One Model, Two Roles: Emergent Specialization in a Shared Recurrent Transformer

Jucheng Shen[*] Rice University
Barbara Su[*] Stanford University
Anastasios Kyrillidis[†] Rice University

Abstract

Can a shared-weight recurrent Transformer—one that repeatedly applies the same Transformer to update evolving latent states—develop distinct internal roles without being partitioned into separate modules? We study this in Asymmetric Input Recurrence (AIR), a minimal two-state reasoning architecture in which the same Transformer model is reused for both updates (per literature, L and H) and the only built-in difference in the update rule is that the encoded input is injected during L-updates but not H-updates. Across Sudoku-Extreme and Maze, decoded rollouts reveal a consistent split: z_H behaves like a fully committed proposal state, whereas z_L retains local uncertainty and shifting intermediate structure. Freeze interventions provide evidence of dynamical dependence between the two states, with responses that vary across tasks, intervention directions, and seeds. Ablations show that to induce specialization, the shared model needs to be able to tell the two update types apart, either from input injection asymmetry or from a separate level token. Mechanistically, attention analysis shows that L-updates are consistently more local than H-updates in both Sudoku and Maze. Together, these results show that, in a two-state recurrent setting, a clear state-identity signal can induce stable, related functional roles inside a shared-parameter recurrent Transformer. Code is available at `https://github.com/juchengshen/air`.

1 Introduction

Small recurrent reasoning models are emerging as a promising approach to parameter-efficient reasoning: reusing shared weights increases computational depth without increasing the parameter count. Concretely, a *recurrent Transformer* reuses the same Transformer block or stack across sequential computation steps to update persistent latent states, so effective computational depth can grow without allocating a separate parameter set at every step; a familiar example is the looped Transformer [Giannou et al., 2023], which applies the same block over and over to refine a single evolving latent sequence. HRM [Wang et al., 2025] and TRM [Jolicoeur-Martineau, 2025] demonstrate strong performance on structured abstract-reasoning benchmarks using two recurrent modules and a single shared-weight recursive network, respectively. Recent extensions strengthen this direction with stochastic recursive exploration in Probabilistic TRM [Sghaier et al., 2026] and GRAM [Baek et al., 2026], and constant-memory fixed-point refinement in Attractor Models [Fein-Ashley and Rashidinejad, 2026], yielding further gains on structured reasoning benchmarks. Inspired by these models, we raise the following question: *when the same model is reused for two latent states, does it learn distinct computational roles or merely process both states in the same way?* The answer matters because it tests whether specialization needs separate parameter models, or whether one shared computation can specialize when it gets a stable signal telling it which state it is updating.

To study this question, we introduce Asymmetric Input Recurrence (AIR), a minimal testbed that keeps TRM's shared recurrent model but uses HRM's one-step truncated gradient. We use this

[*]Equal contribution.
[†]Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

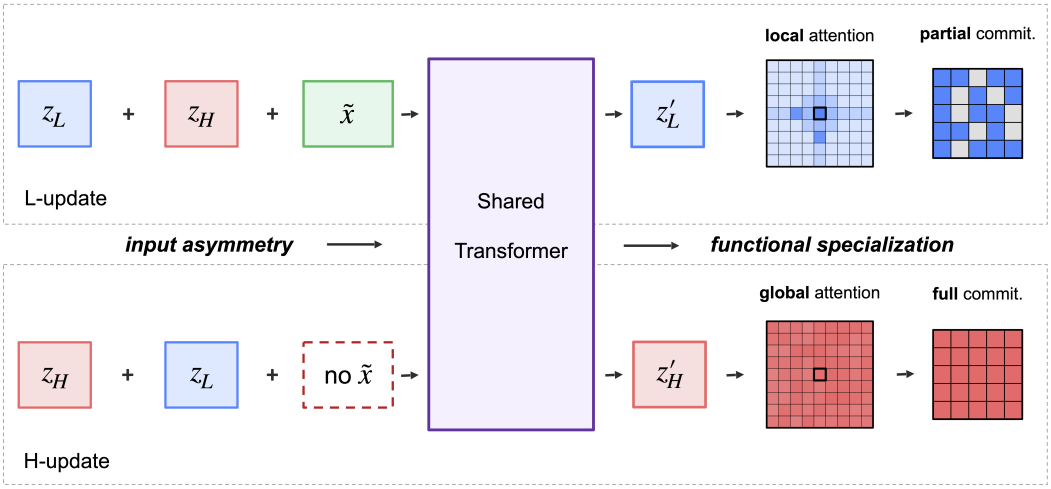


Figure 1: **AIR testbed.** The same shared Transformer $f(\cdot;\theta)$ is reused at two positions in the recurrent schedule: with the encoded input $\tilde{\mathbf{x}}$ for L-updates (top) and without it for H-updates (bottom). Despite identical parameters, this input difference lets the shared model tell the two update types apart: z_L behaves like a partially committed, revision-oriented state while z_H behaves like a committed proposal state, and each query token in L-updates attends to other tokens more locally than H-updates.

approximation for memory efficiency: backpropagating only through the final state updates avoids storing the full recurrent trajectory and keeps activation-memory cost approximately constant with recurrence depth [Wang et al., 2025, Jolicoeur-Martineau, 2025]. Stronger recurrent reasoners share this systems choice: GRAM [Baek et al., 2026] keeps HRM's truncated propagation and Attractor Models [Fein-Ashley and Rashidinejad, 2026] use constant-memory implicit gradients, so AIR's diagnostics yield hypotheses testable in these systems (Section 8). In AIR, the only built-in difference in the update rule is whether the encoded input $\tilde{\mathbf{x}}$ is injected (Figure 1), so AIR is a controlled setting for testing whether this single difference in update content can induce a division of labor between a high-level latent state z_H and a low-level latent state z_L.

Long story short, our answer is yes for the cases this paper studies. Across Sudoku-Extreme [Sapient Intelligence, 2024] and Maze [Sapient Intelligence, 2025], decoded rollouts show a stable split in which z_H behaves like a committed proposal state and z_L retains local uncertainty and shifting intermediate structure. We call z_H *fully committed* since it always decodes to a complete candidate, and z_L *partially committed* since some of its positions stay undecided.

Freeze experiments, used as an out-of-distribution stress test, show that the two states are dynamically interdependent (Section 6), and attention analysis reveals a matching mechanism: L-updates are more local and, on Sudoku, more violation-sensitive, whereas H-updates are more global (Section 7).

Our key contributions are as follows:

1. **Existence of functional specialization.** We demonstrate the existence of functional specialization in a shared-parameter recurrent Transformer. When decoding the intermediate output of each state, we observe that AIR develops a stable committed-z_H / partially uncommitted-z_L split, with a matching L-local versus H-global attention pattern (Sections 4 and 7).
2. **State-identity signal can induce functional specialization.** We empirically show that, under the training setting studied, functional specialization emerges when the shared model can tell the two update types apart—either through input injection asymmetry or through a separate level token (Section 5). Input asymmetry consistently improves Sudoku accuracy, while Maze shows a positive average trend.
3. **Two states are related.** Freeze interventions provide evidence of dynamical dependence between the two states, with responses that vary across tasks, intervention directions, and seeds (Section 6).

2 Related work

Iterative and recurrent reasoning. AIR belongs to work that uses recurrence or weight-sharing to turn small networks into multi-step reasoners. Universal Transformers [Dehghani et al., 2019] and later iterated models showed that weight sharing can support flexible computation and adaptive depth [Giannou et al., 2023, Schwarzschild et al., 2021, Banino et al., 2021]. Its closest predecessors are HRM [Wang et al., 2025] and TRM [Jolicoeur-Martineau, 2025]: HRM assigns the two latent states to different parameter models, whereas TRM ties them to one shared model. AIR keeps the shared-model setting but asks whether distinct roles can still emerge inside that shared recurrence when input reaches at least one state, and when the model can tell which state it is updating. Blayney et al. [2026] show that input re-injection can prevent degeneration in looped language models; we ask whether *asymmetric* input injection can also help a shared model separate two recurrent roles.

Modularity and emergent specialization. Our question is also about the distinction between *structural modularity* and *functional specialization*. Neural Module Networks [Andreas et al., 2016] and Recurrent Independent Mechanisms [Goyal et al., 2021] impose modular structure by design, whereas work on emergent modularity studies when differentiated behavior can arise inside initially monolithic models [Csordás et al., 2021, Tomoda et al., 2025]. AIR isolates this issue in a particularly strict setting: because the recurrent model is shared, any division of labor must emerge from training rather than architecture. Conditioning a shared network on a mode signal is also well established, e.g., conditional and unconditional roles in classifier-free guidance [Ho and Salimans, 2022] and task or target-language tokens in multitask and multilingual models [Johnson et al., 2017, Raffel et al., 2020]; there, however, each mode is defined and trained externally, whereas AIR trains on a single task with every training signal read from z_H: the update types can be told apart (by input injection or a level token), but nothing specifies what each state should compute. Thus our novelty is neither shared recurrence nor conditional computation by itself, but the controlled demonstration, ablation, and mechanistic characterization of a role split that emerges without any supervision of what each recurrent state should do.

Mechanistic analysis and latent workspaces. Methodologically, our paper is closest to mechanistic and perturbation-based analyses of reasoning models [Elhage et al., 2021, Olsson et al., 2022, Brinkmann et al., 2024, Hong et al., 2025], but we analyze recurrent latent states rather than individual heads or circuits. It also connects to work on internal workspaces for reasoning: unlike token-level scratchpads or chain-of-thought [Nye et al., 2022, Wei et al., 2022], AIR resembles latent-reasoning approaches such as COCONUT [Hao et al., 2025], with z_L behaving like a partial workspace and z_H like a more committed proposal state. The difference is that AIR does not prescribe these roles; it asks whether they emerge in a shared-weight recurrent Transformer when the model can tell the two states apart.

3 Preliminaries

We compare three closely related recurrent architectures: HRM, TRM, and our variant AIR. All maintain two latent states, $z_H, z_L \in \mathbb{R}^{B\times S\times D}$, where B is batch size, S is sequence length, and D is hidden dimension. Let $\tilde{\mathbf{x}}$ denote the encoded input.

