The Geometry of Logic: Stratification Induces Semantic Structure and Robust Reasoning

Cristina V. Lopes[1,*] **Yuangang Li**[1,*,‡] **Justin Tian Jin Chen**[2]
Alberto Krone-Martins[1] **Iris Ma**[1] **Md Rakib Hossain Misu**[1]

[1] University of California, Irvine [2] University of California, Davis

Abstract:

Transformer-based language models perform well on symbolic tasks, yet it remains unclear whether they learn generalizable rules or rely on statistical shortcuts. Mechanistic studies link algorithmic behavior to structured internal representations, motivating the hypothesis that robust reasoning benefits from separating values from the types that control their manipulation. Can making this separation an architectural primitive improve the learnability and generalization of logical mechanisms? We introduce **STRAT** (**ST**ratified **R**egisters **A**nd **T**ypes), which partitions the residual stream into orthogonal Data and Type subspaces and uses Type-based attention and gating to govern Data transformations. Controlled arithmetic ablations identify three failure modes associated with data-control interference: the Linear Trap, Gradient Wall, and Open Gate Trap. Mechanistic analysis reveals interpretable logical structure, and in arithmetic, STRAT reduces median OOD error 35-fold relative to a Transformer baseline. On each of 11 datasets spanning 10 tasks, STRAT outperforms the Transformer baseline in mean accuracy, by 26 percentage points on average, with both models trained from 10 base examples per dataset using identical task-specific augmentation where applicable. Under distribution shift, STRAT's mean accuracy drops by only 2.39 percentage points, compared with 11.75 for the Transformer.

Keywords: Model Architecture, Representation Learning, Logical Reasoning, Mechanistic Interpretability

Github Repo: https://github.com/crista/strat

Contact: yuanganl@uci.edu

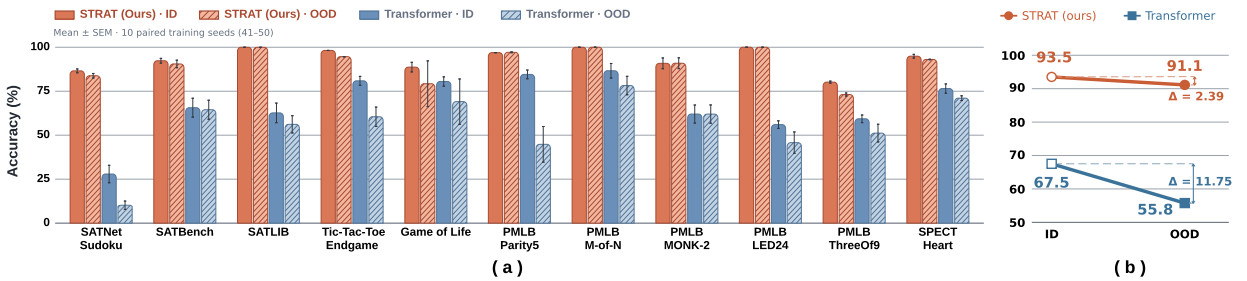

Figure 1: Mean accuracy across 11 datasets spanning 10 tasks, with both STRAT and the Transformer baseline trained from only 10 base examples per dataset: **(a)** ID and OOD accuracy for each dataset; **(b)** macro-averaged accuracy under distribution shift.

1 Introduction

Transformer-based models have demonstrated an impressive fluency in symbolic domains, generating code and solving arithmetic problems with high apparent fidelity (McLeish et al., 2024; Lee et al., 2024; Li et al., 2022; Cho et al., 2025). Yet, a fundamental question regarding their internal mechanics remains unresolved (Nikankin et al., 2025; Mirzadeh et al., 2025): Does the Transformer architecture truly reason, or does it merely approximate logic through massive statistical correlation?

Mechanistic interpretability research suggests that the answer may be "both," depending on the stage of training. Recent findings indicate that Transformer models can transition from rote memorization to genuine algorithmic execution, a phenomenon known as *grokking* (Power et al., 2022; Nanda et al., 2023). This transition often coincides with

*These authors contributed equally. ‡ Corresponding author.

the formation of discrete circuits, such as Induction Heads (Olsson et al., 2022), which implement copy-paste logic, and the emergence of internal world models that track logical state independent of surface statistics (Li et al., 2023).

A unifying hypothesis for these phenomena is the *Linear Representation Hypothesis* (Elhage et al., 2022; Park et al., 2024), which posits that models represent distinct concepts as orthogonal directions in activation space. We extend this view to put forward the hypothesis that robust reasoning relies on a specific geometric configuration: the **stratification** of the embedding space into the basic building blocks of formal logic such as *Values* (the variable content being manipulated) and *Types* (the control signals governing the values). In standard Transformers, this separation must be learned against the gradient noise of high-variance data, potentially explaining the massive data requirements and abrupt phase transitions observed in training.

Taking a constructive approach, in this work, we ask: **What if this stratification were an architectural primitive rather than an emergent property?**

In order to explore this question, we step back from large transformer-based multi-purpose models, and focus on small ones whose purpose is to perform relatively simple but highly logical tasks like arithmetic and Sudoku. Our goal is to study how logic can emerge in neural models.

We introduce **STRAT** (**STrat**ified **R**egisters **A**nd **T**ypes), an architecture that explicitly partitions the residual stream into mutually exclusive subspaces for Data and Types. By structurally enforcing the "geometric hygiene," we observe a radical shift in learning dynamics. Specifically, our contributions are as follows:

1. **The Failure Taxonomy.** Through a rigorous ablation study, we demonstrate that Transformers fail to learn the tasks not due to insufficient scale or optimization noise, but due to **Data-Control Interference**. We identify three distinct failure modes: the *Linear Trap*, the *Gradient Wall*, and the *Open Gate Trap*.
2. **Stratified Attention.** We introduce a dual-stratum architecture that enforces orthogonality between Data (Values) and Control (Types), creating a protected "Gradient Highway" that lets the model learn discrete logical gates despite high-magnitude numerical payloads.
3. **Topological Locking.** We demonstrate that STRAT converges to the correct logical mechanism from only $N = 10$ training examples. While baselines default to linear interpolation (OOD Error > 7000), STRAT locks into the correct geometric topology, achieving a $35\times$ reduction in median error and generalizing to out-of-distribution ranges.
4. **Empirical Evaluation.** We evaluate STRAT on 11 datasets spanning 10 tasks, achieving a mean accuracy advantage of 26 percentage points over the Transformer baseline. Under distribution shift, STRAT maintains stable performance, with mean accuracy dropping by only 2.39 percentage points, compared with 11.75 for the Transformer (see Figure 1)

2 The Model

In this work, we use a standard Transformer architecture optimized for algorithmic stability. We employ widely adopted modern components – specifically relative attention and gated activations – which, as we will show, provide the necessary primitives for symbolic manipulation.

2.1 Representation: The Embedding Space

Tokens $S = (t_1, \ldots, t_T)$ are mapped to $\mathbb{R}^d$ via an embedding matrix E. We forgo absolute positional embeddings, relying strictly on relative biases in the attention layer to handle sequence order.

2.2 Binding: The Attention Mechanism

The Attention Layer solves the *Binding Problem*: associating operands (e.g., numbers) with their governing operators. We employ single-head self-attention with a linear distance penalty (ALiBi) to enforce locality:

$$A = \text{softmax}\left(\frac{(H_0 W_Q)(H_0 W_K)^T}{\sqrt{d_k}} + M + B\right) \tag{1}$$

where $B_{i,j} = -\lambda|i - j|$ decays attention with distance. This shielding function provides a soft window, ensuring that tokens preferentially fetch state from their nearest neighbors while suppressing distant distractors. The output is computed as $H_{attn} = H_0 + A(H_0 W_V)$.

2.3 Conditional Execution: The Gated FFN

The Feed-Forward Network utilizes a Gated Linear Unit (GLU) with a **Sigmoid** activation. This choice is structural: unlike ReLU or GELU, the bounded $(0, 1)$ range of the sigmoid allows the network to implement discrete boolean

logic (Open/Closed).

$$H_{out} = H_{attn} + (\sigma(H_{attn}W_{gate} + b_{gate}) \odot (H_{attn}W_{val} + b_{val})) \tag{2}$$

Here, the *Gate* projection acts as a differentiable switch conditioning the flow of the *Value* payload.

2.4 Readout: Global Accumulation

For arithmetic tasks, the final result is often an aggregation of local updates. We therefore use global sum pooling followed by a linear decoding: $y = \sum_{t=1}^{T}(H_{out}[t]W_{out})$. This pooling mechanism is permutation-invariant with respect to the sequence positions, forcing the model to solve the task by modifying the token states themselves rather than relying on a "read" head at the last position.

3 The Geometric ALU: Existence Proof

We first establish that the architecture is capable of expressing symbolic reasoning by hand-constructing weights that instantiate an Arithmetic Logic Unit (ALU) using a single Transformer block. The geometric algorithm unfolds in four steps: 1. **Representation:** Tokens are mapped to a 6-dimensional orthogonal basis (Table 1), separating numerical payloads (D_0) from boolean flags ($D_1 - D_4$). 2. **Binding:** Numbers query preceding tokens to "acquire" their governing operator via attention. 3. **Gating:** A logic gate activates only for subtracted numbers, acting as a differentiable switch. 4. **Execution:** Numbers that flow through the negative gate are "copied" to a scratchpad dimension (D_5). Effectively, numbers to be subtracted are "Copy-and-Pasted" from D_0 to D_5 – that is, in a nutshell, our geometric algorithm. The full implementation details, including weight matrices, are provided in **Appendix A**.

Table 1: Dimension Assignments in the Residual Stream

Index	Content	Functional Role
D0	`Value`	Numerical magnitude (Payload)
D1–D4	`Flags`	Type Indicators (IsNum, IsOp, etc.)
D5	`Accu`	Conditional Storage

4 The Unlearnable Geometry of Logic

While Section 3 proves the Geometric ALU is *capable* of symbolic reasoning, is this solution *learnable*? We conducted an exhaustive ablation study to see if standard gradient descent could discover this topology from scratch.

4.1 Experimental Protocol

We trained 12 configurations of the 6-dimensional architecture on the signed-sum task for 2,000 steps (128k samples total), in batches of 64, 50 runs (seeds) per config. Configuration details are provided in Appx. B. **Metrics:** We report the **25$^{\text{th}}$-percentile**, **Median**, and **75$^{\text{th}}$-percentile** of the Mean Absolute Error (MAE) to capture the variance between "trapped" and overfitted runs.

In-Distribution (ID): $x \in [1, 100]$. Baselines: *Input Mean Absolute Deviation (MAD)* (≈ 25.0, guessing $y = x$) vs. *Target MAD* (≈ 90.1, guessing mean). **Out-of-Distribution (OOD):** $x \in [2000, 5000]$. Distinguishes logic from curve-fitting.

4.2 Results: A Taxonomy of Failure

Despite the existence of a perfect solution (0.00 error), **none** of the 600 learned models converged to the geometric mechanism. Table 2 reveals that every attempt fell into one of three topological traps.

The *Linear Trap* reflects value-based shortcut fitting; the *Gradient Wall* arises when saturated gates block learning; and the *Open Gate Trap* leaves gates acting as linear pass-throughs. Even low ID error does not imply OOD generalization, supporting the need to isolate control signals from numerical payloads (Appx. B.2).

4.3 Conclusion: Simplicity Bias is Structural

This ablation study provides an answer to the structural critique. **Normalization ($\mathbf{A}_{ln}$)** failed to stabilize the features (ID MAE 16.9). **Scaling ($\mathbf{A}_s$)** failed to balance the gradients. **Curriculum ($\mathbf{A}_{curr}$)** failed to guide the optimization.

The robustness of the failure confirms that **Simplicity Bias** in this domain is not a transient optimization pathology, but a structural inevitability of Unityped Transformers. In the optimization landscape, the Value stream (x) offers a shortcut that is linearly correlated with the target ($|y| \approx |x|$). Deep networks preferentially learn features with simple, linear error surfaces (the Value shortcut) while starving complex, non-linear features (the Logic gate). Without brute-force scale, the logic gate cannot be *trained* to ignore the data; it must be *architecturally isolated* from it.

5 The STRAT Architecture

Our diagnosis of the baseline failures reveals a fatal pathology: **Data-Control Interference**. Standard Transformers conflate *Data* (variable payloads) and *Control* (routing signals) into a single, unityped vector space. To resolve this,

Table 2: The Failure Taxonomy (N=50 runs/config). We report 25%-ile / Median / 75%-ile errors.

			ID MAE (Train)			**OOD MAE** (Test)		
Cfg	**Strategy**	**Hypothesis**	**25%**	**Med**	**75%**	**Med**	**75%**	**Diagnosis / Mechanism**
Ref	Hand-Coded	*Existence Proof*	**0.0**	**0.0**	0.0	**0**	0	**Geometric Logic**
A	Simplified Xformer	Standard Training	3.5	15.5	38.2	5,977	10,809	**Linear Trap**
$\mathbf{A}_s$	A Scaled	Scale Invariance	97.1	98.7	100.1	6,202	6,265	**Linear Trap**
$\mathbf{A}_{ln}$	LayerNorm	Internal Stability	9.6	16.9	25.9	8,378	9,856	**Feature Collapse**
$\mathbf{A}_{curr}$	Curriculum	Complexity Sched.	212.2	600.3	1736.1	2306	2,682	**Linear Trap**
$\mathbf{B}_{fl}$	Frozen Logic	Sharp Gates	25.9	58.7	69.2	5,949	6,715	**Gradient Wall**
$\mathbf{B}_{fa}$	Frozen Att	Perfect Routing	4.9	6.2	8.7	11,817	13,060	**Magnitude Leak**
$\mathbf{B}_{ortho}$	Regularized	Orthogonality	2.3	9.8	35.2	9,750	12,088	**Linear Trap**
$\mathbf{C}_g$	Learned Gain	Self-Amplification	4.9	13.5	36.1	7,997	12,036	**Gradient Wall**
$\mathbf{C}_{sb}$	Signal Boost	Fixed Gain (γ)	10.8	45.5	49.3	5,613	7,228	**Gradient Wall**
$\mathbf{B}_c$	Reg + Boost	Synergy	12.7	46.5	48.2	5,611	9,677	**Gradient Wall**
$\mathbf{C}_b$	Balanced	Operating Point	1.2	1.8	2.9	11,916	12,221	**Operating Point Overfit**
$\mathbf{C}_{bi}$	Bias Init	Gradient Flow	3.1	12.7	37.4	8,471	11,815	**Open gate trap**

we propose the **STRAT** architecture (**STrat**ified **R**egisters **A**nd **T**ypes). We explicitly partition the embedding space $\mathbb{R}^{d_{model}}$ into orthogonal subspaces that never interact via linear mixing.

