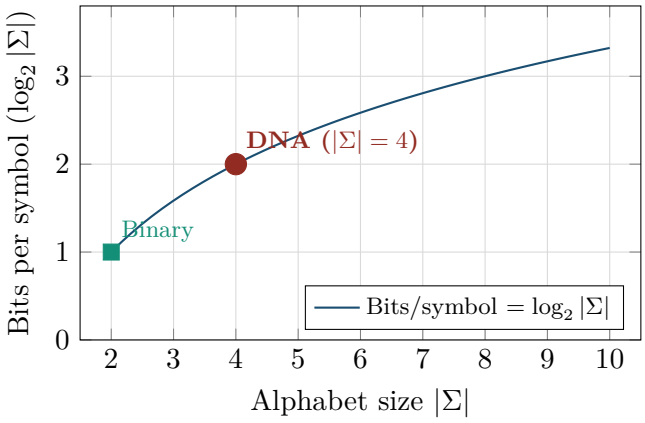


Figure 1.8: Information density as a function of alphabet size. DNA's four-letter alphabet provides 2 bits per symbol—a good balance between density and biochemical fidelity.

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.

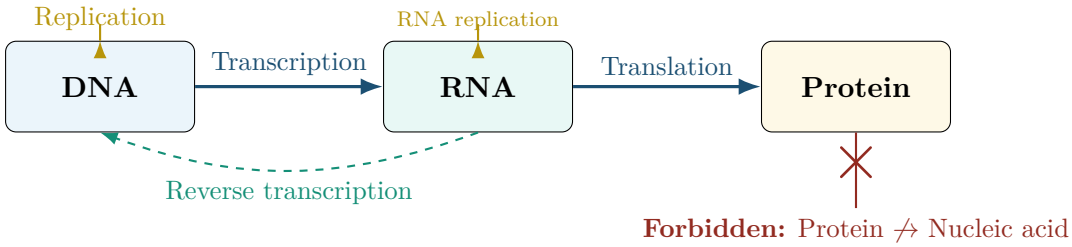


Figure 1.9: The Central Dogma of molecular biology. Solid blue arrows show the main information flow; dashed teal shows reverse transcription; gold loops show self-replication. Information never flows from protein back to nucleic acid.

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 5) 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.

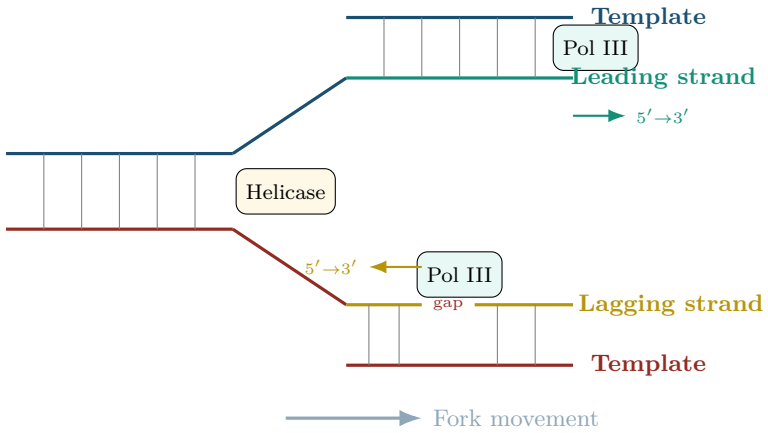


Figure 1.10: The DNA replication fork. Helicase unwinds the double helix. The leading strand (teal) is synthesized continuously; the lagging strand (gold) is synthesized as Okazaki fragments.

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 8 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 8, 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.

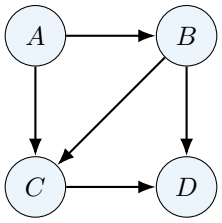
For the DOGMA reader, this chapter serves a specific purpose: it establishes that the computational primitives of DNA—pairing, displacement, catalysis, self-assembly—are not merely biological curiosities. They are *computational operations* that can be formalized, abstracted, and reimplemented on silicon.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

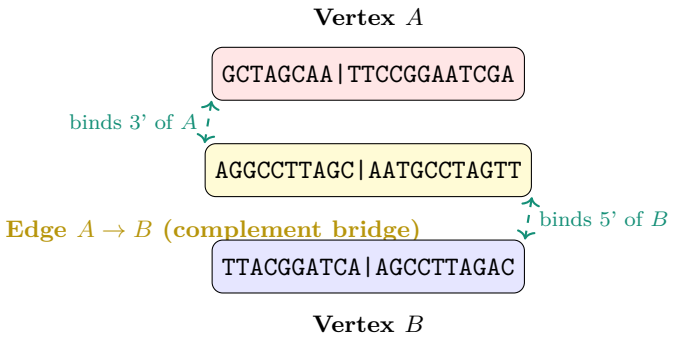
2.1.2 Encoding Graphs in DNA

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.

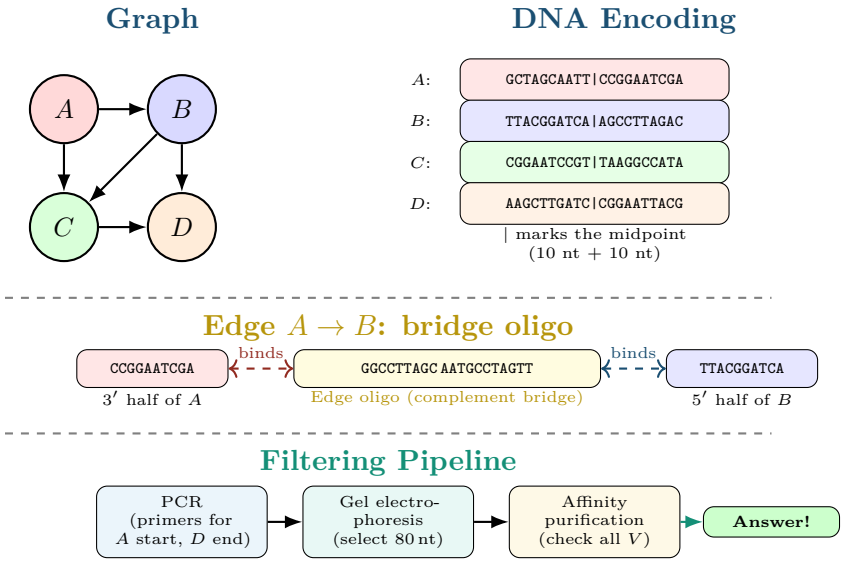


Figure 2.1: Adleman's experiment: a directed graph is encoded in DNA by assigning 20-mer sequences to vertices and complement-bridge oligos to edges. After self-assembly in solution, three filtering steps extract the Hamiltonian path [Adleman, 1994].