HRM [Wang et al., 2025], TRM [Jolicoeur-Martineau, 2025], and AIR all use the following two-state recurrence in general; their specific differences are described below:

$$\textbf{L-update:} \quad z_L \leftarrow f_L(z_L + z_H + n_L\tilde{\mathbf{x}};\, \theta_L)\,, \tag{1}$$

$$\textbf{H-update:} \quad z_H \leftarrow f_H(z_H + z_L + n_H\tilde{\mathbf{x}};\, \theta_H)\,. \tag{2}$$

Here n_L and n_H are injection counts: the number of copies of $\tilde{\mathbf{x}}$ added in each L- and H-update. Unless otherwise stated, all three variants use $(n_L, n_H) = (1, 0)$, so the L-update receives $\tilde{\mathbf{x}}$ and the H-update receives no direct input. Each high-level cycle interleaves C_L L-updates with one H-update and repeats this pattern C_H times.

The first difference between HRM, TRM, and AIR is whether the L- and H-updates use one or two networks. HRM uses separate Transformer models, f_L and f_H, with parameter sets θ_L and θ_H. TRM and AIR tie the two updates into a single shared model, $f_L = f_H = f$. Like looped Transformers, all three models reuse the same computation at every recurrence step (the same category in the taxonomy of Mozer et al. [2026]); the difference is that a looped Transformer refines a single latent state,

whereas these models maintain two, z_L and z_H, and each update takes both as input (Equations (1) and (2)).

The second difference is how the recurrence is trained. HRM and AIR use a one-step truncated gradient [Williams and Peng, 1990]: gradients are computed through only one L-update and one H-update per training step. TRM trains by backpropagating through one full final recursion. HRM also uses ACT-style halting [Graves, 2016], but following the observation of Ge et al. [2025] that evaluation usually runs to the maximum cycle count, we do not analyze ACT here.

Notations. Unless otherwise stated, we decode intermediate states with the same output head used at test time. Throughout the rest of the paper, we refer to each individual L- and H-update as a *sub-step*. Following HRM, we use the default $C_L = C_H = 2$, giving the sub-step pattern **L, L, H, L, L, H** within each cycle. The first four sub-steps run without gradient; gradient is computed only through the final L-update and the final H-update. All experiments use Sudoku-Extreme (9×9 puzzles with 17 givens) and Maze (30×30 grids); full implementation and training details are in Appendix A.

4 Emergent functional specialization

This section asks whether AIR's minimal asymmetry—input injection in L-updates but not H-updates—can produce distinct computational roles. We test this directly by decoding both latent states, z_H and z_L, at every sub-step with the same output head used at test time.

The asymmetric AIR rollouts reveal the same qualitative split on both tasks. z_H is *fully committed*: at every sub-step it decodes to a complete candidate solution, even when that candidate is still wrong. z_L, by contrast, is *partially committed*: some positions remain undecided, and the locations of those undecided positions move across sub-steps. For Sudoku, the undecided symbol is the dedicated `BLANK` token; for Maze, it is the analogous `PAD` token. Both commitment terms describe only how the decoded states look: since only z_H is directly supervised and z_L is read through the same output head, they do not by themselves establish what computation each state performs. Figures 2 and 3 visualize this split. These panels show a representative rollout from one checkpoint per task; inspecting decoded rollouts from all five independently trained checkpoints per task, we observe the same qualitative split in every seed.

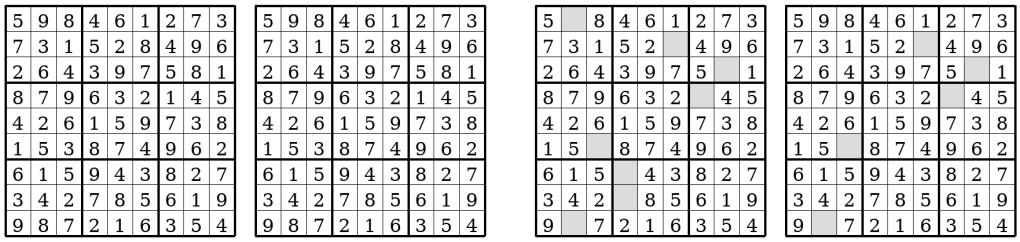


Figure 2: **Sudoku decoded states.** Left = z_H, Right = z_L; z_H stays fully committed, while z_L keeps some cells as `BLANK` (shaded light gray) and shifts those blanks across sub-steps.



Figure 3: **Maze decoded states.** Left = z_H, Right = z_L; **Black** = wall, white = open path, **blue** = solution path, **gray** = undecided (`PAD`), **red** = goal, and **green** = start. z_H stays committed, while z_L shows **gray** undecided cells and other shifting local structure. Multiple cells are decoded as start or goal in the z_L panels, which may reflect endpoint information that is not yet stably localized in these intermediate states.

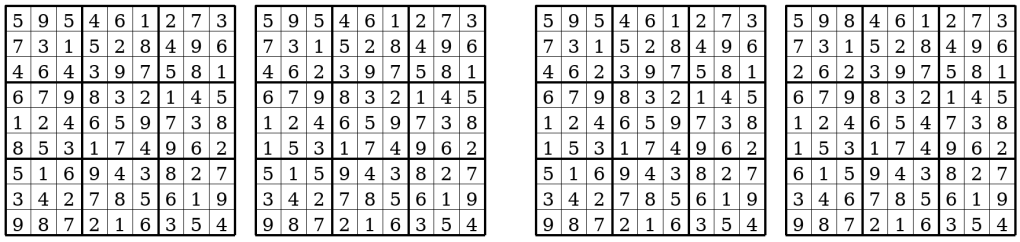


Figure 4: **Sudoku decoded states for the symmetric `Lx_Hx` variant.** Left = z_H, Right = z_L; decoded intermediate states across two sub-steps for each update type. Relative to Figure 2, decoded z_L no longer shows visible `BLANK` cells.



Figure 5: **Maze decoded states for the symmetric `Lx_Hx` variant.** Left = z_H, Right = z_L; decoded intermediate states across two sub-steps for each update type. Relative to Figure 3, decoded z_L no longer shows visible `PAD` / undecided regions.

The symmetric `Lx_Hx` control removes this decoded role split. Figures 4 and 5 repeat the same four-panel view for symmetric models on Sudoku and Maze. Once the asymmetric input difference is removed, decoded z_L no longer shows the visible `BLANK` / `PAD` undecided regions that characterize AIR's partially committed state, and decoded z_H no longer reaches the same degree of early commitment seen in the asymmetric AIR rollouts.

Together, the asymmetric and symmetric rollouts show that functional specialization is not simply a consequence of carrying two recurrent states. The same shared Transformer develops a proposal-like z_H and a locally uncertain z_L when the update types are distinguishable through input injection, but this decoded division nearly vanishes when both update types receive the same input treatment. This comparison motivates the ablations and mechanisms studied next: specialization emerges when the shared model has a reliable state-identity signal for the two latent update types.

5 Input injection and state identity

Section 4 showed that AIR develops distinct z_H/z_L roles. This section asks what signal allows a single shared model to maintain that split. Our hypothesis is simple: the shared model must have a way to tell which state it is updating. Since the only visible difference between the two update types are whether input $\tilde{\mathbf{x}}$ is injected, it is reasonable to test how many copies of $\tilde{\mathbf{x}}$ are added to each update type. We parameterize each configuration by the injection counts (n_L, n_H) in Equations (1) and (2), and define the asymmetry level as $\Delta = |n_L - n_H|$.

Asymmetric input injection can induce functional specialization. Table 1 shows a consistent accuracy advantage on Sudoku and a positive average trend on Maze. On Sudoku, asymmetric configurations ($\Delta > 0$) cluster around 59%, whereas symmetric ones ($\Delta = 0$) fall to about 51%, an $\sim$8-point drop. All 12 asymmetric-versus-symmetric Sudoku comparisons remain significant after Holm correction. On Maze, the descriptive group means are 73.1% versus 69.9%, but only one of 12 comparisons survives correction (L_H2x versus Lx_Hx, adjusted $p = 0.0318$). One thing to note is that the best AIR variants are also comparable to—and on Sudoku even exceed—the original two-network HRM baseline (55.0% on Sudoku, 74.5% on Maze) with only 50% of the latter's parameters, demonstrating the promise of this "one model, two roles" paradigm when designing efficient future LLM architectures. Detailed significance tests for Table 1 can be seen in Appendix Section B.

Table 1: Injection asymmetry ablation on Sudoku-Extreme and Maze. Exact-solve accuracy, mean ± SD over five independently trained seeds per shared-model variant.

Variant	(n_L, n_H)	Δ	**Sudoku (%)**	**Maze (%)**
Asymmetric ($\Delta > 0$)				
L_Hx	(0, 1)	1	$58.7_{\pm 3.3}$	$75.3_{\pm 3.2}$
Lx_H (default)	(1, 0)	1	$60.0_{\pm 2.0}$	$71.0_{\pm 6.3}$
L_H2x	(0, 2)	2	$58.6_{\pm 1.9}$	$75.6_{\pm 1.9}$
L2x_H	(2, 0)	2	$59.1_{\pm 2.4}$	$71.1_{\pm 6.4}$
Lx_H2x	(1, 2)	1	$59.6_{\pm 0.9}$	$70.9_{\pm 2.4}$
L2x_Hx	(2, 1)	1	$58.6_{\pm 2.9}$	$74.5_{\pm 1.6}$
Approx. group mean			59.1	73.1
Symmetric ($\Delta = 0$)				
Lx_Hx	(1, 1)	0	$52.1_{\pm 1.6}$	$69.4_{\pm 2.5}$
L2x_H2x	(2, 2)	0	$50.9_{\pm 2.9}$	$70.3_{\pm 4.2}$
Approx. group mean			51.5	69.9
Two-network baseline (separate parameters)				
HRM [Wang et al., 2025]	—	—	55.0	74.5

Level-token control. A natural alternative explanation is that AIR does not need asymmetric *input* specifically; perhaps input can be injected symmetrically as long as a separate signal tells the model which state it is updating. To test this, we start from the symmetric Lx_Hx baseline ($(1, 1)$, $\Delta = 0$), which preserves L's single input copy and gives H the same input, and add learned level tokens in three ways:

(i) **Addition.** A learned vector τ_L or τ_H is added element-wise to every token before each sub-step.

(ii) **Prepend (strip).** A learned level token is prepended at position 0, participates in self-attention during the forward pass, and is then stripped from the output after the final Transformer layer. In the next sub-step, the same level token is prepended again and repeat.

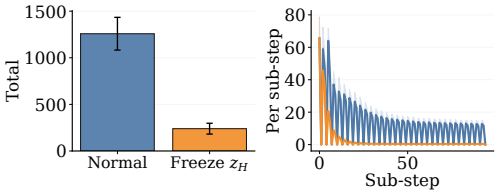
(iii) **Prepend (no strip).** The same prepended token is not stripped after the final Transformer layer; it stays at the beginning of the sequence across cycles.