5.1 Input Representation: Scalar Injection

Standard Transformers learn dense embeddings for all tokens. STRAT employs a hybrid embedding scheme that enforces stratification at the input level. Let the embedding space be partitioned into indices $I_{val} \subset \{1..d\}$ and $I_{type} \subset \{1..d\}$.

1. The Value Stratum (V): For numerical tokens, the raw scalar magnitude $v \in \mathbb{R}$ is injected directly into a reserved dimension (D_0). This input embedding is **non-learnable** (fixed identity), ensuring the model perceives the exact quantity v. While the input is fixed, the *transformations* applied to it (W_{val}) are fully learnable.

2. The Type Stratum (T): Discrete tokens (operators) are mapped to embeddings restricted to the Type dimensions ($D_1..D_4$). We explore two initialization strategies: **Fixed Types (Oracle):** We inject hand-coded orthogonal vectors (e.g., e_1 for Plus, e_2 for Minus) to test the routing mechanism in isolation (Sections 5.2–5.3). **Latent Types (Learned):** We initialize with random noise to test if the model can autonomously discover the semantic structure (Section 5.4).

5.2 STRAT-Lite: Disentangled Attention and Routing

To validate the core interaction between Data (V) and Types (T), we begin by analyzing a simplified instance, **STRAT-Lite**. Figure 2 provides an overview of the architecture, using $12 + 3 - 5$ as an illustrative input. The block update $x^{(l+1)} = \text{Block}(x^{(l)})$ modifies the standard Transformer block in two critical ways:

5.2.1 Phase 1: Disentangled Control Attention

We restrict the Attention mechanism to operate exclusively on the Type Stratum (T). Tokens update their internal state based on the *types* of their neighbors without the numerical values (V) corrupting the routing logic.

$$\begin{aligned} A &= \text{Softmax}\left(\frac{Q(x_T)K(x_T)^\top}{\sqrt{d_{type}}} + \text{ALiBi}\right) \\ \Delta x_T &= A \cdot V(x_T) \\ x_T^{(l)} &\leftarrow x_T^{(l)} + \Delta x_T \quad \text{(Residual Update)} \end{aligned} \tag{3}$$

Note: This update is applied *only* to the Type indices. The Value indices remain unchanged during this phase ($x_V^{(l)} \leftarrow x_V^{(l)}$).

5.2.2 Phase 2: The Topology-Aware GLU Router

The Feed-Forward Network is replaced by a topology-aware Gated Linear Unit (GLU). The gating branch observes only the (now updated) Type Stratum to determine *when* to act, while the value branch observes only the Value Stratum

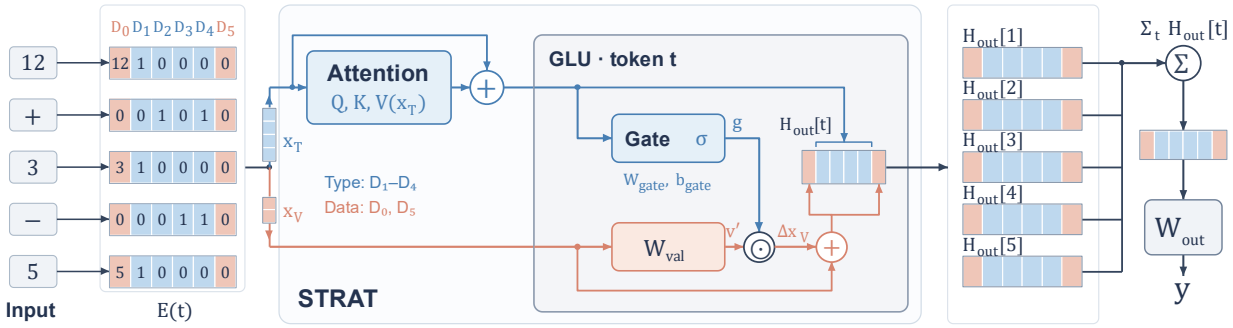

Figure 2: Overview of the STRAT-Lite architecture with an arithmetic example ($12 + 3 - 5$).

to determine *what* to transmit.

$$\begin{aligned} g &= \sigma(W_{gate} x_T + b_{gate}) && \text{(Control Logic)} \\ v' &= W_{val} x_V && \text{(Data Transformation)} \\ \Delta x_V &= g \odot v' && \text{(Gated Update)} \\ x_V^{(l+1)} &\leftarrow x_V^{(l)} + \Delta x_V \end{aligned} \tag{4}$$

This additive update allows the model to perform *destructive interference* (subtraction) on the original signal: $x_{new} = x_{old} + \Delta x$.

Normalization Strategy: Unlike Transformers, STRAT-Lite **omits Layer Normalization** to prevent the high-variance Value signal from suppressing the Type signal via variance normalization.

5.3 Validation and Mechanism

We evaluated STRAT-Lite on the signed-sum task using fixed oracle types, and under the same experimental protocol used for the ablation study (Section 4.1). STRAT-Lite achieves an excellent generalization (25%-ile MAE 25.8; Median MAE 40.4 on OOD data), whereas standard Transformers fail catastrophically (MAE > 1000, see Table 2).

Forensic Analysis: The Two Attractors To understand *how* the model implements arithmetic, we inspected the learned weights of 50 successful seeds. We found that the architectural constraints force the model into one of two distinct topological "attractors," both implementing a valid "Cut-and-Paste" logic.

Attractor A (The Standard Topology): In some of seeds, the model discovers the direct subtraction mechanism: **Gate:** The router learns a positive weight for the "Minus" type ($w_{gate} > 0$), opening the gate only when subtraction is required. **Value:** The value weight is learned as negative ($w_{val} \approx -1$). **Result:** The update $\Delta x = 1.0 \cdot (-x_{in})$ is added to the residual, effectively moving $-x_{in}$ to the accumulator.

Attractor B (The Mirror Topology): In the other seeds, the model discovers an algebraically equivalent but geometrically inverted solution: **Gate:** The router learns a *negative* weight for the "Minus" type ($w_{gate} < 0$), making the gate "normally open" but closed for Minus. **Value:** The value weight is inverted ($w_{val} \approx 1$). **Result:** The model effectively adds x_{in} by default and learns to *suppress* the addition when the type is Minus.

Appx. C provides representative learned weights and their analysis. The existence of these discrete, interpretable attractors confirms that STRAT does not merely fit a curve; it descends into a structured semantic minimum defined by the topology.

5.4 Latent Type Discovery (Learned Types)

Does this architecture require hand-engineered Type definitions? To test this, we ignored our fixed embeddings and initialized the Type Stratum (T) randomly ($\mathcal{N}(0, 0.02)$).

5.4.1 Arithmetic

We trained 10 models on the signed-sum task without explicit tags for Plus or Minus. They converged to the same MAE as the fixed-type models. As shown in Table 3, the model discovered the necessary logic by forcing the learned vectors for Addition and Subtraction to become antipodal ($v_+ \approx -v_-$), maximizing the decision margin for the GLU.

5.4.2 Semantic Clustering (The "Grocery List")

We further tested the model on a **Selective Summation** task with 15 distinct Item IDs drawn from three latent categories (Target, Distractor A, Distractor B). **Results:** The model achieved 99.6% OOD accuracy. Geometric analysis

Table 3: Learned Type Embeddings. Note the antipodal structure ($v_+ \approx -v_-$) between Plus and Minus.

Token	D1	D2	D3	D4
Literal	0.55	−0.25	0.06	0.06
Plus	0.50	−0.59	−0.67	−0.51
Minus	−0.80	0.81	0.83	0.79

Table 4: Learned Centroids. Note the geometric collapse of the two Distractor categories.

Category	D1	D2	D3	D4
Target (Cat 0)	−0.15	−0.18	−0.19	0.27
Distractor A (Cat 1)	0.27	0.29	0.31	−0.44
Distractor B (Cat 2)	0.28	0.29	0.30	−0.44

(Table 4) reveals that the embeddings for the two Distractor categories collapsed into a shared manifold (Distance < 0.01), confirming that STRAT autonomously clustered tokens based on their functional role.

5.4.3 PCA

Figure 3 shows the visualization of the Principal Component Analysis (PCA) applied to the learned Type Stratum ($D_1 - D_4$) after training. The geometry reveals two distinct mechanisms of semantic emergence:

Panel A (Arithmetic): The model spontaneously discovers an **antipodal arrangement** for the operator tokens. The Plus (Blue) and Minus (Red) vectors are positioned at opposite ends of the principal axis ($v_+ \approx -v_-$), maximizing the orthogonality of their respective decision boundaries. The Literal token (Grey) remains in a neutral position near the origin, ensuring that raw numbers do not inherently trigger logic gates. This configuration confirms that the model implements signed arithmetic by learning diametrically opposed geometric directions for inverse operations.

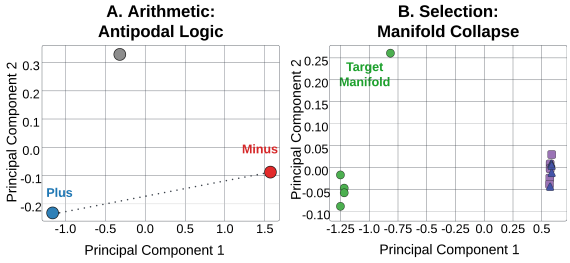

Figure 3: Antipodal discovery. Left: arithmetic. Blue circle: plus; red circle: minus; grey circle: numbers. Right: selective sum. Green cuircles: target category; purple squares: distractor category A; blue triangles: distractor category B.

Panel B (Selection): The model exhibits **manifold collapse**. Although Distractor A (Squares) and Distractor B (Triangles) represent distinct item IDs, the model maps them to an overlapping cluster in the latent space (Purple). In contrast, the Target items (Green) form a distinct, separated manifold. This proves that the model actively constructed a semantic Null Space, treating all non-target items as functionally identical regardless of their surface-level token identity.

6 Topological Locking: Logic from N=10

If a model possesses the correct structural prior, it should not require thousands of examples to approximate the solution. To test the limits of the architecture, we performed an extreme sample efficiency sweep.

6.1 The Hyper-Overfitting Regime

We instantiated **1,600 distinct learners** (400 random seeds per configuration). Each learner was provided with a theoretically minimal training dataset of size **N=10**: ten randomly sampled arithmetic sequences (e.g., "$5 + 3 - 2$") with values $x \in [0, 100]$.

We subjected the learners to a **Hyper-Overfitting Regime**: each model was trained on its tiny dataset for **2,000 epochs**, forcing it to memorize the training set perfectly (MSE ≈ 0). To test robustness, we then evaluated on Out-of-Distribution sequences ($x \in [2000, 5000]$).

6.2 Experimental Setup

As baseline, we used Config A from the ablation study, and tested 4 variants of STRAT-Lite to determine the topology required for convergence: ***(i)*** **STRAT-Lite (2D):** Compresses the Type Subspace to $\mathbb{R}^2$, forcing the model to find a planar separation between operators. ***(ii)*** **STRAT-Lite (2D-Dual):** Adds a second Attention Head to the 2D bottleneck, testing whether parallel routing mechanisms facilitate convergence. ***(iii)*** **STRAT-Lite (3D):** Expands the subspace to $\mathbb{R}^3$. Since there are three distinct token types (Literal, Plus, Minus), this dimension allows for an orthogonal simplex representation. ***(iv)*** **STRAT-Lite (4D):** The high-capacity variant ($\mathbb{R}^4$) used in previous sections.

To strictly test the architectural inductive bias, we handicapped the STRAT models by forcing them to operate in the latent regime (requiring spontaneous type discovery). In contrast, we provided the Baseline (Config A) with explicit oracle supervision, injecting perfect one-hot type encodings directly into its input. To evaluate whether the model learned the subtraction rule versus merely fitting the training data, we define two accuracy thresholds based on the OOD MAE: **Logic Accuracy (MAE $<$ 100):** This measures the success of the routing mechanism. Given the magnitude of OOD inputs ($x \approx 3,500$), a failure to switch signs would result in massive errors (MAE $> 3,000$). An MAE below 100 proves the logic is correct, even if the gates are not perfectly saturated (the "Sigmoid Tax"). **Strict**

Table 5: Topological Locking Results ($N = 10$, 400 Seeds). **Strict Acc** (< 20) denotes perfect circuit execution. **Logic Acc** (< 100) denotes functional gating with Sigmoid tax. The Baseline fails (0% acc). STRAT achieves $\approx 31\%$ logic locking, with the top 10% achieving MAE ≈ 33.

	OOD MAE Distribution (Lower is Better)			**Success Rates** (Higher is Better)	
Model	**10%-ile**	**25%-ile**	**Median**	**Strict (<20)**	**Logic (<100)**
Baseline (Config A)	5,072	6,268	9,046	0%	0%
STRAT Flat (2D/1H)	33	80	293	5%	31%
STRAT Flat MH (2D/2H)	38	97	314	2%	27%
STRAT Compact (3D/1H)	34	77	266	3%	31%
STRAT Std (4D/1H)	**33**	**77**	**258**	4%	**31%**

Accuracy (MAE < 20): This measures the saturation of the Control Signal. It requires that the logic gates are driven to hard limits ($\sigma(x) \approx 1.0$), ensuring the signal passes with $> 99\%$ fidelity.