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

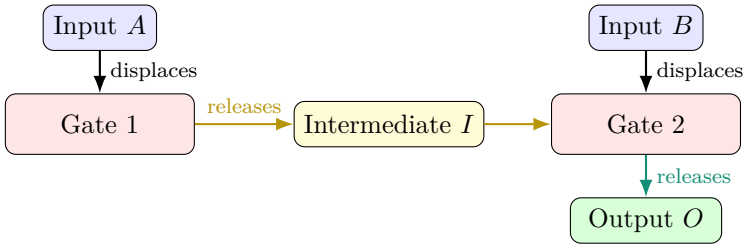
An AND gate produces output 1 only when *both* inputs are 1.

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 8) 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

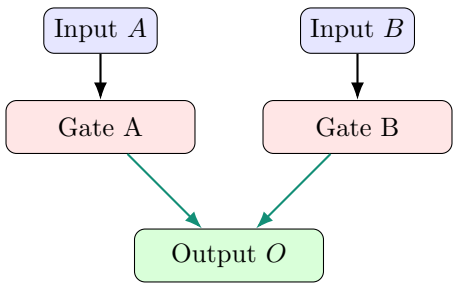
An OR gate produces output 1 when *at least one* input is 1.

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.

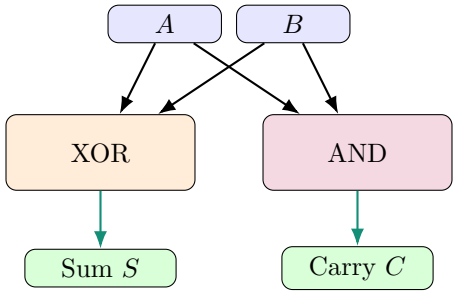


Figure 2.4: DNA half adder: parallel XOR and AND circuits sharing input strands.

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 1) 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.

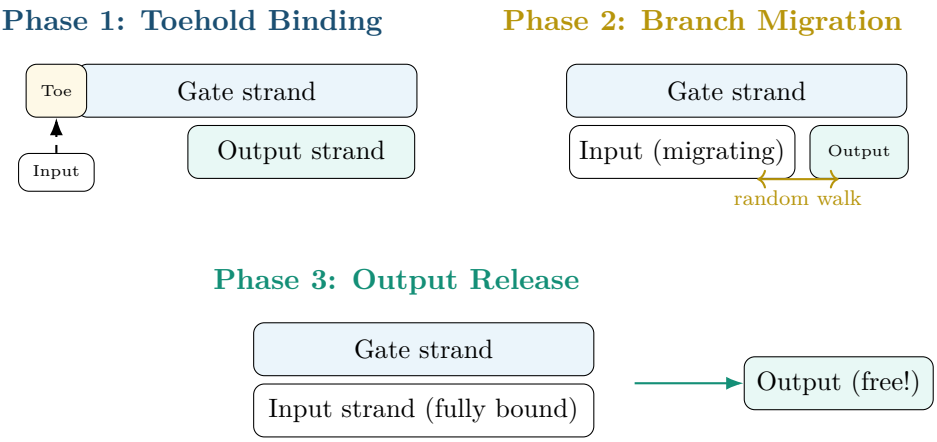


Figure 2.5: The three phases of toehold-mediated strand displacement.

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.

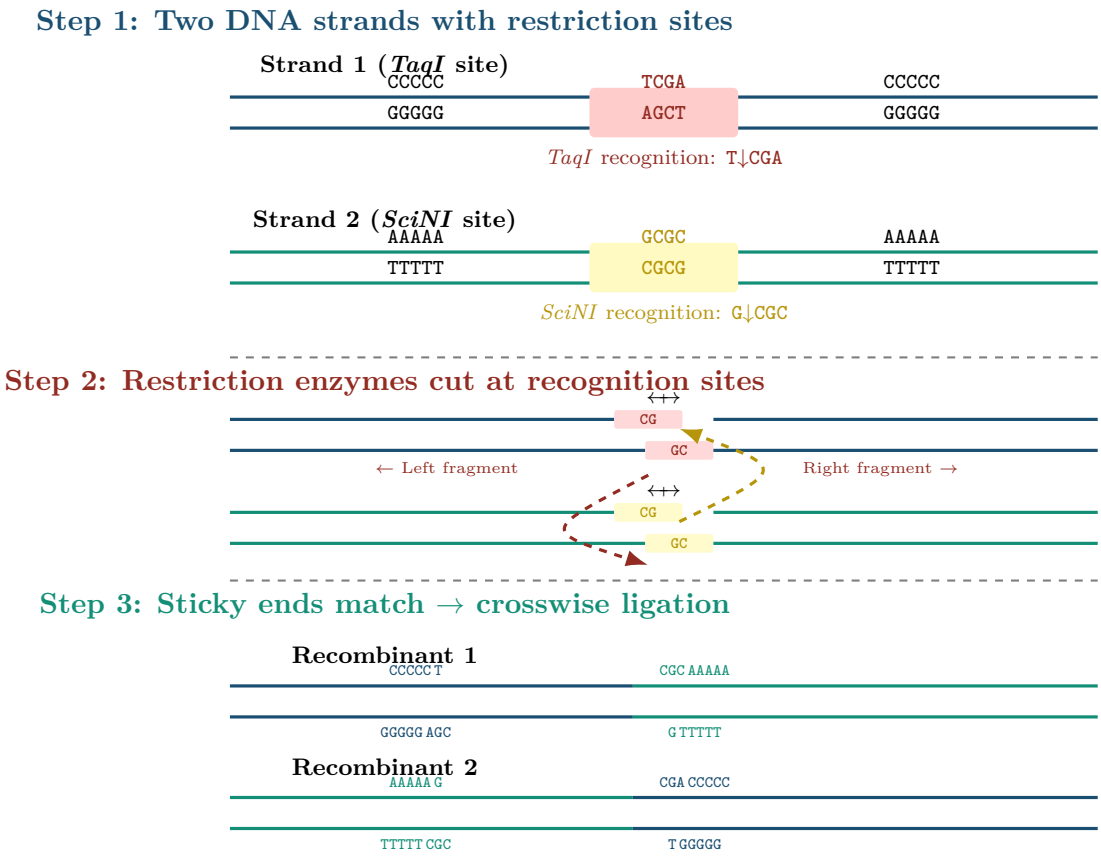


Figure 2.6: Step-by-step splicing: restriction enzymes cut two DNA strands at their recognition sites, producing sticky ends that recombine crosswise to form two new recombinant molecules.