Table 2: Level-token controls on Sudoku-Extreme with symmetric Lx_Hx injection. Mean ± sample SD over five seeds, using each run's best evaluation within 20,000 epochs.

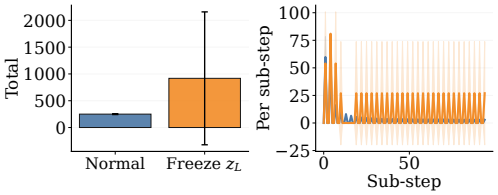
Level-token strategy	**Mechanism**	**Acc. (%)**
Lx_Hx (base, no token)	—	$52.07_{\pm 1.62}$
+ Addition	element-wise add	$50.91_{\pm 3.02}$
+ Prepend (strip)	prepend, attend, strip	$58.23_{\pm 1.56}$
+ Prepend (no strip)	prepend, persist	$55.34_{\pm 5.52}$
For reference: asymmetric $\Delta = 1$ variants		$\sim$59.0

In Table 2, prepend (strip) raises exact solve from 52.07% to 58.23%. We compare each seed's best evaluation over ten evenly-spaced checkpoints using two-sided Welch tests, with Holm correction across all six pairwise comparisons, and report 95% confidence intervals for each difference in accuracy points. Strip significantly outperforms Base and Addition ($p_{\text{adj}} = 0.00168$ and 0.0149), while Addition does not differ significantly from Base ($p_{\text{adj}} = 0.788$). No-strip reaches $55.34 \pm 5.52\%$ but does not differ significantly from any other control ($p_{\text{adj}} = 0.788$, 0.659, and 0.788 against Base, Addition, and Strip); all six differences and their 95% confidence intervals are in Table 5.

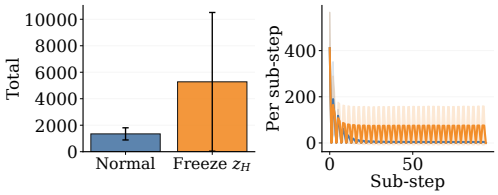
Thus, a transient, structurally separable level token supports recovery under symmetric input injection, whereas Addition shows no detected improvement. Addition mixes identity into every content token; no-strip allows its dedicated slot to accumulate recurrent content, but its higher variance leaves the comparison with Strip inconclusive. A more detailed interpretation of why addition fails while



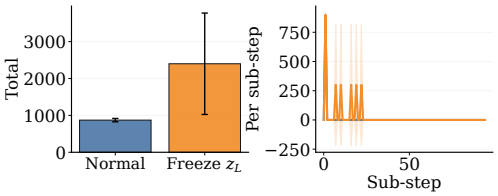
(a) Sudoku: freezing z_H reduces z_L's changes.



(b) Sudoku: freezing z_L has a seed-dependent effect on z_H's changes.



(c) Maze: freezing z_H increases z_L's changes, with substantial variation across seeds.



(d) Maze: freezing z_L increases z_H's changes.

Figure 6: Paired freeze experiments on AIR (asymmetric `Lx_H`) measured by content changes. Bars show total content changes per puzzle and curves show per-sub-step changes, both as mean ± sample SD across three independently trained seeds (1,000 evaluated puzzles per seed).

prepend (strip) works is deferred to Appendix C.1. Longer-training curves and token-consistency analysis are provided in Appendix Section C.2.

Operator-form control. We also test linear, nonlinear, sign-flip, and Hadamard transformations of $\tilde{\mathbf{x}}$ before injection; on Sudoku, none differs significantly from additive injection after Holm correction across four comparisons (details in Appendix D).

6 State-dependence analysis

In this section, we analyze state dependence: we test whether the two states in AIR depend on each other dynamically by applying paired freeze experiments: hold z_H fixed while z_L updates, or hold z_L fixed while z_H updates, and measure how the unfrozen state changes.

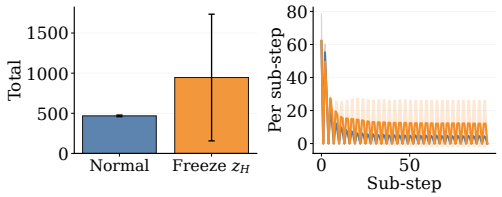
Metric: content change. We quantify state dependence by *content change*, the number of positions whose decoded token changes between consecutive sub-steps. This measures how much one state keeps revising its decoded proposal when the other state is prevented from updating. Writing $\hat{y}_i^{(t)}$ for the decoded token at position i after sub-step t, we define content change $C^{(t)}$ as

$$C^{(t)} = \left|\left\{i : \hat{y}_i^{(t)} \neq \hat{y}_i^{(t-1)}\right\}\right|.$$

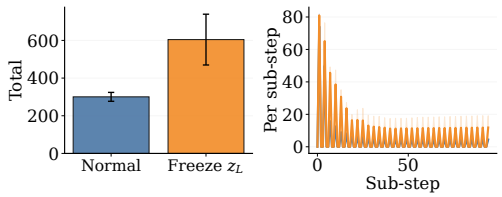
All content-change plots report mean ± sample standard deviation across three seeds, with 1,000 evaluated puzzles per seed; per-seed results are in Appendix H.

Sudoku: freezing the two states has different effects. In Sudoku, freezing either latent state's update disrupts the other's normal dynamics, with responses that depend on the frozen state and the seed. Freezing z_H sharply suppresses z_L's activity, reducing total content changes from 1258.2 to 239.5 and lowering the per-sub-step change after the first few steps (Figure 6a). Freezing z_L produces heterogeneous responses in z_H: total content changes increase on average from 250.1 to 918.0, but this increase is driven by one of three seeds, while the other two show decreases (Figure 6b).

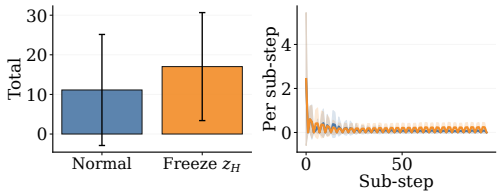
Maze: freezing one state increases content changes in the other. Maze shows the same state interdependence with the opposite pattern: freezing one latent state's update increases the other's content changes. Freezing z_H increases z_L's total content changes in all three evaluated seeds, from 1345.8 to 5277.1 on average, although the magnitude varies substantially and the mean is strongly influenced by one seed (Figure 6c). Freezing z_L increases z_H's content changes from 871.9 to 2400.0 (Figure 6d).



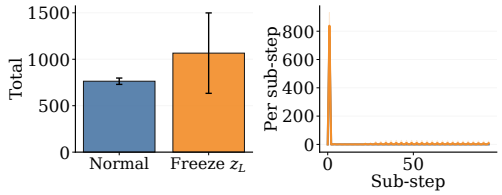
(a) Sudoku: freeze z_H in the symmetric `Lx_Hx` model.



(b) Sudoku: freeze z_L in the symmetric `Lx_Hx` model.



(c) Maze: freeze z_H in the symmetric `Lx_Hx` model.



(d) Maze: freeze z_L in the symmetric `Lx_Hx` model.

Figure 7: Freeze experiments on the symmetric `Lx_Hx` control on Sudoku and Maze, measured by content changes as mean $\pm$ sample SD across three independently trained seeds (1,000 evaluated puzzles per seed).

The symmetric `Lx_Hx` control responds differently (Figure 7). On Sudoku, freezing either state increases content changes relative to normal on average. On Maze, the two directions are qualitatively mismatched: freeze-z_H produces near-zero content-change activity, while freeze-z_L produces a large first-step spike. Thus the symmetric states are also perturbation-sensitive but do not reproduce AIR's task-dependent freeze-z_H pattern.

Effect on task accuracy. Freezing either z_H or z_L also collapses final accuracy to 0% on both tasks; on the same 1,000-puzzle evaluation subset, the unfrozen setting reaches 55.1% on Sudoku and 71.0% on Maze.

7 Mechanism: local versus global attention

Section 4 showed that AIR develops distinct latent-state roles, and Section 6 showed that the two states are dynamically interdependent. Here we ask whether this split appears in attention. We test whether L-updates attend more locally than H-updates, and whether L-updates place extra attention on currently violated cells. On Sudoku, both effects appear: the local/global split is present at every layer, while violation-specific attention appears mainly in deeper layers. On Maze, we find the local/global split but not a separate violation-specific signal.

7.1 Setup

Cells, neighborhoods, and violations. A *query cell* q is any blank cell. For Sudoku, its *neighborhood* $\mathcal{N}(q)$ is the 20 cells sharing a row, column, or 3×3 box with q. A cell is *violated* if its currently decoded value from z_H clashes with another cell in its row, column, or box; let $\mathcal{V}$ denote the set of such cells. We call a query *violation-adjacent* if $\mathcal{N}(q) \cap \mathcal{V} \neq \emptyset$, and *control* otherwise. For Maze, we use the analogous setup on the 30×30 grid. We define $\mathcal{N}_4(q)$ as the four cardinal neighbors of q, $\mathcal{N}_8(q)$ as the eight surrounding neighbors, and $\mathcal{N}_{5\times5}(q)$ as the centered 5×5 window around q excluding q itself. Since these neighborhoods have different sizes, we use the per-cell density contrast $\Delta\rho^{(k)}$, which measures the L$-$H attention-mass difference inside $\mathcal{N}_k(q)$ divided by $|\mathcal{N}_k(q)|$. Maze queries are *error-adjacent* if $\mathcal{N}_4(q)$ contains a cell where decoded z_H disagrees with the ground truth, and *control* otherwise. Details are in Appendices F and G for Sudoku and Maze respectively.

What we measure. We compare attention during the first L-update and first H-update at cycles $\{2, 4, 6, 8, 10, 12, 14, 15\}$, averaging over the 8 heads at each layer. For Sudoku, we report three L$-$H contrasts: Δ_{nbr}, the difference in attention mass on the neighborhood (nbr) $\mathcal{N}(q)$; Δ_{ent}, the H$-$L difference in attention entropy, so positive values mean L is more concentrated; and Δ_{viol}, the

difference in attention mass on $\mathcal{N}(q) \cap \mathcal{V}$. For Maze, we report $\Delta\rho^{(k)}$, Δ_{ent}, and Δ_{viol} analogously. Full per-layer values are in Appendix F.4 for Sudoku and Appendix G.4 for Maze.