6.3 Result Analysis: The Linear Trap vs. Structure

The results in Table 5 reveal a fundamental dichotomy in learning dynamics: **The Baseline Failure (Linear Trap):** The simplified baseline Transformer (Config A) minimizes training loss by learning a linear mapping $f(x) \approx Wx + b$ that passes through the 10 training points. The failure is absolute: even the top 10% of Baseline runs yield an OOD error of **5,072**, and the Strict/Logic accuracy is exactly 0%. This confirms that without stratification, the probability of spontaneously discovering a boolean gate in the presence of high-magnitude data is effectively zero. **The STRAT Success (Topological Locking):** In contrast, STRAT consistently achieves a $\approx 31\%$ logic success rate across variants. The metrics reveal a tiered success structure: ***(i)* Strict Locking ($\approx 4\%$):** A small subset of models achieve MAE < 20, indicating they found the optimal "Cut-and-Paste" circuit with hard saturation. ***(ii)* Logic Locking ($\approx 31\%$):** A significant portion achieve MAE < 100. These models correctly implement the subtraction logic but pay a small "Sigmoid Tax" due to softer gate saturation. ***(iii)* Robust Precision:** The **10%-ile MAE of 33** confirms that the best runs are extremely accurate.

The similarity between variants (Flat vs. Std) suggests that locking is robust to dimensionality: as long as at least one dimension is orthogonal to the value stream (the Type Stratum), the Gradient Highway exists. The higher Median MAE (≈ 260) reflects the $N = 10$ constraint: when the curriculum lacks operator diversity, the model cannot learn to close the gate. However, when the curriculum is valid, STRAT locks; the Baseline never does.

We also compare STRAT with Neural Accumulator (NAC) and Neural Arithmetic Logic Units (NALU) (Trask et al., 2018) to assess generalization against specialized arithmetic methods. Under the same training budget, STRAT Std achieves a median OOD MAE of 258, more than 22-fold lower than NAC (5,819) and NALU (5,770). Appx. F provides the full comparison and implementation details.

6.4 Why STRAT Succeeds: The Gradient Highway

STRAT enables this locking via two mechanisms: ***(i)* Orthogonality:** The Value Stratum is blocked from the Attention mechanism. The router's gradients depend *only* on the Type embeddings, creating a noise-free "Gradient Highway" for learning logic. ***(ii)* Constraint:** The model cannot mix D_0 (Value) and D_1 (Type). To reduce loss, it is forced to use the only available degree of freedom: the GLU gate. This effectively deletes the linear interpolation solution from the hypothesis space.

7 STRAT Evaluation Across 10 Diverse Tasks

Using predefined task-specific Type encodings, we compare STRAT with the Transformer on 11 datasets covering 10 tasks (Table 6). In every comparison, STRAT and the Transformer receive the same observable inputs and training data. Within each dataset, we match network depth, attention-head count, feed-forward hidden widths, optimizer, learning rate, training duration, loss function, and checkpoint-selection rule. Both train on 10 base examples per dataset and the same augmentations, if any, derived from them. STRAT uses fewer trainable parameters than the Transformer baseline in all 11 dataset evaluations (Table 11). We report mean accuracy and SEM over 10 paired runs (seeds 41–50), using example-level accuracy for single-label tasks and exact-board accuracy for SATNet Sudoku and Game of Life. Appx. D details the task-specific augmentation procedures, the Type/Data input encodings supplied to both models, and their respective configurations.

STRAT achieves higher mean accuracy than the Transformer in all 11 evaluations, with a macro-average advantage of 26 percentage points (Table 6). These results support our hypothesis that explicit Data–Type stratification provides an effective inductive bias for generalization from limited examples, with benefits extending beyond arithmetic to diverse task families.

Table 6: We report accuracy (%) across 11 dataset evaluations as mean ± standard error of the mean (SEM) over 10 seeds. The difference columns use STRAT minus Transformer: ID accuracy (Δ ID Acc.),and OOD accuracy (Δ OOD Acc.). The OOD Acc. columns include the OOD performance and the difference relative to the ID accuracy in parentheses (green for increase, red for decrease).

Task	Dataset	STRAT (Ours)		Transformer		Δ ID Acc.	Δ OOD Acc.
		ID Acc. (%)	OOD Acc. (%)	ID Acc. (%)	OOD Acc. (%)		
Constraint satisfaction Sudoku solving	SATNet Sudoku (Wang et al., 2019)	**86.58 ± 1.08**	**83.80 ± 1.30** (-2.78)	27.92 ± 4.97	10.18 ± 2.36 (-17.74)	**+58.66 ± 5.22**	**+73.62 ± 3.24**
Formula satisfaction classification Given-assignment satisfaction	SATBench (Wei et al., 2025)	**92.33 ± 1.36**	**90.50 ± 2.15** (-1.83)	65.64 ± 5.38	64.48 ± 5.40 (-1.16)	**+26.69 ± 5.58**	**+26.02 ± 5.54**
Formula satisfaction classification Given-assignment satisfaction	SATLIB (Hoos & Stützle, 2000)	**99.98 ± 0.02**	**99.96 ± 0.04** (-0.02)	62.67 ± 5.58	56.18 ± 4.91 (-6.49)	**+37.31 ± 5.58**	**+43.78 ± 4.91**
Board classification Winner detection	Tic-Tac-Toe Endgame (Aha, 1991)	**98.19 ± 0.00**	**94.64 ± 0.00** (-3.54)	80.91 ± 2.53	60.49 ± 5.47 (-20.42)	**+17.28 ± 2.53**	**+34.15 ± 5.47**
State-transition prediction Next-state prediction	Game of Life (Strandgaard, 2023)	**88.71 ± 2.72**	**79.26 ± 12.99** (-9.45)	80.54 ± 2.64	69.08 ± 12.89 (-11.46)	**+8.16 ± 3.61**	**+10.18 ± 17.94**
Parity concept learning Five-bit parity	PMLB Parity5 (Olson et al., 2017)	**96.94 ± 0.06**	**97.12 ± 0.25** (+0.19)	84.56 ± 2.48	44.81 ± 10.13 (-39.75)	**+12.38 ± 2.49**	**+52.31 ± 10.19**
Threshold concept learning M-of-N rule	PMLB M-of-N (Olson et al., 2017)	**100.00 ± 0.00**	**100.00 ± 0.00** (0.00)	86.65 ± 4.14	78.23 ± 5.24 (-8.42)	**+13.35 ± 4.14**	**+21.77 ± 5.24**
Categorical rule induction Exact-two classification	PMLB MONK-2 (Olson et al., 2017)	**90.86 ± 3.04**	**90.92 ± 3.07** (+0.06)	62.01 ± 5.13	61.99 ± 5.22 (-0.03)	**+28.84 ± 7.17**	**+28.93 ± 7.29**
Density classification LED segment density	PMLB LED24 (Olson et al., 2017)	**100.00 ± 0.00**	**100.00 ± 0.00** (0.00)	56.01 ± 2.14	45.76 ± 6.06 (-10.25)	**+43.99 ± 2.14**	**+54.24 ± 6.06**
Majority classification Nine-bit majority	PMLB ThreeOf9 (Olson et al., 2017)	**80.18 ± 0.57**	**73.12 ± 0.88** (-7.05)	59.30 ± 2.17	51.18 ± 5.03 (-8.12)	**+20.88 ± 2.14**	**+21.94 ± 5.39**
Density classification SPECT feature density	SPECT Heart (Cios et al., 2001)	**94.91 ± 1.05**	**93.05 ± 0.15** (-1.86)	76.50 ± 2.63	71.04 ± 1.35 (-5.46)	**+18.40 ± 2.37**	**+22.01 ± 1.34**
Average		**93.51**	**91.13 (-2.39)**	**67.52**	**55.77 (-11.75)**	**+26.00**	**+35.36**

Testing under Distribution Shift. To test whether the gains above reflect logic learning rather than dataset-specific shortcut fitting, we evaluate both models without retraining on identical OOD sets that preserve the task rules. For example, Sudoku puzzles have more unevenly distributed clues, while SATBench clause sequences are extended to length 100 by repeating existing clauses. Appx. D provides the complete construction procedures. STRAT achieves higher mean accuracy in all 11 OOD evaluations. Its macro-average accuracy drops by only 2.39 percentage points, compared with 11.75 for the Transformer. This robustness to structural shifts supports the hypothesis that STRAT learns transferable task logic with reduced reliance on dataset-specific shortcuts.

Moreover, on Sudoku, Tic-Tac-Toe, Game of Life, M-of-N, and MONK-2, all of which retain their original prediction targets, STRAT achieves performance comparable to that of current state-of-the-art methods while using only 10 base training examples per dataset; see Appx. E for details.

8 Discussion

8.1 The Geometry of Efficiency (Interpreting N=10)

Perhaps the most striking result of this study is the phenomenon of *Topological Locking* (Section 6), where the model converges to the correct logical rule from as few as $N = 10$ examples. Standard deep learning theory suggests that models require massive data to "grok" abstract rules (Power et al., 2022). STRAT avoids this by altering the geometry of the hypothesis space. By enforcing stratification, we effectively delete the shortcut solution (linear curve-fitting) from the search space. The optimizer is not learning the logic in the traditional sense of carving a manifold; rather, it is sliding into the only remaining minima allowed by the topology. This suggests that *Sample Efficiency is a derivative of Topological Constraints*.

8.2 Emergent Semantics via Null Spaces

In Section 5.4, STRAT did not merely route data based on pre-existing types; it actively constructed a type system. By collapsing the "Distractor" items into a shared null-space manifold (distance < 0.01), the model discovered the concept of an "Ignore" class. This reverses the standard neuro-symbolic paradigm. Rather than grafting symbolic modules onto neural networks, STRAT demonstrates that we can *induce* semantic structure simply by restricting information flow. The Type Subspace acts as a pressure vessel: shielded from the Value stream, gradient descent is forced to use it solely for low-variance abstractions.

8.3 Value-Invariance and Safety

A common assumption is that performance requires opacity. STRAT challenges this. By design, it guarantees **Value-Invariance**: the logic gate is mathematically incapable of being "tricked" by the magnitude of the payload. This has profound implications for AI Safety. Standard Transformer neurons are polysemantic; a large number (e.g., 5000) might accidentally trigger a logic neuron simply because it pushes the activation sum over a threshold. STRAT reduces the surface area for such failures to zero. We envision architectures as hybrid systems: standard *System 1* layers for perceptual processing feeding into *System 2* STRAT blocks for rigorous reasoning (Bengio, 2019).

8.4 The "Sigmoid Tax"

While STRAT eliminates semantic interference, it remains an analog approximation of digital logic. The persistent error in successful models (MAE $\approx$ 25) represents the **Sigmoid Tax**: the asymptotic cost of simulating discrete switching with continuous functions. Future work will explore techniques to anneal these activation functions toward hard step functions (Heaviside) during the final stages of training, potentially crystallizing the learned analog circuit into a discrete, zero-error algorithm.

9 Related Work

9.1 Transformers on Algorithmic Tasks

Despite their dominance in language, Transformers struggle with precise arithmetic and OOD extrapolation (Saxton et al., 2019; Lake & Baroni, 2018). Prior work attributes this to positional failures, proposing Relative or Randomized Positional Encodings (RPE) to force locality (Shaw et al., 2018; Ruoss et al., 2023; Nye et al., 2021). However, our baseline results (Config A) show that even with perfect relative encodings (ALiBi), standard Transformers fail. We identify **Data-Control Interference** as the distinct failure mode: a pathology where high-variance payloads drown out low-variance control signals. This differs from the "grokking" phenomenon (Power et al., 2022), as we show it is an architectural, not merely observational, barrier.

9.2 Mechanistic Interpretability and Negative Heads

Our forensic analysis of the Cut-and-Paste logic (Section 5.3) contributes to mechanistic interpretability (Elhage et al., 2021; Olah et al., 2020). The 'Cut' mechanism, routing a negative copy of a value to cancel it out, bears a striking resemblance to the Copy-Suppression Heads discovered in GPT-2 (McDougall et al., 2023). In LLMs, these heads suppress the logits of previous tokens to correct for naive copying. STRAT suggests that this suppression logic is not just a statistical correction but a fundamental geometric primitive for algorithms over data in residual streams.

9.3 Type Theory and Parametricity

From the perspective of Programming Language Theory (PLT), the standard Transformer operates on a flat, *unityped* vector space (Pierce, 2002). Dense linear layers perform a *destructive mixing* operation ($y = \mathbf{w}^T(\mathbf{x}_{val} + \mathbf{x}_{type})$), collapsing the distinction between the **Term Level** (Values) and the **Type Level**. STRAT resolves this by enforcing **Relational Parametricity** (Reynolds, 1983; Wadler, 1989). By isolating the *Value Stratum* as an opaque container, the logic becomes parametric with respect to the values: it can route the data, but it is architecturally forbidden from inspecting the *bits* of the data itself. This mirrors the design of NALU (Trask et al., 2018; Madsen & Johansen, 2020) but achieves the effect using Transformer components rather than specialized ones.

10 Conclusion

We hypothesize that the reasoning capabilities attributed to scale are the result of implicit **Geometric Stratification**. By explicitly engineering this structure in STRAT, we demonstrated that core reasoning mechanisms can be achieved with extreme efficiency, rather than brute force. Our experiments revealed:

1. **Geometry is Algorithm:** The model implemented task logic not by memorization, but by discovering specific topologies (e.g., antipodal arrangements).
2. **Constraints Enable Generalization:** By restricting the search space to orthogonal strata, the model generalized from just $N = 10$ examples.
3. **Orthogonality Enables Logic:** A single set of weights successfully multiplexed contradictory programs, provided their execution contexts were geometrically orthogonal.

We conclude that logic is not a symbolic rule set superimposed on a network, but the natural consequence of a stratified vector space. Future work should investigate whether the "ghosts" of these strata can be detected within the latent spaces of frontier LLMs.

AI use statement

We used AI assistants to check grammar, polish our writing, draft sections of the paper, retrieve relevant literature, and assist with routine coding tasks. We reviewed all AI-assisted work, revised the text as needed, verified the cited sources for accuracy and relevance, and checked the code for correctness. We take full responsibility for the final content of this work.

Ethics statement

This work studies architectural inductive biases for rule learning using synthetic tasks and publicly available benchmark datasets. We do not recruit human participants, collect new personal data, or use personally identifiable information. We do not anticipate significant ethical concerns arising from these experiments.

Author Contributions

Justin Tian Jin Chen, Alberto Krone-Martins, Iris Ma, and Md Rakib Hossain Misu are listed in alphabetical order by surname.

REFERENCES

David Aha. Tic-Tac-Toe Endgame. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C5688J.