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:

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

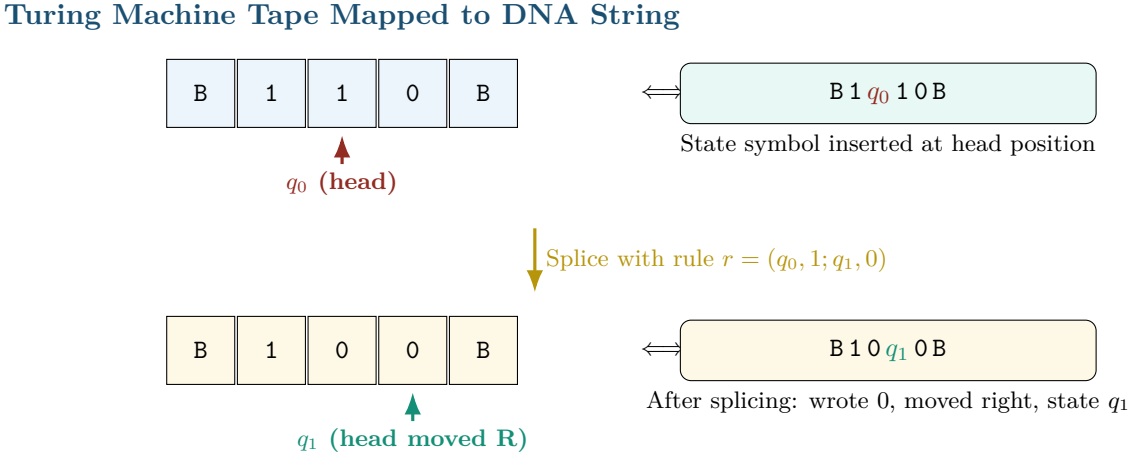


Figure 2.7: A Turing machine tape configuration encoded as a DNA string. One splicing step simulates one Turing machine transition: the state symbol moves, the tape cell is rewritten, and the head advances.

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:

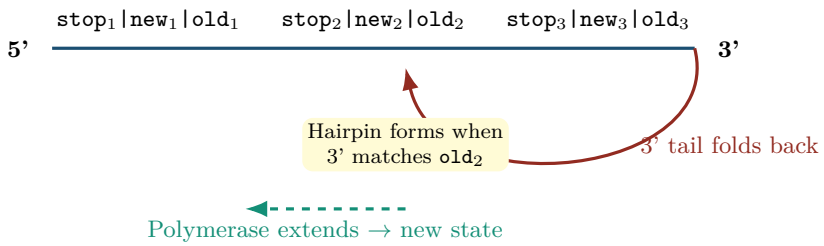


Figure 2.8: Whiplash PCR: a single DNA strand folds into a hairpin when the 3′ tail (current state) recognises a complementary transition entry. DNA polymerase extends the tail, copying the new-state sequence and advancing the computation.

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 1) 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.

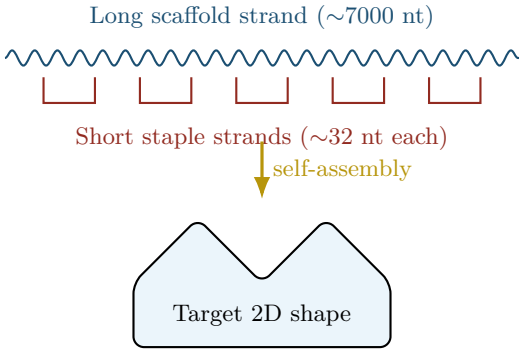


Figure 2.9: DNA origami: staple strands fold a long scaffold into a prescribed 2D shape.

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 4), 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 8, 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.

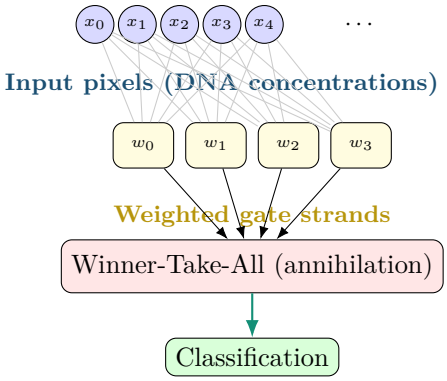


Figure 2.10: Architecture of a DNA neural network for pattern classification.

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 5) 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 1) 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.

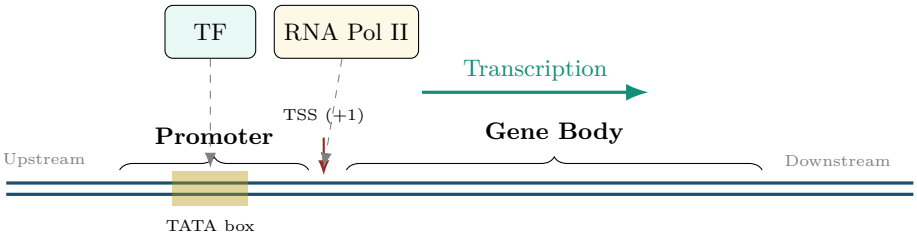


Figure 3.1: A gene promoter region. Transcription factors (TFs) bind to specific sites within the promoter, recruiting RNA Polymerase II to the transcription start site (TSS). Transcription proceeds downstream through the gene body.

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:

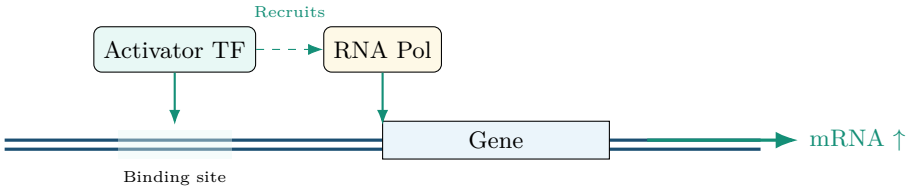
1. **DNA-binding domain (DBD):** A structural motif that recognizes and binds a specific short DNA sequence (typically 6–12 bp), called a *binding motif* or *response element*.
2. **Effector domain:** A region that interacts with other proteins to either *activate* or *repress* transcription.

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

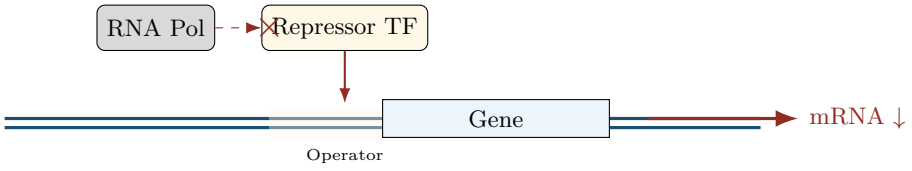


Figure 3.2: (a) An activator TF binds its target site, recruits RNA Polymerase, and increases transcription. (b) A repressor TF binds the operator, physically blocking RNA Polymerase and decreasing transcription.