7.2 Results

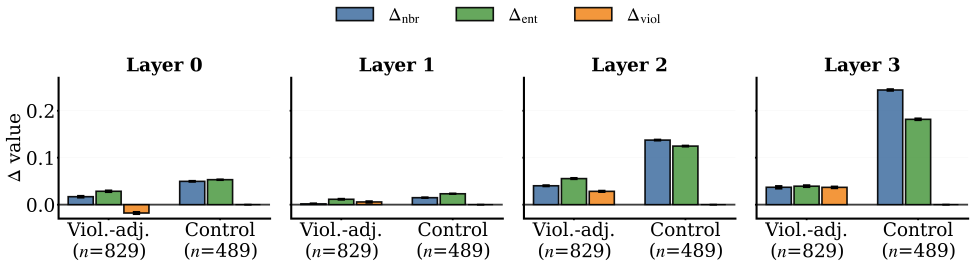


Figure 8: Sudoku L−H attention contrasts across query classes and Transformer layers. L-updates are more local than H-updates ($\Delta_{\text{nbr}}, \Delta_{\text{ent}} > 0$), while violation-specific concentration ($\Delta_{\text{viol}} > 0$) appears mainly in deeper layers. Δ_{viol} is zero on control queries by construction, because $\mathcal{N}(q) \cap \mathcal{V} = \emptyset$ for them.

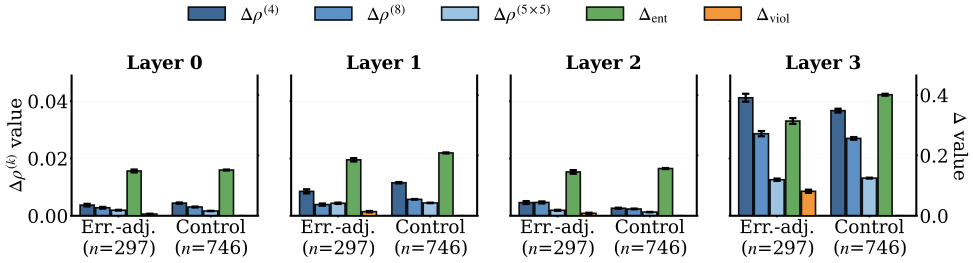


Figure 9: Maze attention contrasts across error-adjacent and control queries. We report per-cell density contrast $\Delta\rho^{(k)}$ for $k \in \{4, 8, 5{\times}5\}$, together with Δ_{ent} and Δ_{viol} over all four layers. Δ_{viol} is zero on control queries by construction, because $\mathcal{N}_4(q) \cap \mathcal{V} = \emptyset$ for them.

L-updates are consistently more local than H-updates. On Sudoku, both Δ_{nbr} and Δ_{ent} are positive across query classes and layers (Figure 8). Thus L places more attention inside $\mathcal{N}(q)$ and has a more concentrated distribution than H. The gap is largest at the deepest layer: on control cells, $\Delta_{\text{nbr}} = 0.24$, roughly 47% more attention mass inside $\mathcal{N}(q)$ for L than for H.

Maze shows the same pattern under the size-invariant density measure. For every neighborhood size k, layer, and query class, $\Delta\rho^{(k)}$ is positive (Figure 9), meaning L places more attention near the query than H. The effect is strongest for the smallest neighborhood and weakens as the neighborhood expands, indicating that L's extra attention is concentrated closest to the query. The entropy contrast agrees: $\Delta_{\text{ent}} > 0$ at every layer for both classes, so L attention is more concentrated over the full 900-cell grid. Representative attention maps are in Appendices F.2 and G.2 for Sudoku and Maze respectively.

Violation-specific routing appears on Sudoku but not on Maze. On Sudoku, Δ_{viol} is negative at layer 0 (-0.018), small at layer 1 ($+0.006$), and positive by layers 2–3 ($+0.028$ and $+0.037$). Since L is already more local overall, this could reflect locality rather than sensitivity to violations. To isolate violation sensitivity, we restrict attention to the same Sudoku neighborhood $\mathcal{N}(q)$ and ask whether L puts disproportionately more attention on violated cells than H does. Specifically, within $\mathcal{N}(q)$, we compare the average attention per cell on violated cells, $\mathcal{N}(q) \cap \mathcal{V}$, against the average attention per cell on non-violated cells, $\mathcal{N}(q) \setminus \mathcal{V}$. This control is near zero at layers 0–1 and positive at layers 2–3 ($+0.0025$ and $+0.0046$), showing that the deeper-layer signal exceeds locality alone.

On Maze, the analogous within-neighborhood control inside $\mathcal{N}_4(q)$ stays near zero at every layer. Thus L's extra mass on Maze error cells is explained by locality, not by a separate violation-specific routing signal.

Causal attention interventions. To test whether attention locality causally contributes to solving, we additionally intervene on attention at inference across five independently trained checkpoints per task. These interventions show that L's neighborhood access supports solving on both tasks, while H's access beyond the neighborhood supports whole-maze solving. The protocols, controls, and results are provided in Appendix Sections F.5 and G.5 for Sudoku and Maze, respectively. Appendix I discusses possible explanations for these differences.

8 Discussion and conclusion

This paper asked whether a shared-weight recurrent Transformer can develop distinct roles for two latent states when the only built-in asymmetry is which update receives the input. On Sudoku-Extreme and Maze, the answer is yes: decoded rollouts show a committed z_H and a revision-oriented z_L, L-updates attend more locally than H-updates, and freeze experiments show that the two states are dynamically interdependent. Injection asymmetry also gives a consistent accuracy advantage on Sudoku, with more limited, configuration-dependent evidence on Maze.

Taken together, these results suggest that specialization does not require separate parameters. What it requires is a clear signal of which state is being updated, which can come either from asymmetric input injection or from a structurally separable level token.

Looking ahead, stronger two-state recurrent reasoners with approximately constant-memory training, such as GRAM [Baek et al., 2026] and Attractor Models [Fein-Ashley and Rashidinejad, 2026], are a natural place to test whether these findings carry over. The same diagnostics—intermediate decoding, attention routing, and causal attention interventions—can check whether their states likewise differ in commitment, whether their input-receiving updates attend more locally, and whether these differences matter causally.

9 Limitations and broader impact

All experiments are conducted on two synthetic, fully observable, grid-based combinatorial tasks, so our conclusions should be interpreted as evidence that asymmetric specialization can emerge in a controlled setting, not yet as a claim of universality. We study only a 4-layer shared Transformer with two recurrent latent states and default $C_L = C_H = 2$ updates per cycle; our mechanistic evidence is strongest on Sudoku, and attention weights remain only a partial view of computation. We train only with HRM's one-step gradient and have not tested whether state identity has the same effect under TRM-style backpropagation. Whether the same role split persists under deeper blocks, different schedules, natural-language reasoning, continuous domains, or partial observability is left to future work.

This work is foundational research on the interpretability and inductive biases of small recurrent reasoning systems trained on synthetic tasks. We do not foresee direct harmful societal impact from these models, though understanding recurrent specialization may inform more efficient and interpretable reasoning systems.

Acknowledgments and Disclosure of Funding

This research was funded in part by Rice University, the U.S. National Science Foundation (CAREER award no. 2145629), and a Microsoft Research Award. AK also thanks the Ken Kennedy Institute at Rice University for its support through the research cluster QuanTAS. We thank the anonymous reviewers and the area chair for comments that improved this paper. The authors declare no competing interests.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016.

Junyeob Baek, Mingyu Jo, Minsu Kim, Mengye Ren, Yoshua Bengio, and Sungjin Ahn. Generative recursive reasoning. *arXiv:2605.19376*, 2026.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning*, 2021.

Hugh Blayney, Álvaro Arroyo, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, Michael M. Bronstein, and Xiaowen Dong. A mechanistic analysis of looped reasoning language models. *arXiv:2604.11791*, 2026.

Jannik Brinkmann, Abhay Sheshadri, Victor Levoso, Paul Swoboda, and Christian Bartelt. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. In *Findings of ACL*, pages 4082–4102, 2024.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? Inspecting functional modularity through differentiable weight masks. In *ICLR*, 2021.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *ICLR*, 2019.

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.

Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. In *ICML*, 2024.

Jacob Fein-Ashley and Paria Rashidinejad. Solve the loop: Attractor models for language and reasoning. *arXiv:2605.12466*, 2026.

Renee Ge, Qianli Liao, and Tomaso Poggio. Hierarchical reasoning models: Perspectives and misconceptions. *arXiv:2510.00355*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *ICML*, 2023.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *ICLR*, 2021.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv:1603.08983*, 2016.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *COLM*, 2025.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv:2207.12598*, 2022.

Guan Zhe Hong, Nishanth Dikkala, Enming Luo, Cyrus Rashtchian, Xin Wang, and Rina Panigrahy. A implies B: Circuit analysis in LLMs for propositional logical reasoning. In *NeurIPS*, 2025.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv:2510.04871*, 2025.

Michael C. Mozer, Shoaib Ahmed Siddiqui, and Rosanne Liu. The topological trouble with transformers. *arXiv:2604.17121*, 2026.

Maxwell Nye, Anders Johan 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. In *ICLR 2022 Workshop on Deep Learning for Code*, 2022.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022.

Lucas Prieto, Melih Barsbey, Pedro A. M. Mediano, and Tolga Birdal. Grokking at the edge of numerical stability. In *ICLR*, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Sapient Intelligence. Sudoku-Extreme dataset. `https://huggingface.co/datasets/sapientinc/sudoku-extreme`, 2024.

Sapient Intelligence. Maze 30×30 hard 1k dataset. `https://huggingface.co/datasets/sapientinc/maze-30x30-hard-1k`, 2025.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. In *NeurIPS*, 2021.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. *arXiv:2605.19943*, 2026.

Yuki Tomoda, Ichiro Tsuda, and Yutaka Yamaguti. Emergence of functionally differentiated structures via mutual information minimization in recurrent neural networks. *Cognitive Neurodynamics*, 20 (1):5, 2025. doi: 10.1007/s11571-025-10377-0.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv:2506.21734*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.