Tashin Ahmed and Q. Tyrell Davis. It's much easier for neural networks to learn game of life dynamics with the right activation function: Polynomial kolmogorov-arnold networks, 2026. URL https://arxiv.org/abs/2606.23587.

Yoshua Bengio. From system 1 deep learning to system 2 deep learning. Keynote at the 33rd Conference on Neural Information Processing Systems (NeurIPS), 2019. Available at https://www.youtube.com/watch?v=T3sxeTgT4qc.

Jaime A. Berkovich and Markus J. Buehler. LifeGPT: Topology-agnostic generative pretrained transformer model for cellular automata. *npj Artificial Intelligence*, 1:23, 2025. doi: 10.1038/s44387-025-00014-w.

Jaime A. Berkovich, Noah S. David, and Markus J. Buehler. AutomataGPT: Transformer-based forecasting and ruleset inference for two-dimensional cellular automata. *Advanced Science*, 2026. doi: 10.1002/advs.202511352.

Keith Burghardt, Jienan Liu, Sadman Sakib, Yuning Hao, and Bo Li. FAMOSE: A ReAct approach to automated feature discovery. *arXiv preprint arXiv:2602.17641*, 2026. doi: 10.48550/arXiv.2602.17641. URL https://arxiv.org/abs/2602.17641.

Hanseul Cho, Jaeyoung Cha, Srinadh Bhojanapalli, and Chulhee Yun. Arithmetic transformers can length-generalize in both operand length and count. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=eIgGesYKLG.

Krzysztof Cios, Lukasz Kurgan, and Lucy Goodenday. SPECT Heart. UCI Machine Learning Repository, 2001. DOI: https://doi.org/10.24432/C5P304.

Felix den Breejen, Sangmin Bae, Stephen Cha, and Se-Young Yun. Fine-tuned in-context learning transformers are excellent tabular data classifiers. *arXiv preprint arXiv:2405.13396*, 2024.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023a.

Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing CAAFE for context-aware automated feature engineering. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

Holger H Hoos and Thomas Stützle. Satlib: An online resource for research on sat. *Sat*, 2000:283–292, 2000.

Ashhadul Islam, Abdesselam Bouzerdoum, and Samir Brahim Belhaouari. Bio-inspired adaptive neurons for dynamic weighting in artificial neural networks. *AI Open*, 7:1–17, 2026. ISSN 2666-6510. doi: https://doi.org/10.1016/j.aiopen.2026.02.001. URL https://www.sciencedirect.com/science/article/pii/S266665102600001X.

Brenden M Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning (ICML)*, pp. 2873–2882, 2018.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=dsUB4bst9S.

Seunghan Lee. Mitigating label shift in tabular in-context learning via test-time posterior adjustment. In *International Conference on Machine Learning*, 2026.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://www.science.org/doi/abs/10.1126/science.abq1158.

Andreas Madsen and Alexander Rosenberg Johansen. Neural arithmetic units. In *International Conference on Learning Representations (ICLR)*, 2020.

Callum McDougall, Arthur Conmy, Cody Rushing, Thomas McGrath, and Neel Nanda. Copy suppression: Comprehensively understanding an attention head. *arXiv preprint arXiv:2310.04625*, 2023. https://arxiv.org/abs/2310.04625.

Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, et al. Transformers can do arithmetic with the right embeddings. *Advances in Neural Information Processing Systems*, 37:108012–108041, 2024.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=O9YTt26r2P.

Maxwell Nye, Anders Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *ArXiv*, abs/2112.00114, 2021. URL https://api.semanticscholar.org/CorpusID:244773644.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3), 2020.

Randal S Olson, William La Cava, Patryk Orzechowski, Ryan J Urbanowicz, and Jason H Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):36, 2017.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39643–39666. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/park24c.html.

Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, Cambridge, MA, 2002. ISBN 978-0-262-16209-8.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

John C Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing 83, Proceedings of the IFIP 9th World Computer Congres*, pp. 513–523, 1983.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*, 2019.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2018.

Jacob M. Springer and Garrett T. Kenyon. It's hard for neural networks to learn the game of life. *arXiv preprint arXiv:2009.01398*, 2020.

Simon Strandgaard. gameoflife-v1. https://huggingface.co/datasets/neoneye/gameoflife-v1, 2023. Dataset; accessed August 3, 2026.

Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 8046–8055, Red Hook, NY, USA, 2018. Curran Associates Inc.

Philip Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pp. 347–359. ACM, 1989. doi: 10.1145/99370.99404.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554. PMLR, 2019.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33832–33849, 2025. doi: 10.18653/v1/2025.emnlp-main.1716.

Zhun Yang, Adam Ishay, and Joohyung Lee. Learning to solve constraint satisfaction problems with recurrent transformer. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=udNhDCr2KQe.

Tianping Zhang, Zheyu Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Jian Li. OpenFE: Automated feature generation with expert-level performance. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Appendix

Table of Contents

A THE GEOMETRIC ALU "PROGRAM"

This appendix provides the exact matrix configurations for the existence proof discussed in Section 3.

A.1 DIMENSION MAP AND EMBEDDINGS

We utilize a sparse orthogonal basis in $\mathbb{R}^6$ (Table 7). D_0 serves as the continuous payload, while $D_1 - D_4$ act as discrete property flags.

Table 7: Dimension Assignments in the Residual Stream.

Index	Identifier	Functional Role
D0	`Value`	Numerical magnitude n (Payload)
D1	`Is_Num`	Binary flag (1.0) for numerical literals
D2	`Is_Add`	Binary flag (1.0) for the '+' operator
D3	`Is_Sub`	Binary flag (1.0) for the '-' operator
D4	`Is_Op`	Generic flag (1.0) for any operator
D5	`Accu`	Reserved register for conditional storage

Table 8: Sparse Token Embeddings $E(t)$

Token	D0	D1	D2	D3	D4	D5
Literal (n)	n	1	0	0	0	0
Plus ($+$)	0	0	1	0	1	0
Minus ($-$)	0	0	0	1	1	0

A.2 ATTENTION WEIGHTS

The Attention Layer solves the *Binding Problem* via an asymmetric "Look-Back" mechanism. We configure the Query/Key projections to establish a handshake between `Number` tokens and `Operator` tokens. The linear distance penalty (ALiBi) resolves collisions, ensuring that operands strictly attend to their nearest neighbor.

Addressing & Locality: We configure the Query projection to map the Number Flag (D_1) to a handshake channel (D_4), and the Key maps the Operator Flag (D_4) to the same. A high gain ($G = 10$) ensures numbers attend strongly to operators. Ties between multiple operators (e.g., $12 + 3 - 5$) are resolved by the linear distance penalty (ALiBi), $S_{ij} = (q_i \cdot k_j^T) - |i - j|$, which enforces a monotonic preference for the nearest neighbor.

Fetching: The Value projection copies the operator's flags (D_2, D_3) into the number's residual stream. The resulting state is a noisy superposition: a number token t contains a strong signal from its nearest operator and weak "leakage" from distant ones.

$$W_Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{10.0} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad W_K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{10.0} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The Value projection W_V copies the operator flags into the number's stream:

$$W_V = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1.0} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1.0} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A.3 FFN WEIGHTS

The attention output – a superposition of the number's identity and the operator's type – is then passed to the Gated Linear Unit (GLU). The GLU acts as a discriminator to filter attention noise. The gate is configured as a boolean AND circuit: it opens only if the token is a Number (D_1) *and* attends to a Minus sign (D_3). Importantly, the gate weights for the Value dimension (D_0) are set to zero, rendering the logic **value-invariant**.

The Gate (W_{gate}): The gate targets the condition $\texttt{Is\_Num} \wedge \texttt{Is\_Sub}$. We set weights of 100.0 for both D_1 (Num) and D_3 (Sub), with a bias of -150.0.

$$\text{Gate} = \sigma(100 \cdot \texttt{Num} + 100 \cdot p_{sub} - 150)$$

For a number token ($\texttt{Num} = 1$), this simplifies to $\sigma(100 \cdot p_{sub} - 50)$. If the nearest neighbor is Subtraction ($p_{sub} \approx 1.0$), the activation is $\sigma(50) \approx 1$. If it is Addition ($p_{sub} \approx 0$), the activation is $\sigma(-50) \approx 0$. Note that W_{gate} has zero weights for D_0, rendering logic independent of magnitude.

The Value (W_{val}): The value matrix simply maps the numerical payload (D_0) to the accumulator dimension (D_5).

$$W_{gate} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{100.0} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{100.0} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad b_{gate} = [0, 0, 0, 0, 0, \mathbf{-150}]$$

The Value matrix W_{val} copies the magnitude D0 to D5.

$$W_{val} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \mathbf{1.0} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A.4 READOUT

The output is computed via global sum pooling and projection W_{out}. $D0$ has the absolute values of all the numbers; $D5$ has the absolute values of the negative numbers, so we set weights $w_{D0} = 1$ and $w_{D5} = -2$.

$$y = \sum(1 \cdot D_0) + \sum(-2 \cdot D_5)$$

B ABLATION STUDY CONFIGURATIONS

To investigate whether the failure of dense networks to discover the geometric topology was due to specific hyperparameter choices or initialization strategies, we designed 12 distinct experimental configurations. These span the gamut of standard deep learning "tricks," including signal boosting, logical pre-initialization, auxiliary loss penalties, and data normalization.

Table 9 summarizes the specific modifications applied to the base model for each configuration.

B.1 DETAILED CONFIGURATION DESCRIPTIONS

Group A: Baselines and Normalization

- **Config A (Simple Baseline):** A simplified 6-dimensional Transformer block without input normalization or layer normalization. We utilize a single-head attention mechanism with **unscaled dot-product scoring** and a fixed linear decay bias ($B_{i,j} = -|i - j|$) to strictly enforce locality. The FFN utilizes a **Sigmoid-GLU activation** with learnable biases, while attention and readout projections are bias-free. Output is computed via **global sum-pooling** followed by a linear projection. *Hypothesis Tested:* Can standard gradient descent, operating in a raw, unityped vector space, overcome the "Simplicity Bias" of linear correlations? This configuration serves as the control to determine if the failure to reason is intrinsic to the lack of stratification

Table 9: Detailed hyperparameters for the 12 Ablation Configurations. All models share the same 6-dim architecture.

Config	Name	Mechanism / Modification	Hypothesis Tested
A	Baseline	Standard Xavier initialization; Raw inputs.	Can standard GD solve it?
As	Scaled	Inputs/Targets scaled (max-normalized).	Is magnitude the problem?
A$_{ln}$	LayerNorm	Internal LayerNorm	Feature Collapse (OOD Fail)
A$_{curr}$	Curriculum	Range $[0, 100] \rightarrow [0, 5000]$	Linear Trap (Locked Early)
B$_{fl}$	Frozen Logic	Logic weights fixed to ideal; others learned.	Can gradients flow through logic?
B$_{fa}$	Frozen Attn	Attention weights fixed to Oracle (Perfect).	Is the failure in Routing or Logic?
B$_{ortho}$	Value orthogonality	Loss penalty: $\lambda\|\|w \cdot x\|\|$.	Can we punish value-dependence?
C$_{sb}$	Signal Boost	Fixed pre-activation gain $\gamma = 100$.	Is the gradient signal too weak?
C$_g$	Learned Gain	Learnable scalar gain α (init 1.0).	Can the model learn to boost signals?
C$_b$	Balanced	Signals balanced manually.	Does a valid operating point exist?
C$_c$	Combined	Config B$_{ortho}$ + Config C$_b$	Do constraints + boosting help?
C$_{bi}$	Bias Init	Gate Bias = +1.0	Dead gates?

rather than specific optimization pathologies like vanishing gradients or covariate shift. *Structural Note:* We employ **global sum-pooling** ($y = W_{out} \sum x_t$) rather than the standard last-token readout. This architectural choice is critical: it aligns the readout mechanism with the accumulation nature of the arithmetic task, forcing the model to modify token states directly rather than relying on a privileged "read head" at the end of the sequence. Additionally, the attention scores are *unscaled* (omitting the $1/\sqrt{d_k}$ factor) given the low dimensionality ($d = 6$).

- **Config A$_s$ (Scaled Inputs):** Identical to Config A, but with **Max-Normalization** applied to the numerical inputs and targets. We apply a fixed scalar gain of $\gamma = 1/5000 = 0.0002$ to the Value payload (D_0) and the regression targets, mapping the operating range from $[0, 5000]$ to $[0, 1]$. *Hypothesis Tested:* Is the "Linear Trap" caused by the sheer magnitude imbalance between the Value channel ($x \approx 5000$) and the Control flags ($x \in \{0, 1\}$)? By normalizing the payload to the same unit scale as the flags, we test if the failure to learn logic is simply a result of the optimizer struggling with heterogeneous scales (i.e., vanishing gradients on the control signals due to large value-driven updates). *Structural Note:* Unlike standard normalization (e.g., BatchNorm) which relies on batch statistics, we employ a **fixed domain-scaling factor**. This ensures that the transformation is perfectly reversible and prevents the leakage of batch statistics into the inference logic. This places the Value neuron D_0 on the exact same magnitude footing as the Type neurons $D_1 - D_5$ without altering the distribution's shape.
- **Config A$_{ln}$ (LayerNorm):** Similar to Config A, but we apply standard Layer Normalization in a **Pre-LN** configuration (before Attention and FFN blocks) to the $d = 6$ residual stream. *Hypothesis Tested:* This configuration tests whether the optimization instability (e.g., Gradient Walls) observed in baselines stems from internal covariate shift. *Structural Note:* We acknowledge that applying LayerNorm to a low-dimensional space ($d = 6$) containing a single high-variance "Value" neuron (D_0) alongside five low-variance "Type" neurons ($D_1 - D_5$) represents a statistical extreme. This design is intentional; it serves as a rigorous test of **Unityped Normalization**. By forcing the numerical payload to share normalization statistics (μ, σ) with the control signals, we explicitly test the hypothesis that coupled normalization destroys the absolute magnitude information required for arithmetic.
- **Config A$_{curr}$ (Curriculum Learning):** We utilize the simple architecture (Config A) but modify the data sampling distribution over time. We employ a stepped curriculum schedule: for the first 25% of training steps ($t < 500$), values are sampled from a small range $x \in [0, 100]$. The range is expanded to intermediate complexity ($x \in [0, 1000]$) for the next 25% ($t < 1000$), and finally to the full Out-of-Distribution range ($x \in [0, 5000]$) for the remaining 50%. *Hypothesis Tested:* Is the failure driven by complexity overload? This tests if the "Linear Trap" is a result of the model being overwhelmed by the high variance of the full distribution early in training. By restricting the initial training to small integers, we test if the model can first learn the logic in a low-noise regime (where linear shortcuts are less dominant) and then robustly transfer that logic to larger magnitudes. *Structural Note:* The failure of this configuration (as observed in results) is particularly insightful. It suggests that the model "locks in" to the linear solution during the easy phase (where $y \approx x$ is a very strong proxy) and cannot unlearn it even when the data distribution shifts, highlighting the path-dependence of the optimization trap.