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).

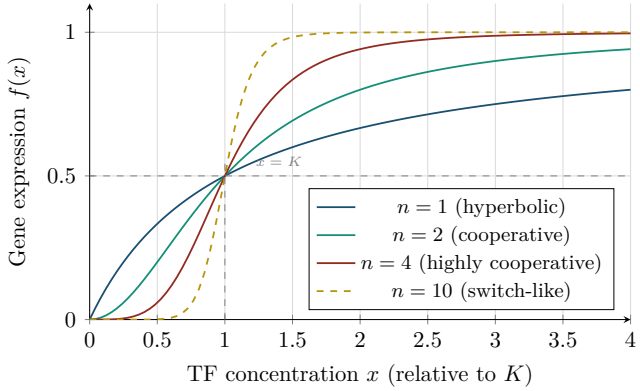


Figure 3.3: The Hill function for different Hill coefficients n. As n increases, the response becomes increasingly switch-like. At $x = K$, the output is always 0.5 (half-maximal). For $n = 1$, the response is gradual (Michaelis–Menten kinetics). For large n, the response approximates a digital step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

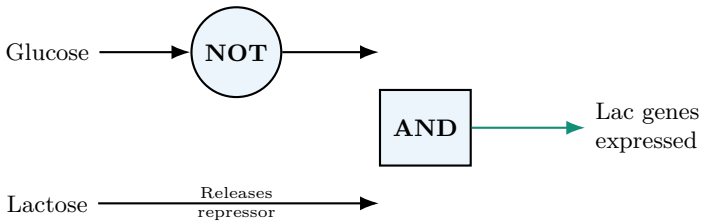
The logic is elegant: the cell should only invest energy in making lactose-digesting enzymes when two conditions are *both* true: (1) lactose is available, and (2) the preferred sugar, glucose, is absent. This is an AND gate with one inverted input.

```
# 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 3).

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.

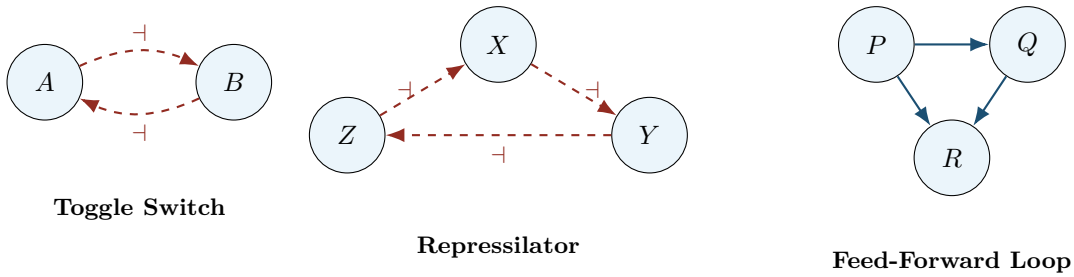


Figure 3.5: Common gene regulatory network motifs. Solid arrows indicate activation; dashed arrows indicate repression. Each motif implements a distinct computational function.

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.

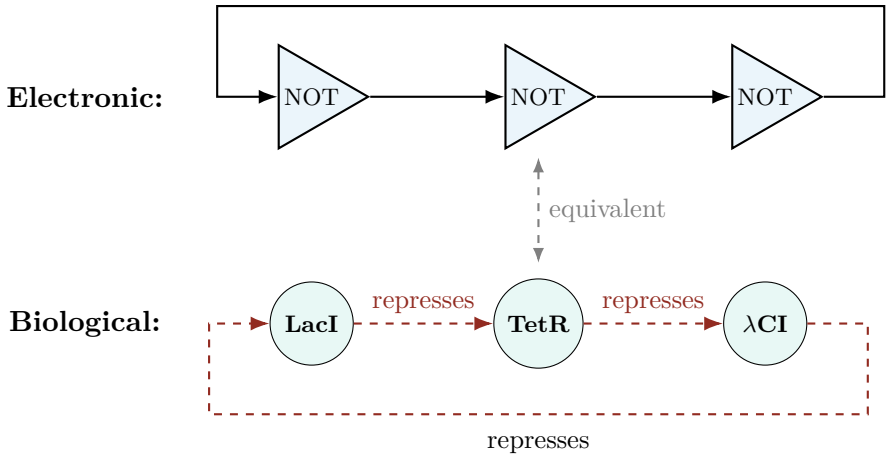


Figure 3.6: The repressilator is the biological equivalent of an electronic ring oscillator. Three repressor genes form a loop where each represses the next, producing sustained oscillations in protein expression levels.

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.

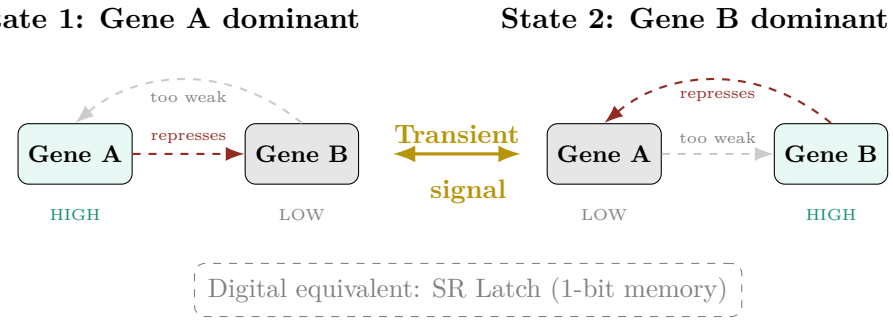


Figure 3.7: The toggle switch as a bistable flip-flop. In State 1, Gene A is dominant and represses Gene B. In State 2, Gene B is dominant and represses Gene A. A transient signal can flip between states, but each state is self-maintaining—a 1-bit biological memory.

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.

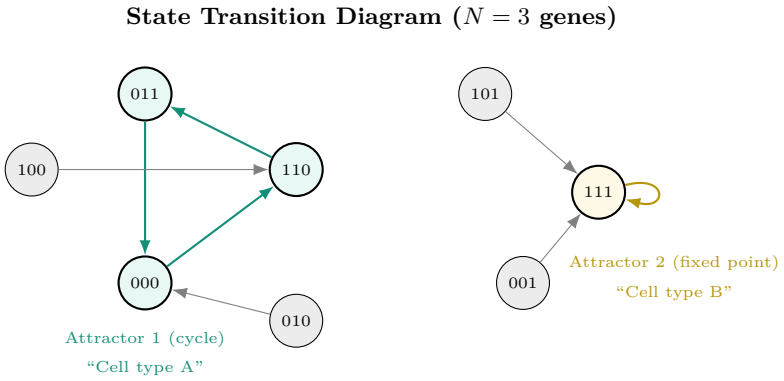


Figure 3.8: State transition diagram for a hypothetical 3-gene Boolean network. The $2^3 = 8$ possible states are connected by the network's update rule. States eventually reach *attractors*—either fixed points or cycles. Each attractor corresponds to a stable cell type. Transient states represent differentiating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

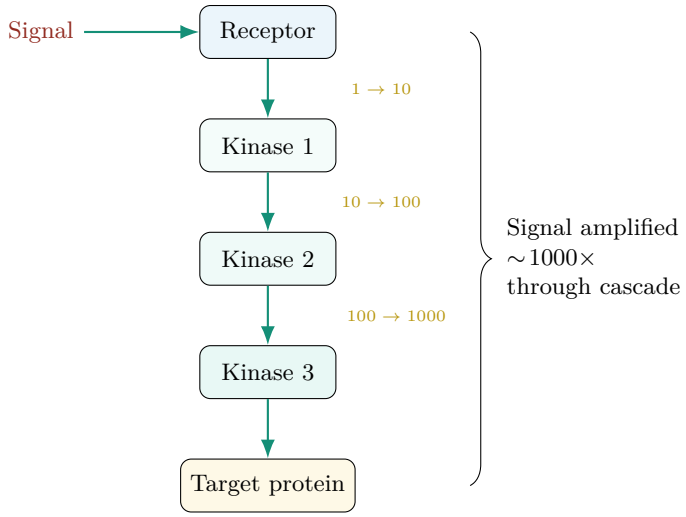


Figure 3.9: A kinase cascade amplifies a weak extracellular signal into a strong intracellular response. Each level activates many copies of the next kinase, producing exponential amplification—analogous to a transistor amplifier chain.