Appendix

A Experimental details . . . 14

B Significance tests for Table 1 . . . 14

C Level-token analysis . . . 15

C.1 Level-token interpretation . . . 15

C.2 Longer training and token consistency . . . 16

C.3 Pairwise tests for the level-token controls . . . 17

D Operator-form control . . . 17

E Single-state ablation . . . 18

F Attention analysis on Sudoku . . . 19

F.1 Setup . . . 19

F.2 Representative examples . . . 19

F.3 Temporal persistence . . . 19

F.4 Layer robustness . . . 21

F.5 Causal attention interventions . . . 21

G Attention analysis on Maze . . . 23

G.1 Setup . . . 23

G.2 Representative examples . . . 23

G.3 Temporal persistence . . . 23

G.4 Layer robustness . . . 25

G.5 Causal attention interventions . . . 25

H Per-seed freeze results . . . 26

I Interpretation of task and layer differences . . . 29

A Experimental details

Architecture. All experiments use a single shared Transformer block with 4 layers, 8 attention heads, hidden dimension $D = 512$, and MLP expansion factor 4. Rotary position embeddings (RoPE) are used. The input encoder maps discrete tokens to D-dimensional embeddings. The same output head (a linear projection followed by softmax over the token vocabulary) is used for both training supervision and intermediate-state decoding. The output head is applied to z_H, so the training loss directly supervises z_H only; z_L is shaped indirectly through the recurrence. Before the first cycle of each new input, z_H and z_L are initialized to broadcast copies of fixed vectors $\mathbf{h}_0, \mathbf{l}_0 \in \mathbb{R}^D$ drawn once at construction from a truncated normal distribution with standard deviation 1 (values outside $[-2, 2]$ resampled).

Recurrence structure. Each high-level cycle interleaves C_L L-updates with one H-update, repeated C_H times, giving the pattern (L$\times C_L$, H)$\times C_H$. With the default $C_L = C_H = 2$ this is **L, L, H, L, L, H**—six sub-steps per cycle. The first four sub-steps run without gradient; gradient is computed only through the final L-update and the final H-update of each cycle (two updates), so that backpropagation traverses only the two last applications of the shared block in each cycle. An ACT-style halting mechanism [Graves, 2016] determines the number of high-level cycles per input, with a maximum of 16 cycles and a halt exploration probability of 0.1.

Tasks. *Sudoku-Extreme* uses 9×9 Sudoku puzzles with 17 given cells (the minimum for a unique solution), represented as a sequence of 81 tokens from a vocabulary of $\{1, \ldots, 9, \texttt{BLANK}\}$. The `BLANK` token represents an unfilled cell. *Maze* uses 30×30 grid mazes, where the network predicts a token for each cell indicating wall, path, or solution. The `PAD` token serves as the undecided marker analogous to `BLANK` in Sudoku.

Training. Following the HRM training recipe [Wang et al., 2025], all models are trained with AdamATan2 [Everett et al., 2024] ($\beta_1 = 0.9$, $\beta_2 = 0.95$) using a learning rate of 1×10^{-4}, weight decay 1.0, and global batch size 768. The learning rate warms up over 2,000 steps. The loss function is stablemax cross-entropy [Prieto et al., 2025], combined with HRM's Q-learning halting loss; both losses are computed from z_H. The gradient update follows HRM's one-step truncated backpropagation [Williams and Peng, 1990]: the network performs every sub-step of the cycle, but the gradient is computed only through the final L-update and the final H-update (no unrolling through earlier sub-steps). Sudoku models are trained for 20,000 epochs; Maze models use the same procedure. Training data consists of 1,000 puzzles with 1,000$\times$ augmentation. All accuracy numbers report peak / early-stop exact-match accuracy on a held-out test set.

Error bars. Unless otherwise stated, evaluations use the full held-out test splits: 422,786 puzzles for Sudoku and 1,000 puzzles for Maze. Unless otherwise stated, accuracy and ablation tables report mean $\pm$ standard deviation across 5 independent training runs with random seeds 0 through 4. Freeze-experiment plots report mean $\pm$ sample standard deviation across three seeds, using 1,000 puzzles per seed (seeds 0–2, except the symmetric Maze control, which uses seeds 2–4). Attention-analysis figures and tables report puzzle-level mean $\pm$ 95% confidence intervals over the analyzed test puzzles.

Compute. Each Sudoku training run takes approximately 4 hours on a single NVIDIA H200 GPU. Each Maze training run takes approximately 3 hours on 8 NVIDIA H200 GPUs. The total compute for the experiments reported in this paper, including ablations and preliminary runs, is approximately 500 GPU-hours.

B Significance tests for Table 1

Testing protocol. We compare each of the six asymmetric configurations in Table 1 with each of the two symmetric configurations using two-sided Welch t-tests computed from the reported means and standard deviations, with $n = 5$ independent training seeds per variant. Each task defines a separate family of 12 comparisons, to which we apply Holm correction at $\alpha = 0.05$. Differences are asymmetric minus symmetric accuracy, in percentage points (pp). The 95% confidence intervals are individual Welch intervals, without multiplicity adjustment; significance is determined by the Holm-adjusted p-value.

Table 3: Asymmetric versus symmetric Sudoku variants from Table 1.

Asymmetric	Symmetric	Difference	95% CI	p	Holm p	Significant?
L_Hx	Lx_Hx	6.6	$[2.55, 10.65]$	0.00748	0.0127	Yes
L_Hx	L2x_H2x	7.8	$[3.26, 12.34]$	0.00426	0.0127	Yes
Lx_H	Lx_Hx	7.9	$[5.24, 10.56]$	0.000156	0.00171	Yes
Lx_H	L2x_H2x	9.1	$[5.39, 12.81]$	0.000645	0.00580	Yes
L_H2x	Lx_Hx	6.5	$[3.93, 9.08]$	0.000426	0.00426	Yes
L_H2x	L2x_H2x	7.7	$[4.02, 11.38]$	0.00170	0.00986	Yes
L2x_H	Lx_Hx	7.0	$[3.95, 10.05]$	0.000995	0.00796	Yes
L2x_H	L2x_H2x	8.2	$[4.29, 12.11]$	0.00137	0.00957	Yes
Lx_H2x	Lx_Hx	7.5	$[5.51, 9.49]$	0.0000728	0.000874	Yes
Lx_H2x	L2x_H2x	8.7	$[5.16, 12.24]$	0.00164	0.00986	Yes
L2x_Hx	Lx_Hx	6.5	$[2.91, 10.09]$	0.00423	0.0127	Yes
L2x_Hx	L2x_H2x	7.7	$[3.47, 11.93]$	0.00300	0.0120	Yes

Sudoku shows a consistent asymmetric-versus-symmetric separation. The 12 Sudoku comparisons in Table 3 yield positive differences of 6.5–9.1 percentage points, with all individual 95% confidence intervals above zero. All 12 comparisons remain significant after Holm correction within the Sudoku family (largest adjusted $p = 0.0127$). Thus, Sudoku provides statistically significant support for the asymmetric-versus-symmetric accuracy separation.

Table 4: Asymmetric versus symmetric Maze variants from Table 1.

Asymmetric	Symmetric	Difference	95% CI	p	Holm p	Significant?
L_Hx	Lx_Hx	5.9	$[1.67, 10.13]$	0.0127	0.1267	No
L_Hx	L2x_H2x	5.0	$[-0.51, 10.51]$	0.0695	0.5561	No
Lx_H	Lx_Hx	1.6	$[-6.09, 9.29]$	0.6192	1.000	No
Lx_H	L2x_H2x	0.7	$[-7.31, 8.71]$	0.8421	1.000	No
L_H2x	Lx_Hx	6.2	$[2.92, 9.48]$	0.00265	0.0318	Yes
L_H2x	L2x_H2x	5.3	$[0.16, 10.44]$	0.0452	0.4069	No
L2x_H	Lx_Hx	1.7	$[-6.11, 9.51]$	0.6031	1.000	No
L2x_H	L2x_H2x	0.8	$[-7.32, 8.92]$	0.8220	1.000	No
Lx_H2x	Lx_Hx	1.5	$[-2.07, 5.07]$	0.3615	1.000	No
Lx_H2x	L2x_H2x	0.6	$[-4.62, 5.82]$	0.7903	1.000	No
L2x_Hx	Lx_Hx	5.1	$[1.94, 8.26]$	0.00671	0.0738	No
L2x_Hx	L2x_H2x	4.2	$[-0.93, 9.33]$	0.0895	0.6262	No

Maze shows a positive average trend with configuration-dependent uncertainty. The descriptive group means favor asymmetric injection (73.1% versus 69.9%, a difference of approximately 3.2 percentage points), but the pairwise evidence is more limited. Four of the 12 comparisons have unadjusted $p < 0.05$; after Holm correction, only L_H2x versus Lx_Hx remains significant (difference 6.2 points, 95% CI $[2.92, 9.48]$, adjusted $p = 0.0318$). The largest uncertainty occurs for Lx_H and L2x_H, whose reported SDs are 6.3 and 6.4 points. Thus, the higher average accuracy does not establish a uniform Maze advantage: the evidence depends on the configuration and its seed variability.

C Level-token analysis

C.1 Level-token interpretation

The level-token control in Section 5 helps clarify what kind of state-identity signal AIR needs. The failure of element-wise addition is informative. Let $\tau_L, \tau_H \in \mathbb{R}^D$ denote the learned level vectors used to mark L- and H-updates, respectively, and let τ_m denote the vector for update type $m \in \{L, H\}$. Suppose an L- or H-update receives token representations $X \in \mathbb{R}^{S \times D}$, where S is the sequence length and D is the hidden dimension. Element-wise addition injects the level vector by replacing every token with

$$X'_i = X_i + \tau_m, \qquad m \in \{L, H\}.$$

In a self-attention layer,

$$Q_i = X_i' W_Q, \qquad K_j = X_j' W_K, \qquad V_j = X_j' W_V.$$

Because the same τ_m is added to every sequence position, it contributes the same state-dependent offsets

$$\tau_m W_Q, \qquad \tau_m W_K, \qquad \tau_m W_V$$

to all queries, keys, and values.

Thus the level signal is not localized to a separate token or relation; it is mixed uniformly into every content token before attention is computed.

Expanding the attention logit between positions i and j makes the failure mode of the *Addition* variant visible:

$$Q_i K_j^\top = X_i W_Q W_K^\top X_j^\top + X_i W_Q W_K^\top \tau_m^\top + \tau_m W_Q W_K^\top X_j^\top + \tau_m W_Q W_K^\top \tau_m^\top.$$

Of the four terms above, only $\tau_m W_Q W_K^\top \tau_m^\top$ is a position-independent constant; the two cross-terms $X_i W_Q W_K^\top \tau_m^\top$ and $\tau_m W_Q W_K^\top X_j^\top$ depend on the position-varying token representations X_i and X_j, so the contribution of τ_m to attention is content- and position-dependent rather than a uniform global bias. What the shared Transformer still cannot do is read τ_m on its own. At each position i, the query splits as $Q_i = X_i W_Q + \tau_m W_Q$. The first part depends on the token X_i and so varies by position. The second part, $\tau_m W_Q$, is the same vector at every position. The same split holds for the keys ($K_j = X_j W_K + \tau_m W_K$) and the values ($V_j = X_j W_V + \tau_m W_V$). Because the level-vector terms $\tau_m W_Q$, $\tau_m W_K$, and $\tau_m W_V$ never appear on their own at any single position, an attention head has no way to read them in a structurally clean way. Prepend-and-strip avoids this problem. There, the level token is at its own dedicated position, and attention can read $\tau_m W_*$ from that position without content mixed in. This differs from asymmetric input injection, where the injected $\tilde{\mathbf{x}}$ is input-dependent and content-structured, and from prepend-and-strip, where the level signal occupies its own sequence position and can be selectively read through attention.