Group B: Topological Constraints

- **Config $\mathbf{B}_{fl}$ (Frozen Logic):** We initialize and freeze the FFN and Readout weights to the exact analytical solution of the "Geometric ALU" (see Appx. A), effectively creating a fixed, differentiable program. Only the Attention layers (W_Q, W_K, W_V) remain trainable. *Hypothesis Tested:* Is the failure located in the Logic (FFN) or the Routing (Attention)? By freezing the downstream logic to a known-perfect state, we test if the optimizer can solve the "Binding Problem" (learning to attend to operators). Failure here confirms the "Gradient Wall" hypothesis: it indicates that even with a perfect functional destination, the gradients generated by discrete, saturated logic gates ($\sigma(x) \approx 0$ or 1) vanish before they can update the upstream attention mechanism. *Structural Note:* The frozen logic implements the standard "Cut-and-Paste" subtraction algorithm ($y = D_0 - 2D_5$) triggered by an AND gate on the Type flags ($IsNum \wedge IsSub$). We use high-gain weights ($w = 100$) to ensure the gates are stiff, mimicking the conditions of a converged binary circuit.
- **Config $\mathbf{B}_{fa}$ (Perfect Routing):** We utilize **Frozen Oracle Attention**, fixing the attention weights to perform perfect token routing (Numbers attend to Operators). The raw attention output ($z \in [0, 1]$) is added directly to the high-magnitude residual stream ($x \in [0, 5000]$). *Hypothesis Tested:* Is the "Binding Problem" the primary bottleneck? By solving the routing problem perfectly for the model, we test if the optimizer's failure to discover logic is simply a failure to find the correct data associations. A failure here (as observed) isolates **Data-Control Interference** as a distinct pathology from Attention Failure: even when the model "knows" the operator is a Minus sign, the signal is too weak relative to the payload to trigger the logic gate. *Structural Note:* This configuration serves as the direct ablation for the Signal Boosting experiments (Group C). It establishes the baseline performance of a "correctly wired" but "energetically imbalanced" network.
- **Config $\mathbf{B}_{ortho}$ (Value-Orthogonality):** We utilize the simple architecture of Config A but augment the optimization objective with a targeted **Value-Independence Penalty**. We add a regularization term $\mathcal{L}_{reg} = \lambda \sum (W_{gate}[:, D_0])^2$ (with $\lambda = 10.0$) to the loss. This specifically penalizes the weights connecting the Value Stratum (D_0) to the Logic Gate, forcing the gate to be orthogonal to the numerical payload. *Hypothesis Tested:* Can stratification be induced via soft constraints? This configuration tests if the "Simplicity Bias" (the Linear Trap) can be overridden by explicitly punishing the specific linear shortcut the model prefers. If the optimizer is discouraged from using the high-magnitude Value channel for decision-making, we test if it naturally falls back on the Control channels ($D_1 - D_5$) or if it fails to converge entirely. *Structural Note:* This represents a "Soft Stratification" approach. Unlike the final STRAT architecture which physically removes these connections (Hard Stratification), this configuration leaves them topologically available but energetically expensive, testing the plasticity of the optimization landscape.

Group C: Signal Injection

- **Config $\mathbf{C}_{sb}$ (Signal boost):** We introduce a fixed, scalar pre-activation gain $\gamma = 100$ applied to the output of the Attention block, immediately preceding the residual connection ($x_{res} = x + \gamma \cdot \text{Attention}(x)$). All other parameters remain learnable. *Hypothesis Tested:* Does the "Gradient Wall" stem from signal attenuation? Standard Dot-Product Attention (softmax) often yields diffuse probability distributions (e.g., $p \approx 0.2$), resulting in weak updates that fail to drive the subsequent sigmoid gates ($\sigma(x)$) into their saturated, non-linear regime. By injecting a static gain $\gamma = 100$, we force the attention signal to be magnitude-compatible with the stiff logical thresholds required by the FFN, testing if this simple scaling resolves the optimization deadlock. *Structural Note:* The gain is applied *post-projection* but *pre-residual*. This is crucial: it amplifies the "update" signal (the Type information) relative to the pass-through "residual" (the Value payload), effectively altering the signal-to-noise ratio in favor of the control logic without altering the FFN architecture itself.
- **Config $\mathbf{C}_g$ (Learned Gain):** We introduce a single **learnable scalar parameter** α, initialized to 1.0, which scales the output of the Attention block before the residual connection ($x_{res} = x + \alpha \cdot \text{Attention}(x)$). *Hypothesis Tested:* Can the optimizer autonomously discover the high-gain regime required for logic? This tests the "Chicken-and-Egg" dilemma of the Gradient Wall: to get strong gradients from the sigmoid gate, the input signal must be large (high gain); but to learn a large gain, the model needs strong gradients. Initializing $\alpha = 1.0$ tests if there is a viable path across the optimization landscape from the standard operating point to the high-gain solution. *Structural Note:* We use a scalar parameter rather than a vector to prevent the model from using the gain term to fundamentally alter the direction of the attention update (i.e., reshaping the embedding). It acts strictly as a **volume knob** for the Control Signal.
- **Config $\mathbf{C}_b$ (Balanced Operating Point):** We utilize **Frozen Oracle Attention** (Config B_{fa}), initializing and fixing the attention weights to perform perfect token routing (see Appx. A). To address the signal-to-noise imbalance, we initialize a learnable gain parameter to $\alpha = 50.0$. This ensures the Control signal entering the residual stream is of comparable magnitude to the Data payload. *Hypothesis Tested:* Does a valid *Operating*

Point exist? By manually placing the model in a regime with high Signal-to-Noise Ratio (SNR) and perfect routing, we test if the failure to learn is due to the impossibility of the task or simply the difficulty of reaching this basin of attraction. We essentially "teleport" the optimizer to the vicinity of a good solution to see if it can converge. *Structural Note:* Unlike Config B_{fl} (Frozen Logic), here we freeze the *Routing* and learn the *Logic*. The initialization of $\alpha = 50.0$ acts as a manual "balancing" of the subspace variances, preventing the Data Stratum from drowning out the Control Stratum during the initial critical gradients.

- **Config $\mathbf{C}_c$ (Synergistic Constraints):** We combine the structural intervention of **Signal Injection** (C_{sb}) with the optimization constraint of **Soft Stratification** (B_{ortho}). The model utilizes a learnable pre-activation gain initialized to $\alpha = 50.0$ to ensure the Control signal is visible (addressing the Gradient Wall). Simultaneously, we apply the Value-Independence Penalty ($\lambda = 10.0$) to the loss function to actively discourage the linear shortcut (addressing the Linear Trap). *Hypothesis Tested:* Can we engineer a viable optimization path by attacking both failure modes simultaneously? This configuration tests the "Carrot and Stick" hypothesis: the gain $\alpha = 50$ acts as the "Carrot" (making the logic gate accessible), while the penalty λ acts as the "Stick" (making the linear shortcut expensive). We test if this dual pressure is sufficient to guide the optimizer into the correct topological solution. *Structural Note:* While Config C_{sb} uses a fixed gain, here we employ a **learnable parameter** initialized to the high-gain regime ($\alpha = 50$). This provides the necessary initial signal strength to overcome the Sigmoid Tax, while allowing the model to fine-tune the gain magnitude during convergence.
- **Config $\mathbf{C}_{bi}$ (Bias Initialization):** This is identical to Config A, but we modify the initialization of the FFN Logic Gates by explicitly setting the bias terms to $b_{gate} = +1.0$ (replacing the standard uniform initialization). This sets the initial activation of the sigmoid gates to $\sigma(1.0) \approx 0.73$. *Hypothesis Tested:* Is the failure caused by "Dead Gates"? Standard initialization can sometimes place logic gates in a regime where they block flow or provide weak gradients, causing the optimizer to abandon the path. By forcing the gates into a conductive ("Open") state at initialization, we test if maintaining strong initial gradient flow allows the model to eventually learn the discrete closing mechanism, or if it traps the model in a linear local minimum. *Structural Note:* This initialization places the model in a **Pseudo-Linear Regime**. Since the gate activation is roughly constant (≈ 0.73) and conductive at start, the GLU behaves like a linear projection ($y \approx 0.73 W_{val} x$). This lowers the barrier to learning arithmetic but raises the risk of the "Open Gate Trap," where the model overfits the values without ever learning to shut the gate (switch) for distractor tokens.

B.2 Detailed Failure Analysis

B.2.1 The "Honest" Failures

High ID error reveals that these models failed to learn the logic gate entirely, defaulting to heuristic baselines.

The Linear Trap (Configs A, $\mathbf{A}_s$, $\mathbf{A}_{curr}$, $\mathbf{B}_{ortho}$): The Baseline (**Config A**) achieves a median ID MAE of 15.5, suggesting it is merely tracking the input mean or applying a linear regression on the value channel ($y \approx x$).

Regarding scale, **Config $\mathbf{A}_s$ (Scaled Inputs)** also failed to resolve the problem. In fact, scaling the inputs to $[0, 1]$ resulted in worse performance (ID MAE 98.7), likely because normalizing the massive value channel to the same magnitude as the sparse control flags deprived the optimizer of the only strong gradient signal it had.

Similarly, **Config $\mathbf{A}_{curr}$ (Curriculum)** yielded the highest errors of all (ID MAE 600.3). This result indicates that the model "locked in" to a linear solution during the early, easy phase ($x \in [0, 100]$) and could not unlearn it when the distribution shifted, highlighting the path-dependence of the trap.

The Gradient Wall (Configs $\mathbf{B}_{fl}$, $\mathbf{C}_{sb}$, $\mathbf{B}_c$): Medians in the range of $\approx$ 45-60 (e.g., **Config $\mathbf{B}_{fl}$** at 58.7) indicate models stuck between the Linear Trap and a random baseline. This validates the "Gradient Wall" hypothesis: even when we freeze the logic to be perfect (B_{fl}) or boost the signal (C_{sb}), the gradients vanish at the saturation points of the sigmoid gates, stranding the weights in a local minimum where the gate remains permanently closed or open.

B.2.2 The "Dishonest" Successes

A subset of configurations appeared successful during training (low ID error) but failed catastrophically on OOD data. This is the most dangerous failure mode, as it masquerades as a solution.

Magnitude Leaks (Config $\mathbf{B}_{fa}$ & $\mathbf{C}_b$): Config $\mathbf{B}_{fa}$ (Frozen Oracle Attention) achieved an excellent ID MAE of 6.2, implying it had solved the task. However, its OOD error exploded to **11,817**. Even with perfect routing provided by the oracle, the dense layers used the *value magnitude* as a proxy for the decision boundary rather than the boolean flag. Similarly, **Config $\mathbf{C}_b$** (Balanced Operating Point) achieved the best training score of the entire ablation (ID MAE 1.8)

but the worst generalization (OOD MAE $11,916$). This proves that fixing the signal-to-noise ratio via initialization merely allows the model to overfit the operating point more efficiently, without discovering the topological switch.

The Open Gate Trap (Config $\mathbf{C}_{bi}$): Initializing gate biases to $+1.0$ allowed the model to achieve a respectable ID MAE of 12.7. However, the OOD error $(8,471)$ confirms that the gates simply remained open, acting as a linear pass-through that failed to block distractor tokens in the high-magnitude regime.

C THE TWO REGIMES OF LEARNED LOGIC

Our analysis of the learned weights across 10 independent training runs reveals that while the STRAT-12 architecture consistently converges to the same topological mechanism (the "Cut-and-Paste" router), the specific implementation depends on the sign initialization of the Readout Layer. This symmetry breaking results in two distinct logical regimes: the **Standard Attractor** (Add-by-Default) and the **Mirror Attractor** (Subtract-by-Default).

Here we present the raw weights from two representative runs to demonstrate these algorithms. These weights were extracted after 2,000 steps of training.

C.1 RUN 1: THE STANDARD ATTRACTOR (SEED 42)

```
[1] READOUT LAYER (W_out)
Equation: y = sum(W_out * x)
        D0(Val)    D1(Num)     D2(+)     D3(-)    D4(Op)  D5(Scratch)
Output   1.0005    -0.0149   -0.0123   -0.2240    0.2774       1.1755

[2] THE ROUTER (FFN)
(A) Logic Gates (W_gate)
Input Columns: D1(Num), D2(+), D3(-), D4(Op)
           D1(Num)     D2(+)     D3(-)    D4(Op)       Bias
Gate_D0    -0.7223   -2.7710    2.8255    2.6739     0.0483
Gate_D5    -0.5233   -3.3227    2.4424    2.5473    -0.1033

(B) Value Routing (W_val)
Input Columns: From_D0, From_D5
           From_D0   From_D5
To_D0      -1.0120   -0.0112  (The "Cut": Adds ~ -1.0x to D0)
To_D5      -0.8460    0.0017  (The "Paste": Adds ~ -0.85x to D5)

[3] ATTENTION LAYER (Control Subspace Only)
Rows/Cols correspond to flags: D1(Num), D2(+), D3(-), D4(Op)

(A) Query (W_q)
          D1       D2       D3       D4
D1    0.4299  -0.8417  -1.4502  -1.3617
D2    0.3148  -0.9022  -1.1602  -1.1371
D3    0.8778  -1.7965  -0.1897  -1.1168
D4    0.9766  -1.0427  -0.0547  -0.7767

(B) Key (W_k)
          D1       D2       D3       D4
D1   -0.2308  -0.5511   0.5377   0.7437
D2   -0.6876  -0.9046   0.4638   0.7889
D3   -0.7544  -2.5763   0.3965   0.0591
D4   -0.2349  -2.4472   0.9284  -0.1426

(C) Value (W_v)
          D1       D2       D3       D4
D1    0.8222   0.1646  -0.2627   0.0101
D2    1.6802   0.8191  -1.0734  -1.0369
D3   -1.3298  -0.2486   0.9975   1.2483
```

```
D4   -1.2396  -0.7179   1.3799   0.9310
```

Overall Interpretation: Standard logic. Positive Readout. The Attention layer (W_v) passes flags largely unmodified (positive weights on diagonal), and the Gate opens on MINUS (D_3) to trigger the Cut-and-Paste.