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).

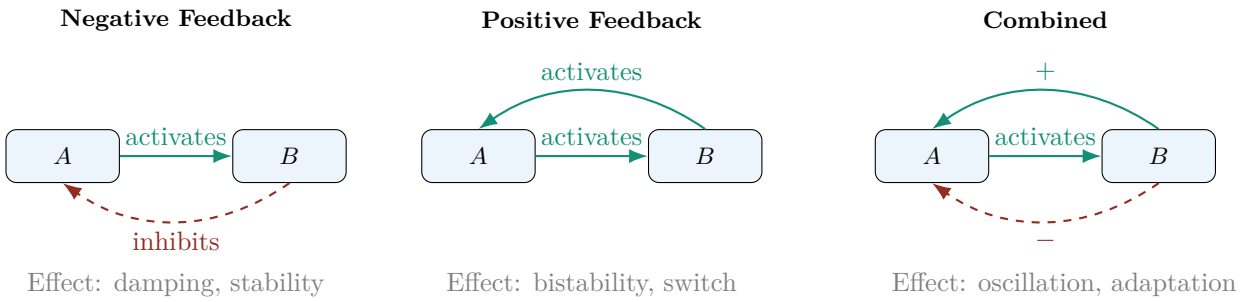


Figure 3.10: Three types of feedback in signal transduction. Negative feedback creates stability and damping. Positive feedback creates bistability and switching. Combined feedback can produce oscillations or adaptation.

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.

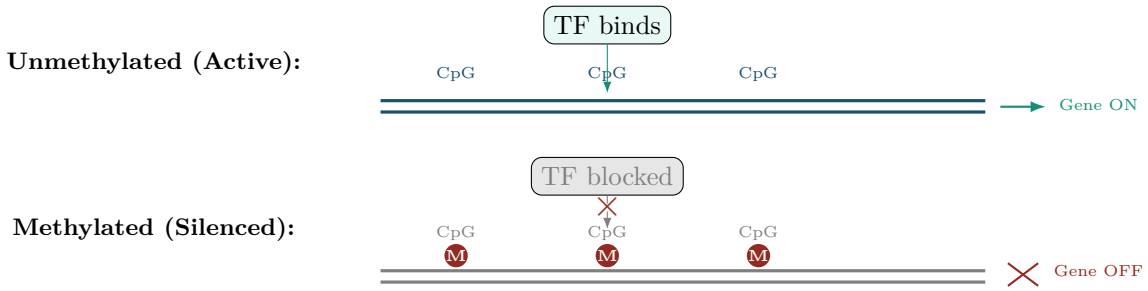


Figure 3.11: DNA methylation silences genes. Methyl groups (M) added to CpG sites block transcription factor binding, effectively switching the gene off without changing the DNA sequence.

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 3) 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

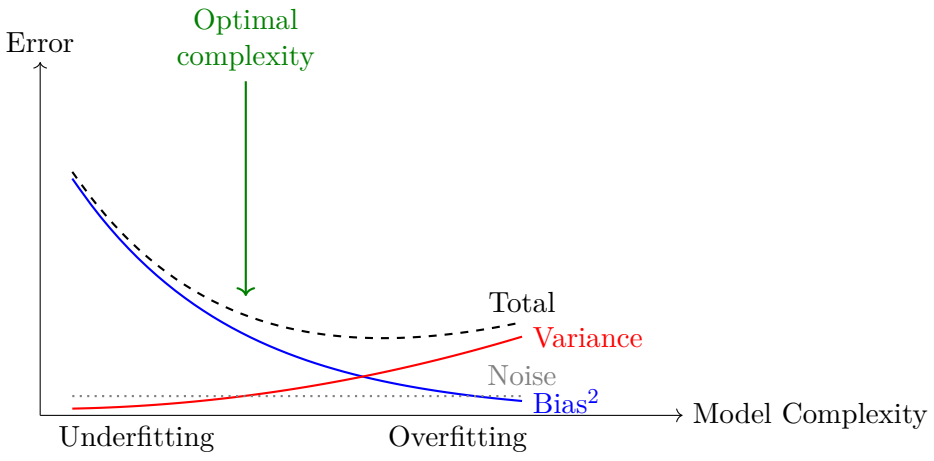
The expected error of a model can be decomposed as:

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

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., 2021]. 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.

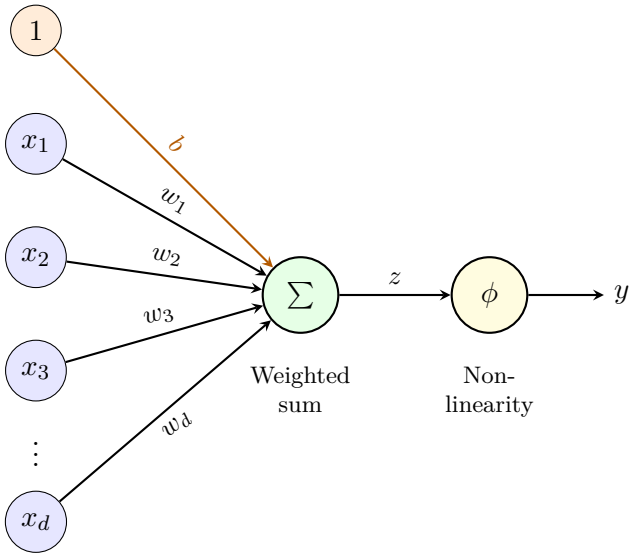
4.2 Neural Networks from First Principles

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

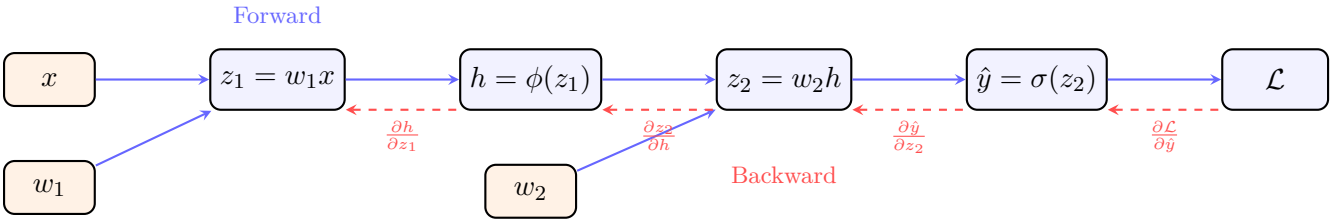
4.3.1 Backpropagation: Computing Gradients Efficiently

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:

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

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

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

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

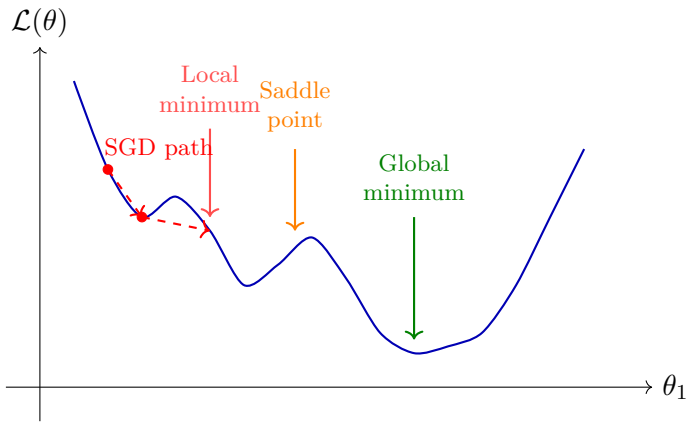
where $g_t = \nabla_\theta \mathcal{L}(\theta_t)$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ are the standard defaults. The bias correction terms ($\hat{m}_t$ and $\hat{v}_t$) are essential in early training when the moving averages are biased toward zero due to initialization at $m_0 = v_0 = 0$. 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 6).

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 1) 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., 2021].

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:

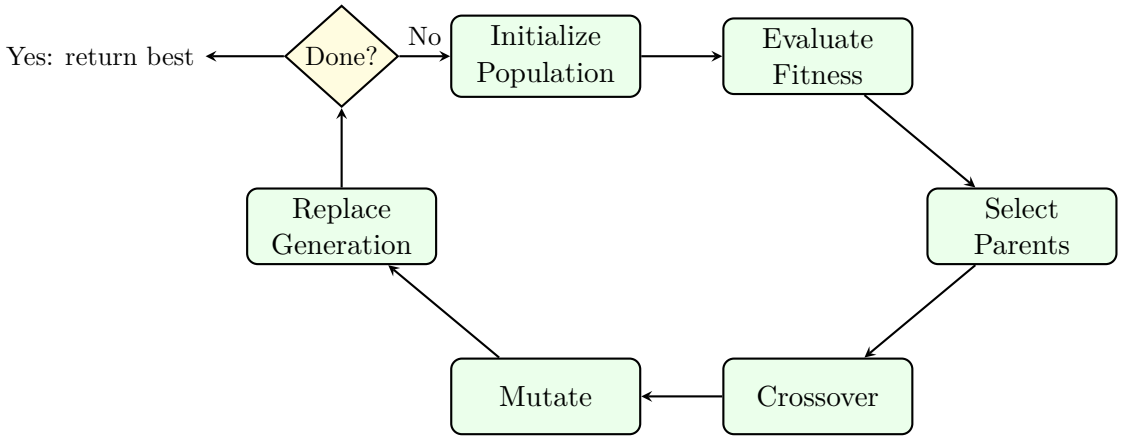
1. **Initialize:** Create a random population of N candidate solutions ("individuals"), each encoded as a genome (e.g., a bit string, a real-valued vector, or a structured object).