By contrast, prepend (strip) gives the model a structurally separable level token: the signal stays in its own sequence position, can be read through attention, and is removed after the call. Its value is also reset at every call, so it continues to mark the update type instead of accumulating content-dependent history. Prepend (no strip) instead allows its position-0 slot to accumulate recurrent content. The consistency analysis in Appendix C.2 supports this interpretation, although its accuracy is not significantly different from prepend (strip) in the matched comparison.

Taken together, these controls support a transient, structurally separable state-identity signal as an effective design under the studied training setup; they do not establish that persistent tokens cannot provide useful identity information.

C.2 Longer training and token consistency

Training beyond the 20,000 budget. We train the symmetric Lx_Hx prepend (no strip) control for five seeds to 40,000 epochs, twice the main comparison's budget, and record training and evaluation (using train and test split of the dataset, respectively) loss and exact solve every 2,000 epochs. Figure 10 shows all 20 checkpoints as mean ± SD across seeds. Unlike Table 2, which selects each seed's best evaluation within 20,000 epochs, these curves aggregate values at the same epoch across seeds. This distinction accounts for the no-strip table value of $55.34 \pm 5.52\%$ versus $53.8 \pm 5.4\%$ at the fixed 20,000 checkpoint.

Training and evaluation diverge after the mid-training peak. Training loss falls overall from 0.785 ± 0.026 at 2,000 to 0.272 ± 0.152 at 40,000, while training exact solve rises from $2.2 \pm 3.9\%$ to $94.5 \pm 5.7\%$. Evaluation loss reaches its minimum at 24,000 (0.367 ± 0.041), and evaluation exact solve peaks at 28,000 ($58.5 \pm 3.6\%$). By 40,000, evaluation loss rises to 0.488 ± 0.166 and exact solve falls to $50.0 \pm 10.4\%$, below the fixed 20,000 value. Thus, longer training gives a temporary gain followed by overfitting and greater variation across seeds.

Persistent slots acquire puzzle-dependent content. To examine whether the retained token remains a clean level marker, we measure its cross-puzzle consistency on 128 held-out puzzles at every checkpoint. For each state $m \in \{H, L\}$, let $s_{m,i}$ denote the persistent position-0 vector for

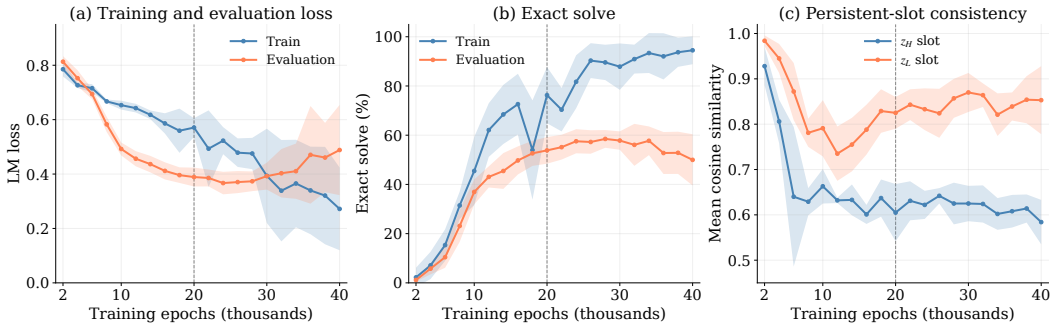


Figure 10: Longer training of prepend (no strip) on Sudoku-Extreme with symmetric Lx_Hx injection. The dashed line marks the main comparison's 20,000 budget.

puzzle i at the final recurrent step, and let $\bar{s}_m = 128^{-1} \sum_{i=1}^{128} s_{m,i}$. Consistency is the average cosine similarity $128^{-1} \sum_{i=1}^{128} \cos(s_{m,i}, \bar{s}_m)$, computed separately for each seed. A value near one indicates a common direction across puzzles; lower values indicate puzzle-dependent variation in the slot.

This variation develops early and persists throughout training. H-slot consistency drops from 0.928 ± 0.042 at 2,000 to 0.640 ± 0.154 at 6,000, then fluctuates near 0.60–0.66 before ending at 0.584 ± 0.049. L-slot consistency falls from 0.984 ± 0.012 to 0.872 ± 0.063 over the same early interval and ends at 0.853 ± 0.075. The learned level-vector norms at 40,000 are $\|\tau_H\| = 0.584 \pm 0.037$ and $\|\tau_L\| = 0.599 \pm 0.034$. Thus, the level vectors learn nonzero values, while the recurrent slots that carry them acquire puzzle-specific content. Longer training does not restore their initial cross-puzzle consistency. These observations support content accumulation in persistent slots, but do not establish that it alone causes the later evaluation decline or that update identity becomes unrecoverable.

C.3 Pairwise tests for the level-token controls

Table 5 reports all six pairwise comparisons among the four matched `Lx_Hx` level-token controls in Table 2. As in Appendix B, we use two-sided Welch t-tests over each run's best evaluation within 20,000 epochs ($n = 5$ seeds per control) and apply Holm correction across the six comparisons at $\alpha = 0.05$. Differences are the first control minus the second, in percentage points.

Table 5: Pairwise tests among the matched `Lx_Hx` level-token controls on Sudoku-Extreme (Table 2).

Comparison	Difference	95% CI	p	Holm p	Significant?
Addition − Base	−1.16	[−4.89, 2.57]	0.478	0.788	No
Strip − Base	6.16	[3.85, 8.48]	0.000281	0.00168	Yes
Strip − Addition	7.32	[3.60, 11.05]	0.00297	0.0149	Yes
No-strip − Base	3.27	[−3.47, 10.02]	0.263	0.788	No
No-strip − Addition	4.43	[−2.40, 11.26]	0.165	0.659	No
Strip − No-strip	2.89	[−3.86, 9.64]	0.314	0.788	No

D Operator-form control

Table 6 reports accuracy on Sudoku-Extreme for four operator-form variants, all keeping the asymmetry pattern of the default Lx_H configuration while replacing the default additive input $+\tilde{\mathbf{x}}$. The transformation g in the linear and nonlinear variants is a learned $D \times D$ weight matrix that maps a token's D-dimensional embedding to another D-dimensional embedding, where D is the shared Transformer's hidden dimension. All four alternatives remain in the approximately 58–60% interval, compared with 60.0% for additive injection. Two-sided Welch tests computed from the reported means and SDs ($n = 5$ seeds per variant), with Holm correction across the four comparisons to the additive baseline, find no statistically detectable differences. The differences range from -0.1 to -1.7 percentage points, and all individual 95% confidence intervals include zero. These results do not establish equivalence among operator forms.

Table 6: Operator-form control on Sudoku-Extreme. Each variant modifies only the L-update argument while keeping the H-update as $f(z_H+z_L)$. Mean $\pm$ standard deviation across 5 seeds.

Variant	L-input	Acc. (%)
Default (additive)	$f(z_L+z_H+\tilde{\mathbf{x}})$	$60.0_{\pm 2.0}$
Linear transform	$f(z_L+z_H+g(\tilde{\mathbf{x}}))$	$59.6_{\pm 1.3}$
Nonlinear transform	$f(z_L+z_H+\tanh(g(\tilde{\mathbf{x}})))$	$59.9_{\pm 1.5}$
Sign flip	$f(z_L+z_H-\tilde{\mathbf{x}})$	$59.2_{\pm 1.2}$
Hadamard product	$f(z_L+(z_H+z_L)\odot\tilde{\mathbf{x}})$	$58.3_{\pm 1.0}$

E Single-state ablation

The injection ablations in Section 5 show that the shared model needs a way to distinguish its two update types. A natural follow-up question is whether the two latent states z_H and z_L are themselves necessary, or whether periodic input injection into a *single* latent state would suffice.

To test this, we replace z_H and z_L with a single state $\mathbf{z}$ and keep everything else identical. Within the original L-cycle, $\mathbf{z}$ receives input injection ($\mathbf{z} \leftarrow f(\mathbf{z}+\tilde{\mathbf{x}})$); within the original H-cycle, it does not ($\mathbf{z} \leftarrow f(\mathbf{z})$). This is analogous to the L,L,H,L,L,H pattern used in the default two-states asymmetric variant.

Table 7: Single-state ablation on Sudoku-Extreme (5 seeds).

Variant	Acc. (%)
Single state $\mathbf{z}$ (periodic injection)	$51.1_{\pm 1.8}$
For reference:	
Symmetric two-state, $(1,1)$	$52.1_{\pm 1.6}$
Asymmetric two-state, $(1,0)$ (default)	$60.0_{\pm 2.0}$

The single-state model achieves 51.1% accuracy, on par with the symmetric two-states baselines and far below the asymmetric default (60.0%). This indicates that periodic input injection alone is not sufficient. A useful way to see the difference is to write the single-state recurrence as one stream

$$\mathbf{z}^{(t+1)} = f\Big(\mathbf{z}^{(t)} + a_t\tilde{\mathbf{x}};\theta\Big), \qquad a_t \in \{0,1\},$$

where a_t is a periodic injection indicator. Although the shared model can observe whether the current sub-step is injected or non-injected, both update types overwrite the same state $\mathbf{z}$. Thus any injected-step features and non-injected-step features must coexist in the same representational buffer:

$$\mathbf{z}^{(t)} = \Phi_t(\mathbf{z}^{(0)}, \tilde{\mathbf{x}}; a_0, \ldots, a_{t-1}).$$

By contrast, AIR maintains two persistent slots,

$$z_L^{(t+1)} = f\Big(z_L^{(t)} + z_H^{(t)} + \tilde{\mathbf{x}};\theta\Big), \qquad z_H^{(t+1)} = f\Big(z_H^{(t)} + z_L^{(t+1)};\theta\Big).$$

The injected and non-injected computations are therefore not merely two phases of one trajectory; they are written back into different latent states. This gives the model two memory buffers with different update histories: z_L accumulates the input-conditioned, partially committed refinement dynamics, while z_H accumulates the non-injected, fully committed proposal dynamics. The single-state control removes this storage separation, so the injected and non-injected states cannot stably specialize into separate latent roles.