Condition: Readout Layer initializes with positive weights ($w_{out} > 0$). **Strategy:** The model treats the residual stream as an "Addition Conveyor Belt." It performs addition lazily (by doing nothing) and subtraction actively (by destroying the payload and creating a negative copy).

1. The Readout Layer The readout weights for the input (D_0) and scratchpad (D_5) are positive.

$$y \approx 1.00 \cdot D_0 + 1.18 \cdot D_5 \tag{5}$$

Implication: If the logic gates stay closed ($D_0 = x, D_5 = 0$), the output is $+x$.

2. The Gate Logic The gates are biased to be closed (≈ 0), and learn positive weights for the `MINUS` flag (D_3).

$$\begin{aligned} \text{Bias}_{D0} &\approx 0.05 \\ w_{gate}(D_3) &\approx +2.83 \quad \text{(Opens on Minus)} \\ w_{gate}(D_2) &\approx -2.77 \quad \text{(Closes on Plus)} \end{aligned}$$

3. The Active Routing (Minus Token) When triggered, the Value Router performs the "Cut-and-Paste" operation:

- **Cut ($D_0 \to D_0$):** The update is $-1.01x$, canceling the residual x. ($D_0 \to 0$).
- **Paste ($D_0 \to D_5$):** $w \approx -0.85$. A negative copy is moved to D_5. ($D_5 \to -0.85x$).

Total Output: $1.0(0) + 1.18(-0.85x) \approx -\mathbf{x}$.

C.2 Run 2: The Mirror Attractor (Seed 1000)

```
[1] READOUT LAYER (W_out)
Equation: y = sum(W_out * x)
        D0(Val)    D1(Num)     D2(+)     D3(-)    D4(Op)  D5(Scratch)
Output  -1.0025     0.1690    0.5371    0.0803    0.6230      -1.0771

[2] THE ROUTER (FFN)
(A) Logic Gates (W_gate)
Input Columns: D1(Num), D2(+), D3(-), D4(Op)
           D1(Num)     D2(+)     D3(-)    D4(Op)       Bias
Gate_D0     1.6707   -2.4147    1.8345    1.6123     0.8002
Gate_D5     1.5939   -2.1411    1.8549    1.7351     0.7962

(B) Value Routing (W_val)
Input Columns: From_D0, From_D5
           From_D0   From_D5
To_D0      -0.8788    0.0005
To_D5      -1.0361   -0.0021

[3] ATTENTION LAYER (Control Subspace Only)
Rows/Cols correspond to flags: D1(Num), D2(+), D3(-), D4(Op)

(A) Query (W_q)
          D1       D2       D3       D4
D1   -1.2564  -0.6667   0.2862  -0.6111
D2   -0.7708   0.4338  -1.2994  -0.4427
D3    1.0786   0.7103   0.5763   0.7327
D4   -0.8597  -0.4932  -0.6717  -0.9511

(B) Key (W_k)
          D1       D2       D3       D4
```

```
D1    0.9547  -1.3567  -0.3891  -1.2468
D2    0.2952  -0.3929  -1.2440  -1.1796
D3   -1.1966   0.9828   0.8603   1.0142
D4    0.7991  -0.5533  -0.7871  -1.2244

(C) Value (W_v) -- Note the inversion on D2(+) and D3(-)
          D1       D2       D3       D4
D1   -0.0692   1.2879  -1.2549  -1.0564
D2   -0.8868  -2.3064   1.0775   0.6756  <-- CRITICAL INVERSION
D3    0.4572   0.8875  -0.7919  -0.0625
D4    0.8125   1.2767  -0.2441  -1.2211
```

Overall Interpretation: Inverted logic. Negative Readout. Note the critical inversion in `W_v` (Row D2, Col D2 is -2.30), which flips the PLUS flag to trigger the "Cut-and-Paste" logic, effectively turning addition into a double-negative operation.

Condition: Readout Layer initializes with negative weights ($w_{out} < 0$). **Strategy:** The model treats the residual stream as a "Subtraction Conveyor Belt." It performs subtraction lazily (by doing nothing) and addition actively (by inverting the logic to force a double-negative).

1. The Readout Layer The readout weights are inverted.

$$y \approx -1.00 \cdot D_0 - 1.08 \cdot D_5 \tag{6}$$

Implication: If the logic gates stay closed ($D_0 = x, D_5 = 0$), the output is $-1.00(x) = -\mathbf{x}$. The default state is Subtraction.

2. The Logic Inversion To perform Addition, the model must trigger the "Cut-and-Paste" mechanism on `PLUS` tokens. However, the gate weights initially appear paradoxical:

$$w_{gate}(D_2 \text{ aka Plus}) \approx -2.41 \quad \text{(Naively Closes on Plus)}$$
$$w_{gate}(D_3 \text{ aka Minus}) \approx +1.83 \quad \text{(Naively Opens on Minus)}$$

Under standard flags, this would fail (Closing on Plus would yield the default $-x$). The model resolves this by **inverting the control flags in the Attention Layer**.

- The Attention matrix W_V learns a large negative weight (-2.31) for the self-attention of the `PLUS` flag.
- A `PLUS` token (1.0) is transformed into a negative flag (≈ -1.3).
- **Gate Activation:** $(-1.3) \times (-2.41) \approx +\mathbf{3.13}$.

The gate effectively **Opens on Plus** due to the double inversion (Negative Flag $\times$ Negative Weight).

3. The Active Routing (Plus Token) When triggered, the router executes the same geometric move:

- **Cut ($D_0 \to D_0$):** $w \approx -0.88$. ($D_0 \to \approx 0$).
- **Paste ($D_0 \to D_5$):** $w \approx -1.04$. ($D_5 \to -1.04x$).

Total Output: Because the readout is negative, the pasted negative value becomes positive:

$$y \approx -1.08 \cdot D_5 = -1.08 \cdot (-1.04x) \approx +\mathbf{x} \tag{7}$$

C.3 Synthesis

Both regimes implement an identical geometric operation: an orthogonal basis exchange enabled by the topological isolation of the magnitude subspace.

D Evaluation Across Diverse Tasks: Experimental Details

This appendix documents the 11 evaluations reported in Section 7 (Table 6). Each section covers one dataset in a fixed order: the source data and prediction target, the construction of the training, test, and OOD test sets, the model configuration, and the training configuration.

Table 10: Comparison of the two learned algorithms.

Feature	Standard (Run 1)	Mirror (Run 2)
Readout Sign	Positive $(+)$	Negative $(-)$
Default Operation	Addition $(+x)$	Subtraction $(-x)$
Active Trigger	`MINUS` Token	`PLUS` Token
Attention Role	Pass-through	Signal Inversion
Resulting Logic	$y = \text{Default} - \text{Routed}$	$y = \text{Default} + \text{Routed}$

Table 11: Trainable parameter counts for STRAT and the Transformer baseline across the 11 dataset evaluations.

Dataset	STRAT parameters	Transformer parameters
SATNet Sudoku	396,335	805,349
SATBench	1,648	5,310
SATLIB	1,648	5,310
Tic-Tac-Toe Endgame	33,474	58,498
Game of Life	17,993	30,761
PMLB Parity5	35,314	60,594
PMLB M-of-N	17,722	30,426
PMLB MONK-2	17,722	30,426
PMLB LED24	17,722	30,426
PMLB ThreeOf9	17,722	30,426
SPECT Heart	17,722	30,426

D.1 SATNet Sudoku

SATNet Sudoku provides 1,100 pairs of 9×9 puzzles and completed solutions (Wang et al., 2019), split into 100 training and 1,000 test puzzles. The task fills every empty cell while preserving the given digits and Sudoku constraints.

D.1.1 Dataset Construction

Training-set construction. We use the first 10 puzzles from the 100-puzzle training split as the base training set. At each epoch, we replace each base puzzle with a randomly transformed version and apply the same transformation to its solution. The transformation randomly relabels the nine digits, permutes the three row bands and the rows within each band, applies the corresponding permutations to column stacks and columns, and transposes the grid with probability 0.5.

Test-set construction. We combine the other 90 puzzles from that split with the original 1,000 test puzzles, yielding 1,090 test boards.

OOD test-set construction. A *band* is a horizontal group of three rows, and a *stack* is a vertical group of three columns. For each puzzle, we define I as the maximum of two quantities: the range of clue counts across the three bands and the range across the three stacks. Across the 1,100 source puzzles, the mean clue count is 36.21 and the mean I is 3.10. In total, 97.1% of the puzzles satisfy $I \leq 5$, and the largest I among the 10 training puzzles is 5. From the remaining 1,090 puzzles, we select all eight puzzles with exactly 35 clues and $I \in \{6, 7\}$. We then apply the same Sudoku automorphisms used for training augmentation to generate 500 unique puzzle–solution pairs. Because these automorphisms preserve I, every generated puzzle retains $I \in \{6, 7\}$. The resulting OOD set therefore lies outside the support of the augmented training set with respect to I. For consistent presentation and evaluation orientation, we rotate each pair so that its densest band occupies the bottom three rows.

D.1.2 Model Configuration

We represent each of the 81 cells with one token. STRAT assigns 64 channels to Type and 18 to Data. Thirty-seven Type channels encode row, column, and box membership, whether the cell is given, and the given digit. The other 27 channels start at zero. Because a given digit fixes the constraints on its cell, we treat it as immutable Type information. Data allocates nine channels to a digit register and nine to a candidate register. Given cells initialize the digit register with a one-hot digit; every other Data channel starts at zero.

STRAT stacks 16 four-head attention layers. It computes queries and keys from Type, then uses the resulting routes to carry Data into a width-128 GLU with one hidden layer. Separate LayerNorm modules normalize Type and Data. A final Data readout produces nine digit logits per cell.

The Transformer concatenates the same 64 Type channels and 18 Data channels, then adds two zero channels to form a width-84 stream divisible across four heads. Its 16 attention layers use a one-hidden-layer FFN of width 128. Self-attention and the FFN update the full stream, and the readout produces nine digit logits per cell.

D.1.3 Training Configuration

We train both models on the augmented base puzzles for 6,000 epochs using AdamW with a learning rate of 10^{-3}. The loss is cross-entropy over the empty cells, and gradients are clipped to a norm of 1.0. For each paired seed from 41 to 50, we evaluate the checkpoint from the final epoch. We count a board as correct only if all 81 cells match the reference solution, with the given cells copied directly from the input puzzle.

D.2 SATBench

SATBench contains 2,100 Boolean formulas in conjunctive normal form (CNF), derived from natural-language logic scenarios (Wei et al., 2025). We use these formulas to construct an assignment-verification task. Given a formula and a Boolean assignment, predict whether the assignment satisfies every clause.

D.2.1 Dataset Construction

Training-set construction. The dedicated training split contains 10 satisfiable formulas with 4–50 clauses and 16–90 variables. A SAT solver provides one satisfying assignment for each formula. At every epoch, we resample training examples from these 10 formula–assignment pairs. Each epoch contains 16 positive and 16 negative examples. For a positive example, we use the solver assignment directly. To construct a negative example, we perturb the solver assignment by flipping one, two, or three variables and select the first perturbation that violates at least one clause. If none of these perturbations produces a violation, we flip five variables instead.

Test set. The test split contains 2,090 formulas with 4–50 clauses: 1,040 satisfiable and 1,050 unsatisfiable. Each satisfiable formula is paired with a solver-produced satisfying assignment and labeled positive, while each unsatisfiable formula is paired with a seeded random assignment and labeled negative.

OOD test set. From the test set, we sample 250 positive and 250 negative examples and cyclically repeat each formula's clause sequence until it contains exactly 100 clauses.

D.2.2 Model Configuration

We serialize each formula with eight token types: evaluated true literal, evaluated false literal, OR, AND, clause start, clause end, end of sequence, and padding. An eight-dimensional one-hot vector serves as the Type state. Data has three channels. The first holds $+1$ for a true literal, -1 for a false literal, and zero for structural tokens; the other two provide workspace.

Four single-head attention layers process the sequence. In STRAT, attention reads and updates the eight-dimensional Type state, while a width-32 GLU with one hidden layer updates the width-3 Data state.

The Transformer packs the same fixed fields into an 11-dimensional residual stream. It has four single-head layers and a width-32 FFN with one hidden layer. Attention and the FFN update the full stream.

D.2.3 Training Configuration

Each epoch draws a balanced batch from the 10 training formulas: 16 satisfying assignments and 16 violating assignments. A SAT solver supplies the satisfying assignments. We search for violating assignments by flipping one to three variables; if that search fails, the implementation applies a five-variable fallback flip. STRAT and the Transformer receive the same sampled formula and assignment pairs. We train for 2,000 epochs with Adam at learning rate 10^{-2}, binary cross-entropy, and gradient clipping at norm 1.0. The final-epoch checkpoint yields assignment-level accuracy for each paired seed in 41–50.

D.3 SATLIB

The SATLIB collection we use contains 2,000 random 3-CNF formulas over 20 variables (Hoos & Stützle, 2000): 1,000 satisfiable formulas with 91 clauses and 1,000 unsatisfiable formulas with 110 clauses. Every clause has exactly three literals. SATLIB labels each formula as satisfiable or unsatisfiable; we instead use the formulas for the same assignment-verification task as SATBench, with the same definition of a violated clause.

D.3.1 Dataset Construction

Training set. The first 10 satisfiable formulas in file order form the training pool. We generate training examples using the same procedure as in SATBench. At each epoch, we sample 16 positive examples from solver-produced satisfying assignments and 16 negative examples by flipping a small number of variables in those assignments.