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 6) 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 2) 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

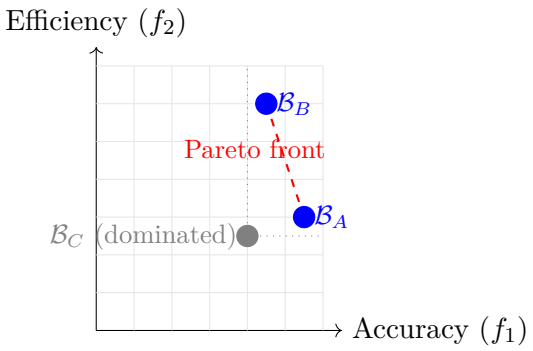
Definition 4.1 (Pareto Dominance and Optimality)**.** Given two solutions $\mathbf{x}$ and $\mathbf{y}$ evaluated on m objectives $f_1, \ldots, f_m$ (all to be maximized):

- $\mathbf{x}$ *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 3), 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. [2021]: 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 1.

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 1) 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.

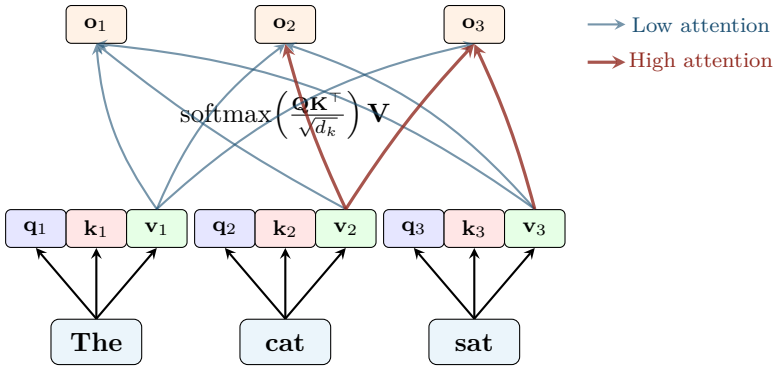


Figure 5.1: Self-attention mechanism for three tokens. Each token is projected into query ($\mathbf{q}$), key ($\mathbf{k}$), and value ($\mathbf{v}$) vectors. Attention scores (computed as $\mathbf{q}_i \cdot \mathbf{k}_j/\sqrt{d_k}$, then softmax-normalized) determine how much of each value vector 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

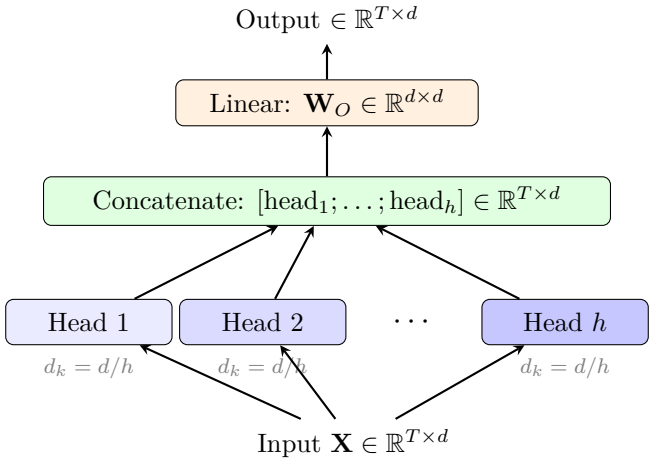


Figure 5.2: Multi-head attention. The input is processed by h independent attention heads, each operating in a (d/h)-dimensional subspace. Outputs are concatenated and projected back to the full d-dimensional space.

5.3 The Transformer Architecture

Vaswani et al. [2017] 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.

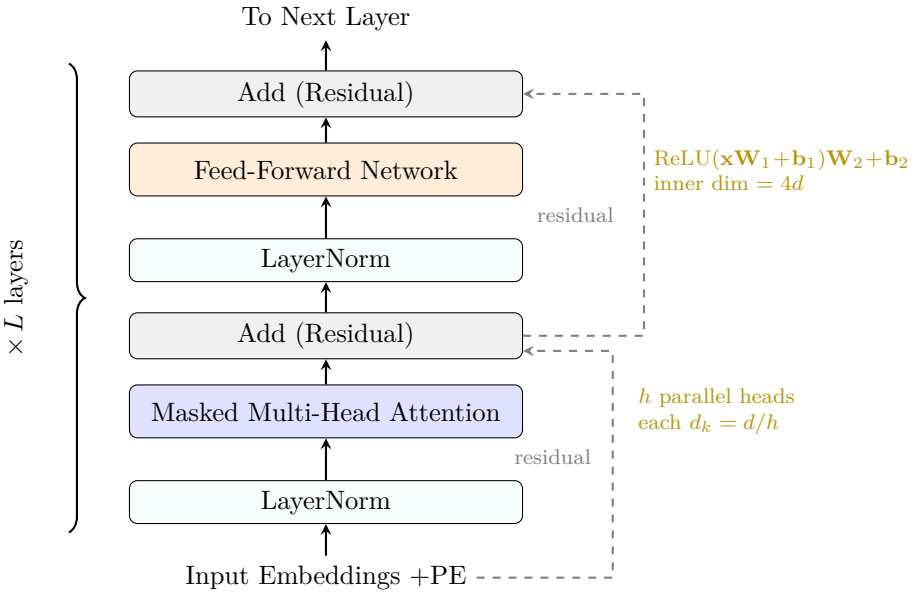


Figure 5.3: A single transformer decoder block with pre-norm residual connections. The masked multi-head attention sub-layer prevents each position from attending to future tokens. LayerNorm is applied before each sub-layer (pre-norm configuration). The block is repeated L times to form the full decoder 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. [2017] 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. [2020] 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. [2022] 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.

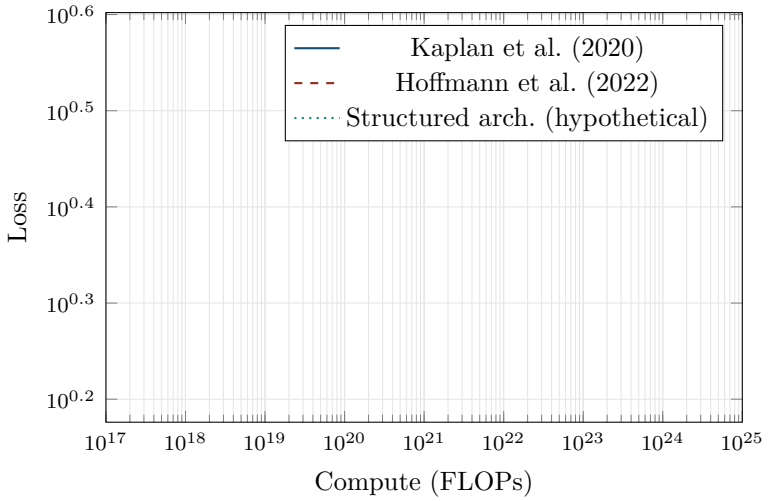


Figure 5.4: Scaling laws for language models. Loss decreases as a power law in compute. The Chinchilla-optimal curve achieves lower loss than the Kaplan curve at each compute level by better balancing parameters and data. The dotted line illustrates the hypothesis that structured architectures could 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 1 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. [2022] 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., 2022]. 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., 2023].