F Attention analysis on Sudoku

This section gives the formal definitions for the Sudoku attention statistics used in Section 7 and provides additional visualizations. We focus on the same L-update versus H-update comparison as in Section 7.

F.1 Setup

Notation. Throughout this appendix, we identify each test puzzle by a tag `pNNNN` (e.g., `p0121`) and denote a board cell by $rXcY$, meaning the cell at row X and column Y of the 9×9 Sudoku board (1-indexed, with $r1c1$ at the top-left and $r9c9$ at the bottom-right).

Formal definitions. For each blank query cell q and attention head h, let $\tilde{p}_q^{(L)}(h), \tilde{p}_q^{(H)}(h) \in \mathbb{R}^{81}_{\geq 0}$ denote the raw per-head attention weights from q to the 81 puzzle-cell keys at the L-update and H-update. For each query cell we report the arithmetic mean across the 8 heads of

$$\begin{aligned}
\Delta_{\mathrm{nbr}}(q) &= \mathrm{mass}_L^{\mathrm{nbr}}(q) - \mathrm{mass}_H^{\mathrm{nbr}}(q), \\
\Delta_{\mathrm{viol}}(q) &= \mathrm{mass}_L^{\mathrm{viol}}(q) - \mathrm{mass}_H^{\mathrm{viol}}(q), \\
\Delta_{\mathrm{ent}}(q) &= \mathrm{entropy}(p_q^{(H)}) - \mathrm{entropy}(p_q^{(L)}),
\end{aligned}$$

where $\mathcal{N}(q)$ is the Sudoku neighborhood of a query cell q (the 20 cells sharing a row, column, or 3×3 box) and $\mathcal{V}$ is the set of cells currently violating a constraint under decoded z_H. Let k denote the key cell to which q attends. Then $\mathrm{mass}^{\mathrm{nbr}}(q) = \sum_{k\in\mathcal{N}(q)} \tilde{p}_{qk}$, $\mathrm{mass}^{\mathrm{viol}}(q) = \sum_{k\in\mathcal{N}(q)\cap\mathcal{V}} \tilde{p}_{qk}$, and $\mathrm{entropy}(p_q) = -\sum_k p_{qk} \log p_{qk} \,/\, \log 81$ over the normalized distribution $p_{qk} = \tilde{p}_{qk} / \sum_{k'} \tilde{p}_{qk'}$.

Within-neighborhood control. For a subset $S \subseteq \mathcal{N}(q)$ and update type $m \in \{L, H\}$, define the per-cell attention density $\rho_m^S(q) = \frac{1}{|S|} \sum_{k\in S} \tilde{p}_{qk}^{(m)}$. The within-neighborhood control statistic used in Section 7 is then

$$\Delta\rho_{\mathrm{viol}}(q) \;=\; \big[\rho_L^{\mathcal{N}(q)\cap\mathcal{V}}(q) - \rho_H^{\mathcal{N}(q)\cap\mathcal{V}}(q)\big] \;-\; \big[\rho_L^{\mathcal{N}(q)\setminus\mathcal{V}}(q) - \rho_H^{\mathcal{N}(q)\setminus\mathcal{V}}(q)\big],$$

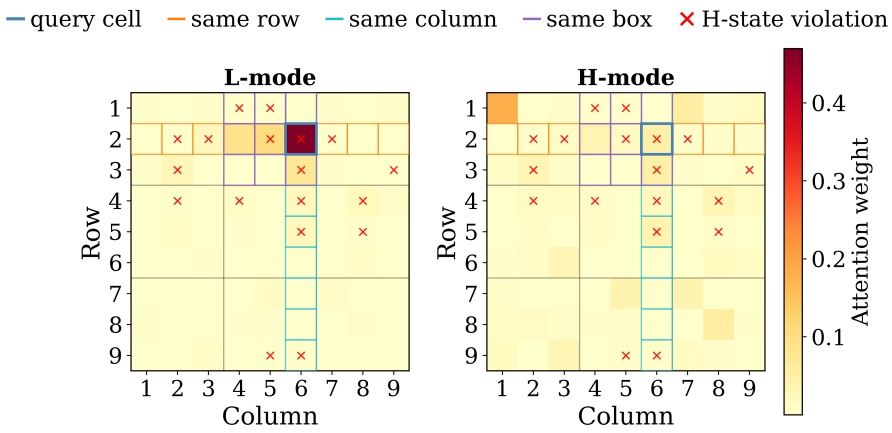
that is, the L−H per-cell density on violated cells inside $\mathcal{N}(q)$ minus the L−H per-cell density on the non-violated cells inside $\mathcal{N}(q)$. We restrict the average to violation-adjacent queries whose $\mathcal{N}(q)$ contains both bins (so both $\rho^{\mathcal{N}(q)\cap\mathcal{V}}$ and $\rho^{\mathcal{N}(q)\setminus\mathcal{V}}$ are well defined). The Maze appendix (Appendix G) uses the analogous statistic with $\mathcal{N}(q)$ replaced by $\mathcal{N}_4(q)$.

F.2 Representative examples

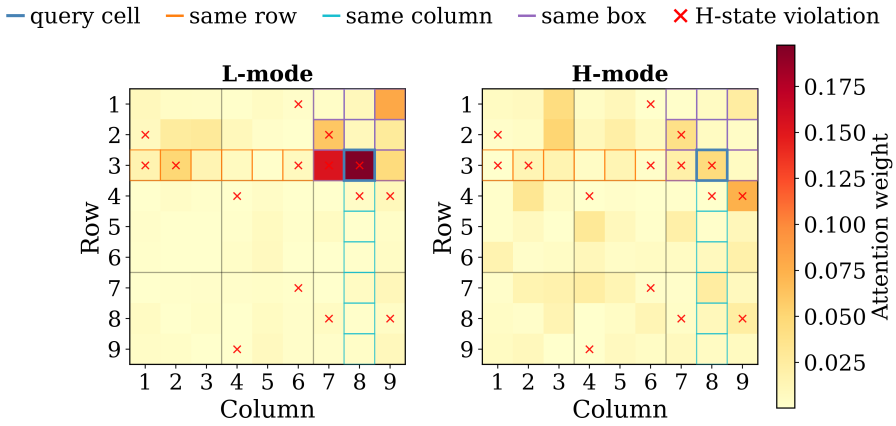
Figure 11a shows a blank query cell whose neighborhood contains a violated cell (puzzle `p0121`, cycle 2, head 1, layer 0). For this query, L-update is more concentrated inside $\mathcal{N}(q)$, places more attention mass on violating cells, and has lower entropy than H-update. Figures 11b and 11c show two additional queries from different puzzles and cycles with similar qualitative patterns.

F.3 Temporal persistence

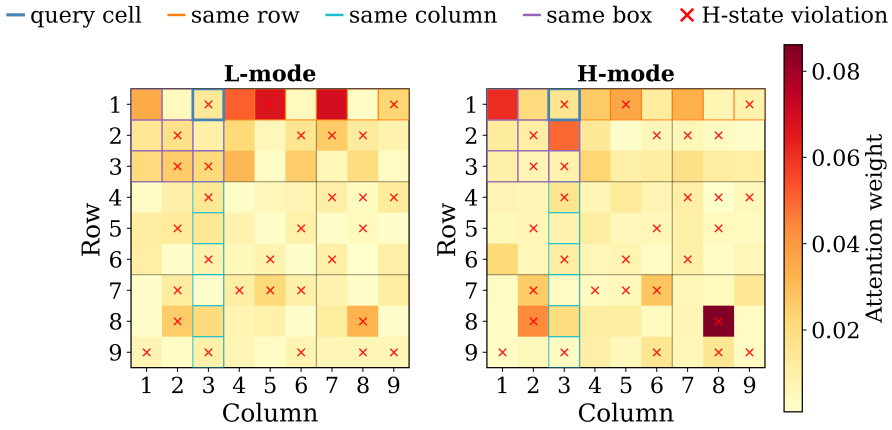
Figure 12 tracks one query cell, head, and layer across multiple cycles. The L-update/H-update separation in terms of attention mass distribution persists over time, which we denote as *temporal persistence*: L-update repeatedly emphasizes neighboring regions, whereas H-update remains more global. This rules out the interpretation that the difference is confined to a single cycle.



(a) Puzzle `p0121`, cycle 2, head 1, layer 0, query $r2c6$. L-update concentrates roughly 0.81 of its mass inside $\mathcal{N}(q)$ versus 0.24 for H-update.



(b) Puzzle `p0926`, cycle 2, head 0, layer 0, query $r3c8$. L-update concentrates most of its attention onto the same-box neighbor (e.g. cells $r3c7$, $r1c9$), while H-update places substantial mass outside the neighborhood (e.g., $r4c9$).



(c) Puzzle `p0025`, cycle 4, head 2, layer 0, query $r1c3$. L-update concentrates along the query's row inside the neighborhood; H-update is more global.

Figure 11: Three representative L-update versus H-update attention examples on Sudoku.

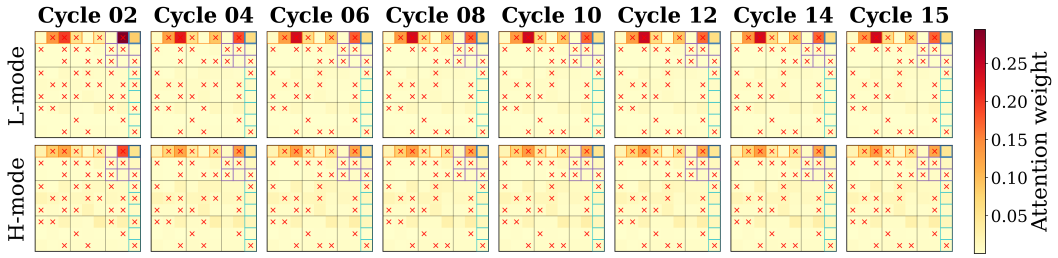


Figure 12: Temporal persistence for a single query cell (puzzle p0829, head 5, layer 0, query $r1c9$) across cycles 2 through 15.

F.4 Layer robustness

Figure 8 in Section 7 shows the L-update/H-update separation across all four layers; Table 8 gives the corresponding numerical values with 95% confidence intervals.

Table 8: L-update/H-update attention separation across all Transformer layers, averaged over the 8 heads at each layer (puzzle-level mean $\pm$ 95% CI). Aggregated over n=829 violation-adjacent and n=489 control puzzles drawn from 1,000 test puzzles. Δ_{viol} is zero on control cells by construction, because $\mathcal{N}(q) \cap \mathcal{V} = \emptyset$ for them.