Test set. The remaining 1,990 formulas form the ID test set. For the 990 satisfiable formulas, positive examples use solver-produced satisfying assignments. For the 1,000 unsatisfiable formulas, negative examples use random assignments drawn from the evaluation seed.

OOD test set. We select the first 250 satisfiable formulas from the test set in file order. For each formula, we use its solver-produced satisfying assignment as a positive example and create a negative example by flipping one variable. This produces 500 examples over 250 formulas with balanced labels. We then append the first nine clauses to each 91-clause formula, producing a 100-clause formula.

D.3.2 Model Configuration

SATLIB uses the SATBench model configuration and the same eight token types. STRAT has an eight-dimensional Type state, a width-3 Data state, four single-head layers, and a width-32 GLU with one hidden layer.

The Transformer packs the same fixed fields into an 11-dimensional stream. Its four single-head layers and a width-32 FFN with one hidden layer.

D.3.3 Training Configuration

Each epoch draws 16 satisfying and 16 violating assignments from the 10 training formulas, following the SATBench construction. We train both models for 2,000 epochs with Adam at learning rate 10^{-2}, binary cross-entropy, and gradient clipping at norm 1.0. For each paired seed in 41–50, the final checkpoint gives assignment-level accuracy on the 1,990 test formulas.

D.4 Tic-Tac-Toe Endgame

The UCI Tic-Tac-Toe Endgame dataset contains 958 legal terminal board positions (Aha, 1991). Each board represents its nine cells as *X*, *O*, or blank. We retain the original binary label, which is positive when *X* forms a complete row, column, or diagonal.

D.4.1 Dataset Construction

Training set. We select 10 base boards from the 958 examples. Because the winning-line rule is invariant to rotations and reflections, we apply all eight symmetries of the square to each base board and remove duplicates. This produces 76 unique training boards, which we use to train both models.

Test set. The remaining 948 boards form the initial test set. Among them, 66 are rotations or reflections of a base board and therefore appear in the augmented training set. We exclude these 66 boards, leaving 882 boards for the ID evaluation.

OOD test set. Ten base boards cannot cover every winning configuration, so the augmented training set contains incidental correlations between mark locations and the label. We construct the OOD set from test boards whose spatial statistics favor the opposite label under these correlations. For each board, we count *X*, *O*, and blank cells within three regions: the center, the four corners, and the four edges. These counts form a nine-dimensional vector $r(x)$ that is invariant to rotations and reflections. Let c be the difference between the mean $r(x)$ of positive and negative training boards. A positive score $r(x)^\top c$ indicates greater similarity to the positive training boards under these spatial statistics. From the 948 test boards, we retain positive boards with $r(x)^\top c < 0$ and negative boards with $r(x)^\top c > 0$, yielding 216 positive and 119 negative candidates. Removing boards that appear in the augmented training set leaves

195 positive and 112 negative boards. We downsample the positive candidates to 112 boards. The resulting OOD test set contains 112 positive and 112 negative boards.

D.4.2 Model Configuration

We encode each board as one classification token, nine cell tokens, and eight line tokens. The line tokens represent the three rows, three columns, and two diagonals. STRAT represents every token with separate width-32 Type and Data streams. For a cell token, Type encodes its role, row, column, diagonal membership, and immutable board value. Line-token Type identifies the line and its member cells. The board value belongs to Type because it determines how the corresponding cell participates in each candidate line.

Each cell token initializes four identical eight-channel Data registers. A register contains a seven-way categorical field and a presence bit. The classification and line tokens start with zero Data.

STRAT applies two four-head attention layers and a two-hidden-layer GLU with hidden widths 64 and 64. The Transformer concatenates the same observable Type and Data fields into one width-64 residual stream. It applies two four-head attention layers and a two-hidden-layer FFN with hidden widths 64 and 64. Both models predict from the classification token.

D.4.3 Training Configuration

We apply the eight rotations and reflections of the square to each base board, then remove duplicate configurations. Both models receive the same augmented set. Training runs for 2,000 epochs with full-batch cross-entropy, AdamW at learning rate 10^{-3}, and gradient clipping at norm 1.0. For paired seeds 41–50, the final-epoch checkpoint provides board-level classification accuracy.

D.5 Game of Life

From the public *gameoflife-v1* collection, we retain 767 one-step transitions on 6×6 boards.[1] The boards wrap around at the boundaries, giving every cell exactly eight neighbors (a toroidal Moore neighborhood). The target is the board after one update under the B3/S23 rule (Strandgaard, 2023). Under this rule, a dead cell becomes alive when exactly three neighbors are alive, while a live cell survives when two or three neighbors are alive.

D.5.1 Dataset Construction

Training set. The first 10 board–successor pairs in file order form the base training set. At each epoch, we sample one random transformation and apply it jointly to every board and its successor in the batch. The transformation combines a toroidal translation, a rotation by a multiple of 90°, a row reflection, and a column reflection. Because the B3/S23 update commutes with these transformations, every transformed pair remains a valid one-step transition.

Test set. The remaining 757 pairs form the ID test set.

OOD test set. The B3/S23 update depends on each cell's number of live neighbors. We therefore test whether the models generalize to spatial arrangements that do not occur in training. For each board, we record the number of live cells and the number of live–live adjacencies. A live–live adjacency is an unordered pair of neighboring live cells on the torus. Both quantities are invariant under the training transformations. The 10 training boards contain 7–32 live cells and at least 10 live–live adjacencies. We generate 500 unique boards with exactly seven live cells: 250 have eight live–live adjacencies and 250 have nine. We compute each successor using the B3/S23 rule. Only 7 of the 757 ID test boards fall in this regime, and none of the 500 generated boards appears in the source collection. Every OOD successor contains at least one live cell, so an all-dead prediction achieves zero exact-match accuracy.

D.5.2 Model Configuration

We represent each of the 36 cells with one token. STRAT assigns 32 channels to Type and 32 to Data. Type stores the immutable alive/dead input state as a two-channel one-hot vector (channel 0 dead, channel 1 alive); all other Type channels start at zero. The input bit belongs to Type because it determines which branch of the transition rule governs the cell. Data holds the alive bit in channel 0 and initializes the other channels to zero as scratch. One four-head STRAT attention layer computes routes from Type and transports Data to a two-hidden-layer GLU with hidden widths 68 and 68. The Transformer concatenates the same observable fields into a width-64 residual stream. Its single four-head layer uses a two-hidden-layer FFN with hidden widths 68 and 68.

[1] https://huggingface.co/datasets/neoneye/gameoflife-v1

D.5.3 TRAINING CONFIGURATION

We train both models on the same 10 board pairs for 6,000 epochs. At each epoch, we apply the same randomly sampled toroidal translation, rotation, and row and column reflections to the full batch. Optimization uses AdamW at learning rate 10^{-3}, binary cross-entropy over all 36 cells, and gradient clipping at norm 1.0. We evaluate the final-epoch checkpoint for paired seeds 41–50. A prediction counts as correct only when all 36 cells match the reference next state.

D.6 PMLB PARITY5

The Parity5 task uses ten-bit inputs. The first five bits are one of the 32 assignments listed in PMLB Parity5 (Olson et al., 2017); we call them the relevant bits. The other five bits are *nuisance bits* that always share one value, 0 or 1. The label is the parity of the five relevant bits: an input is positive when an odd number of them are active. The nuisance bits never affect the label. The input space contains $32 \times 2 = 64$ distinct inputs.

D.6.1 DATASET CONSTRUCTION

Training set. We randomly select 10 of the 32 assignments. For each selected assignment, the training set contains four copies with the nuisance value equal to the parity label and one copy with the opposite nuisance value. This produces 50 training examples over 20 distinct inputs, with the nuisance value agreeing with the label in 80% of the examples.

Test set. The test set contains 160 class-balanced examples covering all 64 distinct inputs. Each input whose nuisance value agrees with the label appears four times, whereas each disagreeing input appears once. The nuisance value therefore agrees with the label in 80% of the examples, matching the training distribution.

OOD test set. The OOD test set contains the same 64 inputs with the frequencies reversed: each agreeing input appears once, whereas each disagreeing input appears four times. This produces 160 class-balanced examples with 20% nuisance–label agreement. The ID and OOD sets have identical input support and use the same parity rule; they differ only in the nuisance–label correlation.

D.6.2 MODEL CONFIGURATION

we represent each token with the same 32-dimensional Type representation and 32-dimensional Data representation. STRAT processes these representations as separate streams. In Type, the first dimension identifies the classification token, the next two distinguish relevant from nuisance bits, and the following two encode the observed binary value. The classification token activates its own indicator and the relevant-group indicator. Each bit token activates its corresponding group indicator and one of the two value dimensions. The remaining Type dimensions are initialized to zero. In Data, the first two dimensions encode the observed binary value of each bit token, while the remaining 30 dimensions are initialized to zero and serve as scratch space for intermediate computation. The classification token's Data is initialized entirely to zero.

STRAT applies two four-head attention layer and a two-hidden-layer GLU with hidden widths 68 and 68. The Transformer concatenates the same observable fields into a width-64 residual stream. It uses two four-head layer and a two-hidden-layer FFN with hidden widths 68 and 68.

D.6.3 TRAINING CONFIGURATION

We use full-batch cross-entropy, AdamW at learning rate 10^{-3}, and gradient clipping at norm 1.0 for 2,000 epochs. For paired seeds 41–50, we evaluate the final-epoch checkpoint and report accuracy on the 160-example ID and 160-example OOD test sets described above.

D.7 PMLB M-OF-N

The PMLB M-of-N dataset contains 1,324 examples with 10 binary features (Olson et al., 2017). A label is positive when at least three of seven designated features are active. Three additional features do not affect the label. The notation $(3, 7, 10)$ denotes the threshold, the number of relevant features, and the total number of features. We refer to the three irrelevant features as nuisance bits.

D.7.1 Dataset Construction

Training set. We randomly select 10 examples from the 1,324-example dataset. We augment each base example with the seven cyclic shifts of its relevant features while leaving the nuisance features unchanged, producing 70 distinct training inputs

Test set. After excluding every source row whose input appears in the augmented training set, the remaining 1,231 rows form the test set.

OOD test set. We construct the OOD test set by shifting the relevant-feature-count distribution from negative examples with zero to two active relevant features and positive examples with three to seven in ID to negative examples with exactly two active relevant features and positive examples with exactly three in OOD. We begin with the 1,231 held-out source rows and retain only the examples satisfying these conditions. The resulting 494 source rows correspond to 378 unique inputs after duplicates are merged.

D.7.2 Model Configuration

We encode each input with one classification token followed by 10 feature tokens. For both models, each token is represented by 32 Type dimensions and 32 Data dimensions. STRAT processes these representations as separate streams. In Type, the first dimension identifies the classification token, the next two distinguish relevant from nuisance features, the following 10 identify the individual features, and the next two encode the observed binary value. Each feature token activates its corresponding group indicator, feature identity, and one of the two value dimensions. The classification token activates only its own indicator, and the remaining Type dimensions are initialized to zero. In Data, the first two dimensions encode the observed binary value of each feature token, while the remaining 30 dimensions are initialized to zero and are available as scratch space for intermediate computation. The classification token's Data is initialized entirely to zero.

STRAT applies one four-head attention layer and a two-hidden-layer GLU with hidden widths 68 and 68. The Transformer concatenates the same observable fields into a width-64 stream. Its single four-head attention layer uses a two-hidden-layer FFN with hidden widths 68 and 68. Both models read the prediction from the classification token.

D.7.3 Training Configuration

Both models receive the same full batch and train for 2,000 epochs. For paired seeds 41–50, we report final-checkpoint accuracy.

D.8 PMLB MONK-2

PMLB MONK-2 contains 601 examples with six categorical attributes having 3, 3, 2, 3, 4, and 2 possible values, respectively (Olson et al., 2017). For each attribute, the distinguished value is encoded as 1. We retain the original binary target: an example is positive exactly when two of the six attributes equal 1.

D.8.1 Dataset Construction

Training set. We randomly select 10 base examples. We augment the 10 base examples with label-preserving transformations. Because the label depends only on how many attributes equal 1, two transformations preserve the label. The first permutes the non-1 values within each attribute. The second permutes attributes with the same cardinality: the three ternary attributes among themselves and the two binary attributes among themselves. Applying all combinations of these transformations to the 10 base rows and removing duplicates yields 212 unique training inputs.

Test set. The remaining 591 rows form the initial test set. We exclude the 288 rows whose inputs occur in the augmented training set, leaving 303 input-disjoint rows.

OOD test set. We construct the OOD test set by shifting the probability mass assigned to inputs with category-count compositions absent from the augmented training set from 18.48% in ID to 32.70% in OOD, while preserving the distribution of the number of attributes equal to 1. Because the label is positive exactly when two attributes equal 1, this also preserves the class proportions and the labeling rule. Specifically, we merge duplicate inputs in the 303 ID rows to obtain 220 unique inputs, assigning each input an initial probability proportional to its source frequency. For each input, we count how many attributes take each of the four categorical values. Within each group having the same number of attributes equal to 1, we multiply the probability of inputs whose count pattern is absent from the augmented training set by four and then renormalize the group.

D.8.2 Model Configuration

We encode each example with one classification token followed by six attribute tokens. Each token has a width-32 Type stream and a width-32 Data stream. In Type, the classification token activates only the first dimension. For each attribute token, the first dimension is zero, the next six dimensions encode the attribute identity, and the following four encode its observed categorical value. The remaining 21 Type dimensions are zero. In Data, each attribute token reserves the first seven dimensions for categorical one-hot encoding; only the first four of these dimensions can be nonzero at initialization for MONK-2. The remaining 25 dimensions are initialized to zero as scratch space. The classification token's Data is initialized entirely to zero.

The STRAT model has one four-head attention layer and a two-hidden-layer GLU with hidden widths 68 and 68. For the Transformer, we place the same observable fields in a width-64 residual stream. It uses one four-head layer and a two-hidden-layer FFN with hidden widths 68 and 68.

D.8.3 Training Configuration

We train for 2,000 epochs. For paired seeds 41–50, the final checkpoint provides accuracy on the test examples.

D.9 PMLB LED24

PMLB LED24 contains 3,200 examples with 24 binary features derived from a seven-segment display benchmark (Olson et al., 2017). The first seven features correspond to the display segments, and the remaining 17 are random nuisance features. We construct a binary classification task using this input space, assigning a positive label if and only if at least six of the seven segment features are active (i.e., equal to 1).