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 6) 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 6) 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?

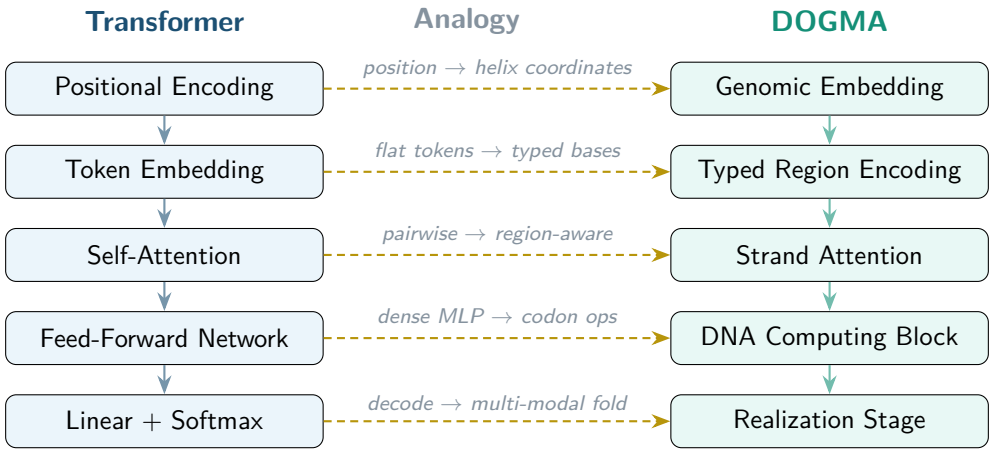


Figure 5.5: Side-by-side comparison of the Transformer architecture and the DOGMA architecture, highlighting the biological analogues. Each Transformer component has a DOGMA counterpart that adds biological structure: positional encoding becomes genomic (helix-aware) embedding, self-attention becomes 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 1.

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. [2020] and Hoffmann et al. [2022]. 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., 2020] reveal smooth power-law relationships between compute, data, parameters, and loss. Chinchilla [Hoffmann et al., 2022] 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., 2022].
- 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. [2017]: 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. [2020]: Neural scaling laws—empirical foundations for the scaling paradigm.
- Hoffmann et al. [2022]: 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:

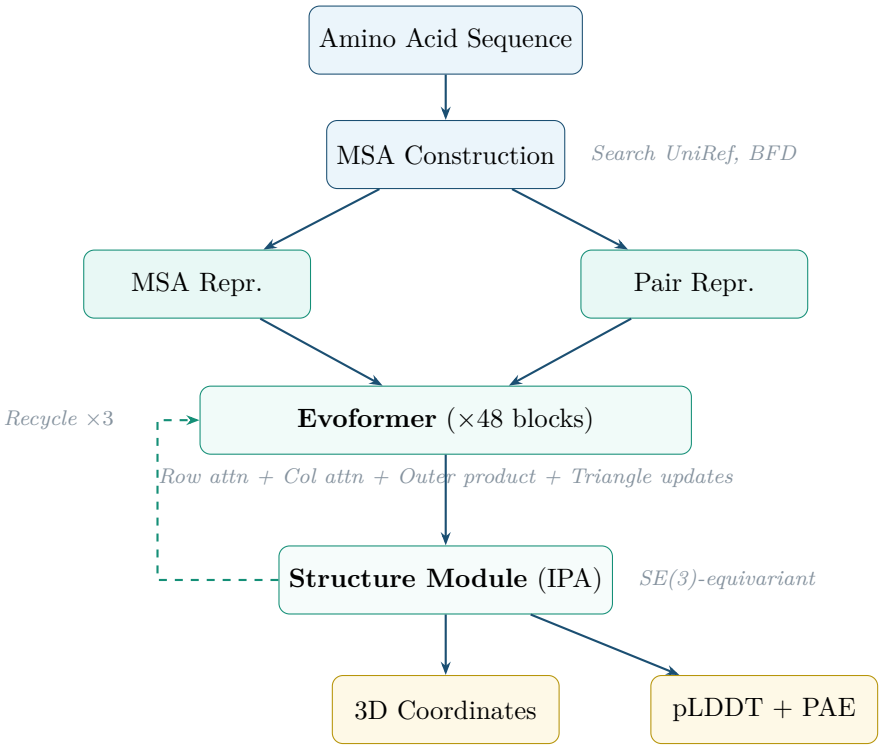


Figure 6.1: The AlphaFold2 pipeline. An amino acid sequence is used to construct a multiple sequence alignment (MSA). Two representations—MSA and pairwise—are jointly refined through 48 Evoformer blocks that exchange evolutionary and geometric information. The Structure Module converts pairwise features 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

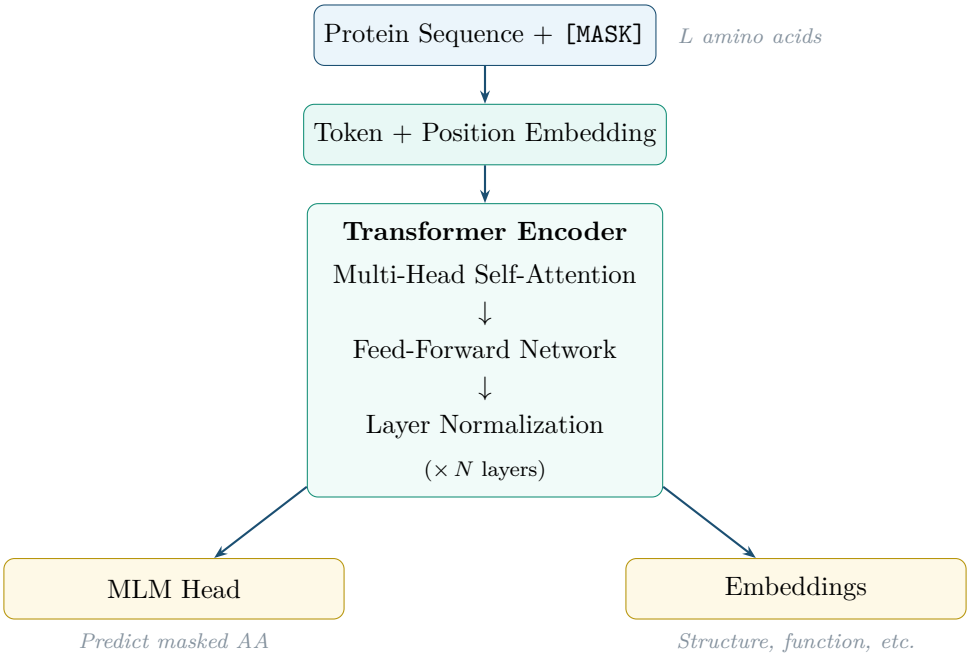


Figure 6.2: ESM-2 architecture overview. Protein sequences with masked positions are embedded and processed through N transformer encoder layers. The MLM head predicts masked amino acids (training). The learned embeddings (inference) encode structural and functional information useful for downstream 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 1) 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., 2021] adopts ***k*-mer tokenization**: the DNA sequence is converted to overlapping k-mers, typically with $k = 6$. For example:

$$\texttt{ATCGATCG} \longrightarrow \texttt{ATCGAT} \quad \texttt{TCGATC} \quad \texttt{CGATCG}$$

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., 2024] 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 1).

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 1).

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 1) 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 1 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 1.

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 1.

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., 2021, Dalla-Torre et al., 2024].
- **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. [2021]: DNABERT—pre-trained bidirectional encoder for DNA.
- Dalla-Torre et al. [2024]: 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.

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

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., 2017].

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.

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.

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.

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.

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.

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. *Nature Methods*, 2024. 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.

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.

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.

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, 2022.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

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. *Bioinformatics*, 37(15):2112–2120, 2021. 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*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Proceedings of ICLR*, 2015.

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.

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.

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.

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.

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.

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.

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.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2023.

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.

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, 2017.

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*, 2022.

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.

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.

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.