D.9.1 Dataset Construction

Training set. We select 10 examples, five per class, from the 3,200-example dataset using a greedy feature-coverage procedure with selection seed 3036. We augment each selected example by enumerating all seven cyclic shifts of its segment features and their reversals, each with and without simultaneous complementation of all 17 nuisance features. These transformations preserve the label because they leave the number of active segment features unchanged. After removing duplicates, we use the same augmented training set for both models.

Test set. The remaining 3,190 examples, labeled by the same rule, form the test set.

OOD test set. In the original 3,200 examples, between 1 and 15 of the 17 nuisance features are active; neither the all-zero nor the all-one configuration occurs. To construct the OOD test set, we retain positive examples with exactly six active segment features and 15 active nuisance features, and negative examples with exactly five active segment features and zero or one active nuisance feature. These conditions place the segment counts at the decision boundary and shift the nuisance counts to class-conditional extremes outside their ID ranges. We use a fixed hash ordering to select 250 examples per class, producing 500 unique inputs.

D.9.2 Model Configuration

We encode each example with one classification token followed by 24 feature tokens. Each token has 64 dimensions: the first 32 form Type, and the remaining 32 form Data. In Type, the first dimension identifies the classification token. For each feature token, one of the next two dimensions indicates whether it is a display-segment or nuisance feature, one of the following 24 dimensions identifies the feature, and one of the next two dimensions encodes its observed binary value. The remaining three Type dimensions are zero. The classification token activates only its own indicator. In Data, the first two dimensions encode the observed binary value of each feature token, while the remaining 30 dimensions are initialized to zero and serve as scratch space. The classification token's Data is initialized entirely to zero.

STRAT applies one four-head attention layers and a two-hidden-layer GLU with hidden widths 68 and 68. The Transformer concatenates the same observable fields into a width-64 residual stream. It uses one four-head layers and a two-hidden-layer FFN with hidden widths 68 and 68. Both models predict from the classification token.

D.9.3 Training Configuration

We augment the first seven fields with cyclic rotations and reflections. For each transformed example, we either retain or jointly complement the 17 irrelevant fields; both operations preserve the density target. Afterward, we remove

duplicates and give both models the same augmented set. We train for 2,000 epochs. For paired seeds 41–50, we report final checkpoint accuracy on the remaining 3,190 examples.

D.10 PMLB THREEOF9

PMLB ThreeOf9 contains all 512 assignments to nine binary variables (Olson et al., 2017). We use this complete input table for nine-bit majority classification. Positive labels require at least five active bits. We select 10 training examples.

D.10.1 DATASET CONSTRUCTION

Training set. We randomly pick 10 assignments.

Test set. The other 502 assignments form the test set.

OOD test set. We construct the OOD set by retaining assignments whose bits at positions 4, 6, and 9 contain at least two 1s for a negative label, or at most one 1 for a positive label. The true label remains positive when at least five of the nine input bits are 1, and negative otherwise. In the complete source distribution, predicting positive when these three bits contain at least two 1s and negative otherwise matches the label on 71.48% of inputs. In OOD, this rule is wrong on every input, reversing the feature–label association while preserving the task rule.

D.10.2 MODEL CONFIGURATION

We encode each example with one classification token followed by nine feature tokens. Each token has 64 dimensions: the first 32 form Type, and the remaining 32 form Data. In Type, the first dimension identifies the classification token. For each feature token, one of the next nine dimensions identifies the feature, and one of the following two dimensions encodes its observed binary value. The remaining 20 Type dimensions are zero. The classification token activates only its own indicator. In Data, the observed binary value of each feature token activates one of the first two dimensions. The remaining 30 dimensions are initialized to zero to zero and serve as scratch space. The classification token's Data is initialized entirely to zero.

The Transformer packs the same observable fields into a width-64 stream. Its single four-head layer uses an FFN with two width-68 hidden layers. Both models predict from the classification token.

D.10.3 TRAINING CONFIGURATION

Both models use the same 10 examples directly, without augmentation. We train for 2,000 epochs. For paired seeds 41–50, we evaluate the final checkpoint on the remaining 502 examples and report accuracy.

D.11 SPECT HEART

The UCI SPECT Heart dataset contains 267 patients represented by 22 binary features extracted from cardiac SPECT images (Cios et al., 2001). Its official split contains 80 training and 187 test examples. Our task uses the first nine features and predicts whether at least four are active.

D.11.1 DATASET CONSTRUCTION

Training set. We select 10 rows from the official 80-row training split, without augmentation.

Test set. We use the official 187-row test split as the source of the ID test set. We remove the 24 rows whose projected nine-bit inputs also occur in the training set, leaving 163 input-disjoint examples over 91 distinct inputs. Repeated source rows are retained, preserving their original frequencies.

OOD test set. We construct the OOD test set by shifting the input support from the 112 nine-bit configurations observed in the complete source dataset to configurations not observed in either official split. Specifically, we apply all nine cyclic rotations to each observed configuration and deduplicate the resulting union into 371 inputs. After removing the 112 configurations present in the source dataset, 259 previously unobserved inputs remain, each of which appears once in the OOD test set. Of these, 188 are high-density and 71 are low-density. Cyclic rotations preserve the number of active features and therefore preserve the four-of-nine labeling rule. The test set is OOD because every retained input lies outside the support observed across the complete source dataset, while the labeling rule remains unchanged.

D.11.2 Model Configuration

We encode each example with one classification token followed by nine feature tokens. Each token has 64 dimensions: the first 32 form Type, and the remaining 32 form Data. In Type, the first dimension identifies the classification token. For each feature token, one of the next nine dimensions identifies the feature, and one of the following two dimensions encodes its observed binary value. The remaining 20 Type dimensions are zero. The classification token activates only its own indicator. In Data, the observed binary value of each feature token activates one of the first two dimensions, while the remaining 30 dimensions are initialized to zero and serve as scratch space. The classification token's Data is initialized entirely to zero.

STRAT applies one four-head attention layer and a two-hidden-layer GLU with hidden widths 68 and 68. The Transformer concatenates the same observable fields into a width-64 residual stream. Its single four-head layer uses a two-hidden-layer FFN with hidden widths 68 and 68. Both models predict from the classification token.

D.11.3 Training Configuration

Both models train directly on the same 10 examples, without augmentation. We use full-batch training for 2,000 epochs.

E Comparison with State-of-the-Art Methods on Datasets Retaining Their Original Prediction Targets

We compare the STRAT results in Table 6 with published state-of-the-art results on five datasets retaining their original prediction targets; Appx. D gives the experimental details. Our augmented training data consist entirely of rule-preserving variants of the same **10 base examples per dataset**, providing variation within this small source set without introducing additional source instances.

E.1 SATNet Sudoku

The Recurrent Transformer of Yang et al. (2023) achieves 100% exact-board accuracy on 9×9 SATNet Sudoku (Wang et al., 2019), training on 9,000 puzzles and testing on 1,000. It applies a single four-head Transformer layer recurrently, with 32 steps during training and 64 at evaluation.

STRAT achieves $86.58 \pm 1.08\%$ exact-board accuracy (Table 6). Although this is 13.42 percentage points below the reported perfect score, STRAT trains on only 10 base puzzle–solution pairs, about 0.1% as many base puzzles, with rule-preserving augmentation (Appx. D). Our evaluation uses 1,090 held-out puzzles: the official 1,000 test puzzles plus 90 unused puzzles from the training split. Averaged over ten training seeds, STRAT outperforms our Transformer baseline ($27.92 \pm 4.97\%$), which receives the same training data and augmentation. STRAT also maintains $83.80 \pm 1.30\%$ accuracy on OOD puzzles with more unevenly distributed clues.

E.2 Tic-Tac-Toe

Transformer-based results on Tic-Tac-Toe Endgame are rarely reported as standalone accuracy. TabPFN (Hollmann et al., 2023a), a 12-layer in-context-learning Transformer, reaches 0.9759 ROC-AUC over five 50/50 splits of the 958 boards. CAAFE (Hollmann et al., 2023b) augments TabPFN with GPT-4-generated features and reports validation accuracy rising from 70.0% to 100% in a single illustrative run on a 95-instance subsample. Later work (den Breejen et al., 2024; Lee, 2026) reports only aggregate scores across benchmark suites. The best overall result, 1.000 ROC-AUC, comes from OpenFE with XGBoost (Zhang et al., 2023; Burghardt et al., 2026).

STRAT achieves $98.19 \pm 0.00\%$ mean accuracy (Table 6) from only 10 randomly selected base boards, expanded via label-preserving rotations and reflections to 76 training inputs (Appx. D). This is fewer than one-sixth of the 479 boards TabPFN conditions on. Averaged over ten independent seeds, STRAT outperforms our Transformer baseline($80.91 \pm 2.53\%$). On an OOD set whose spatial statistics oppose the training correlations, STRAT reaches $94.64 \pm 0.00\%$, while the Transformer falls to $60.49 \pm 5.47\%$.

E.3 Game of Life

Datasets similar to ours, which pair random toroidal Life boards with their one-step successors under the B3/S23 rule, have been used to evaluate Transformer architectures. On such data, LifeGPT (Berkovich & Buehler, 2025), a decoder-only GPT with 12 layers and 8 attention heads, trains on 10,000 random 32×32 boards and predicts the next state of 109 of 110 held-out broad-entropy boards perfectly. AutomataGPT (Berkovich et al., 2026) extends this setting to about one million trajectories spanning 100 cellular-automaton rules and reaches 98.5% one-step accuracy on unseen rules. On similar one-step Life data, minimal CNNs rarely converge (Springer & Kenyon, 2020), while a convolutional network with polynomial Kolmogorov–Arnold activations, trained on random 32×32 toroidal boards, succeeds in 128 of 128 training runs (Ahmed & Davis, 2026).

STRAT achieves $88.71 \pm 2.72\%$ exact-board accuracy (Table 6) on 757 held-out one-step transitions of 6×6 toroidal boards. It trains on only 10 base pairs, 0.1% of LifeGPT's training set, augmented with random toroidal translations, rotations, and reflections (Appx. D). Averaged over ten independent seeds, STRAT outperforms our Transformer baseline ($80.54 \pm 2.64\%$). On 500 OOD boards with fewer live–live adjacencies than any training board, STRAT reaches $79.26 \pm 12.99\%$, while the Transformer falls to $69.08 \pm 12.89\%$.

E.4 PMLB M-OF-N

The strongest reported result on PMLB M-of-N (3,7,10) is 100% test accuracy, achieved by the Chebyshev Adaptive Network (Islam et al., 2026). This MLP variant replaces connection weights with degree-k Chebyshev polynomials. For this dataset, Islam et al. (2026) utilize a degree-3 expansion on a network with a 4-2-output layer structure, reporting the best test accuracy across ten runs. Their dataset comprises 794 examples (roughly 60% of the 1,324-example PMLB release).

STRAT matches this $100.00 \pm 0.00\%$ perfect accuracy (Table 6) while using fewer than 9% as many examples as Islam et al. (2026). From only 10 randomly selected base examples, expanded via label-preserving cyclic shifts to 70 training inputs (Appx. D), STRAT achieves zero variance across ten independently trained models (seeds 41–50). Furthermore, STRAT demonstrates perfect out-of-distribution (OOD) generalization ($100.00 \pm 0.00\%$) on a test set that concentrates both classes at the decision boundary, with negatives having exactly two and positives exactly three active relevant features, an evaluation absent from Islam et al. (2026).

E.5 PMLB MONK-2

On PMLB MONK-2, Islam et al. (2026) report 100% test accuracy using the identical Chebyshev Adaptive Network architecture and best-of-ten evaluation protocol described above. Their dataset includes 360 examples (about 60% of the 601-example PMLB release).

STRAT achieves a comparable $90.86 \pm 3.04\%$ mean accuracy (Table 6). Our model is trained from only 10 randomly selected base examples, expanded to 212 unique inputs via label-preserving permutations (Appx. D). Averaged over ten independent seeds, STRAT successfully maintains its performance under distribution shift, reaching $90.92 \pm 3.07\%$ when the share of unseen category-count compositions increases from 18.48% to 32.70%.

F Additional Arithmetic Baselines

Sequence adaptations. We implement the Neural Accumulator (NAC) and Neural Arithmetic Logic Unit (NALU) modules of Trask et al. (2018) with a single-layer, 64-unit LSTM controller. At each token, affine heads predict the arithmetic matrices from the controller state. The controller receives a one-hot category (literal, plus, or minus) and the numerical value divided by 100; operator values are zero. The arithmetic cell receives the unscaled value and a two-dimensional numerical memory initialized to zero. Its first memory coordinate after the last token predicts the answer. Both models process every real token and freeze their states on padding, with supervision only on the final answer. Parameter counts are 18,700 for NAC and 19,090 for NALU.

Training and reporting. The task and training budget follow Section 6. We use full-batch Adam with MSE loss and clip gradient norms at 1. Learning rates are 0.03 for NAC and 0.01 for NALU, selected using development training loss. Both use float64 arithmetic and the final checkpoint. Each run is evaluated on 1,000 OOD expressions. NAC and NALU share per-run training and test samples. Table 12 includes all 400 runs per model. Median training MAEs are 0.34 and 4.02, respectively.

Table 12: Comparison with specialized arithmetic baselines. The Transformer and four STRAT-Lite rows reproduce Table 5; NAC and NALU are evaluated over 400 runs each. P10/P25 denote the 10th/25th percentiles of per-run OOD MAE; the last two columns give the percentage of runs below each error threshold.

	OOD MAE ↓			Runs below threshold ↑	
Model	P10	P25	Median	MAE < 20	MAE < 100
Baseline (Config A)	5,072	6,268	9,046	0%	0%
LSTM-controlled NAC	4,120	5,083	5,819	0%	0.25%
LSTM-controlled NALU	4,973	5,327	5,770	0%	0%
STRAT Flat (2D/1H)	33	80	293	5%	31%
STRAT Flat MH (2D/2H)	38	97	314	2%	27%
STRAT Compact (3D/1H)	34	77	266	3%	31%
STRAT Std (4D/1H)	**33**	**77**	**258**	4%	**31%**