Class	Layer	Δ_{nbr}	Δ_{ent}	Δ_{viol}
Viol.-adj.	0	$0.017_{\pm 0.002}$	$0.029_{\pm 0.001}$	$-0.018_{\pm 0.002}$
	1	$0.002_{\pm 0.001}$	$0.012_{\pm 0.001}$	$0.006_{\pm 0.001}$
	2	$0.040_{\pm 0.001}$	$0.056_{\pm 0.001}$	$0.028_{\pm 0.001}$
	3	$0.037_{\pm 0.002}$	$0.039_{\pm 0.002}$	$0.037_{\pm 0.002}$
Control	0	$0.050_{\pm 0.001}$	$0.053_{\pm 0.000}$	$0.000_{\pm 0.000}$
	1	$0.015_{\pm 0.001}$	$0.023_{\pm 0.000}$	$0.000_{\pm 0.000}$
	2	$0.138_{\pm 0.001}$	$0.125_{\pm 0.001}$	$0.000_{\pm 0.000}$
	3	$0.244_{\pm 0.002}$	$0.182_{\pm 0.002}$	$0.000_{\pm 0.000}$

The local-versus-global split is visible throughout the network: $\Delta_{\text{nbr}} > 0$ and $\Delta_{\text{ent}} > 0$ at every layer for control cells, and at all layers for violation-adjacent cells as well, though the effect is weakest at layer 1. The violation-specific signal behaves differently. At layer 0, Δ_{viol} is slightly negative on violation-adjacent cells, so L-updates are not yet preferentially routing attention to error cells within the neighborhood. By layers 2 and 3, however, Δ_{viol} is clearly positive. We therefore interpret locality as an early and robust property of L-updates, with explicit violation sensitivity emerging later in depth.

F.5 Causal attention interventions

Intervention protocol. The locality analysis in Section 7 describes where L and H attend; here we test whether changing that access affects solving. We intervene on attention at inference, keeping model parameters fixed, and evaluate five independently trained checkpoints on the same 1,000 Sudoku test puzzles. Because the answer is decoded only from the H-state, intervening on the final H-update would directly alter the update that produces the decoded output. We therefore leave the final H-update of each recurrent cycle unchanged and intervene only on its earlier H-updates; L interventions affect every L-update.

We use two interventions: attention knockout and diffuse attention. Knockout removes selected keys from attention by adding a large negative mask (-10^9) to their logits before softmax. For a query cell q, let $\mathcal{N}(q)$ be its 20 peers sharing a row, column, or 3×3 box. We either forbid access to $\mathcal{N}(q)$ or confine access to it, calling the latter a local update. To control for the number of accessible cells, we also confine attention to a random set of 20 other cells per query, sampled once with a fixed seed and reused across checkpoints. All knockout masks retain self-attention and unrestricted access to and from puzzle-prefix tokens. A diffuse update instead spreads attention uniformly over all sequence

positions, including prefix tokens. Both interventions apply to every head and layer of the selected update type.

Evaluation. We measure per-cell accuracy and exact solve (the percentage of puzzles with every evaluated cell correct) from the final decoded H-state after the normal halting procedure. Table 9 reports mean $\pm$ sample SD across five seeds; changes below are relative to the no-intervention baseline in this table and are expressed in percentage points. We test effects with Welch t-tests on seed-level measurements. A one-sided test is used only for a disruption predicted a priori to degrade performance relative to baseline; comparisons between interventions and other comparisons use two-sided tests.

Table 9: Sudoku causal attention interventions at inference (five-seed mean $\pm$ SD).

Intervention	Per-cell accuracy	Exact solve
None (baseline)	$83.6_{\pm 0.6}$	$55.2_{\pm 1.6}$
L diffuse	$51.2_{\pm 3.8}$	$0.0_{\pm 0.0}$
L excludes neighborhood	$53.7_{\pm 6.1}$	$0.0_{\pm 0.1}$
L local only	$64.2_{\pm 9.9}$	$12.8_{\pm 13.8}$
L random only	$51.1_{\pm 8.2}$	$0.0_{\pm 0.0}$
H diffuse	$78.5_{\pm 1.4}$	$36.9_{\pm 5.0}$
H excludes neighborhood	$59.8_{\pm 1.6}$	$0.8_{\pm 0.7}$
H local only	$81.7_{\pm 1.5}$	$51.2_{\pm 4.0}$
L local, H diffuse	$60.4_{\pm 5.6}$	$4.4_{\pm 6.8}$
L diffuse, H local	$57.2_{\pm 1.9}$	$0.1_{\pm 0.2}$
Both diffuse	$47.3_{\pm 2.3}$	$0.0_{\pm 0.0}$

L's neighborhood access contributes causally to solving. Making L diffuse reduces per-cell accuracy by 32.4 percentage points ($p = 2 \times 10^{-5}$), and excluding its neighborhood reduces it by 29.9 points ($p = 2 \times 10^{-4}$); both nearly eliminate exact solving. These interventions change attention while leaving the learned parameters fixed, showing that L's access to neighboring cells is part of the computation supporting correct answers. Confining L to its neighborhood retains some exact solving (12.8%), but remains below baseline (55.2%), indicating that local access alone is insufficient to preserve performance.

Controlling which cells L can attend to. To distinguish neighborhood structure from the number of accessible cells, we compare local attention with the matched-size random control. Both allow L to attend to 20 other cells and differ only in their locations. The true neighborhood retains more per-cell accuracy (64.2% versus 51.1%), corresponding to reductions of 19.4 and 32.5 points from baseline. This ordering suggests that the choice of cells matters, although the Sudoku comparison does not reach the 0.05 significance threshold ($p = 0.053$).

Disrupting L attention increases rule violations per error. Per-cell accuracy alone does not reveal whether a prediction respects Sudoku constraints: a cell can differ from the target solution while remaining consistent with its row, column, and box. We therefore also measure violations per wrong cell in the final decoded H-state. A filled cell participates in a violation if its value repeats another in its row, column, or box; each participating cell is counted once. For each seed, we divide the total number of these cells by the total number of cells differing from the target solution. This ratio can exceed one because correctly predicted cells can also conflict with incorrect neighbors.

Making L diffuse raises violations per wrong cell from 0.84 to 1.41 ($p < 10^{-8}$; Table 10), showing that the disruption increases rule conflicts relative to the number of prediction errors. Making H diffuse gives a ratio near baseline (0.91, $p = 0.72$), with no statistically detectable change. Together, these results support a role for L's attention in maintaining local constraint consistency.

Table 10: Sudoku rule violations under attention interventions (five-seed mean ± SD).

Intervention	Per-cell accuracy (%)	Violations per wrong cell
None (baseline)	$83.6_{\pm 0.6}$	$0.84_{\pm 0.04}$
L diffuse	$51.2_{\pm 3.8}$	$1.41_{\pm 0.03}$
H diffuse	$78.5_{\pm 1.4}$	$0.91_{\pm 0.38}$

G Attention analysis on Maze

This section gives the formal definitions for the Maze attention statistics used in Section 7 and provides the supporting visualizations. As in Section 7, *local* means attention concentrated near the query, *global* means more global attention, and all reported quantities are averaged over the 8 attention heads.

G.1 Setup

Notation. We identify each Maze-30×30 test puzzle by a tag `pNNNN` and denote a board cell by $rXcY$, the cell at row X and column Y of the 30×30 grid (1-indexed, with $r1c1$ at the top-left and $r30c30$ at the bottom-right). All maze panels show an 11×11 window centered on the query.

We extract attention from all four layers and compare the first L-update with the first H-update across cycles $\{2, 4, 6, 8, 10, 12, 14, 15\}$. We take the cells that lie in the solution path in the 30×30 maze as query cells. For each query cell q, we define three neighborhoods analogous to the Sudoku neighborhood: the 4-connected neighborhood $\mathcal{N}_4(q)$, the 8-connected neighborhood $\mathcal{N}_8(q)$, and the centered 5×5 window $\mathcal{N}_{5\times5}(q)$ (excluding q itself), with $\mathcal{N}_4(q) \subseteq \mathcal{N}_8(q) \subseteq \mathcal{N}_{5\times5}(q)$.

The violation set $\mathcal{V}$ contains cells where decoded z_H disagrees with the ground truth. A query is *error-adjacent* if $\mathcal{N}_4(q) \cap \mathcal{V} \neq \emptyset$ and *control* otherwise. For each query we report the per-cell density contrast

$$\Delta\rho^{(k)}(q) = \frac{\mathrm{mass}_L(\mathcal{N}_k(q)) - \mathrm{mass}_H(\mathcal{N}_k(q))}{|\mathcal{N}_k(q)|},$$

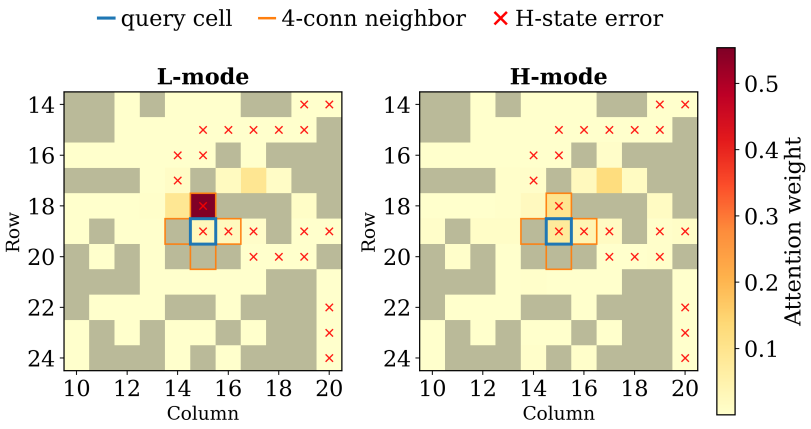
which keeps comparisons across neighborhood sizes on the same scale; the entropy contrast $\Delta_{\mathrm{ent}}(q) = \mathrm{entropy}(p_q^{(H)}) - \mathrm{entropy}(p_q^{(L)})$ where $\mathrm{entropy}(p_q) = -\sum_k p_{qk} \log p_{qk} \,/\, \log 900$ over the normalized 900-cell distribution; and the violation contrast $\Delta_{\mathrm{viol}}(q)$, the L–H difference in mass on $\mathcal{N}_4(q) \cap \mathcal{V}$. We aggregate over 1,000 test puzzles, yielding n=297 error-adjacent and n=746 control puzzles after class assignment of error-adjacent and control, and report 95% confidence intervals.

G.2 Representative examples

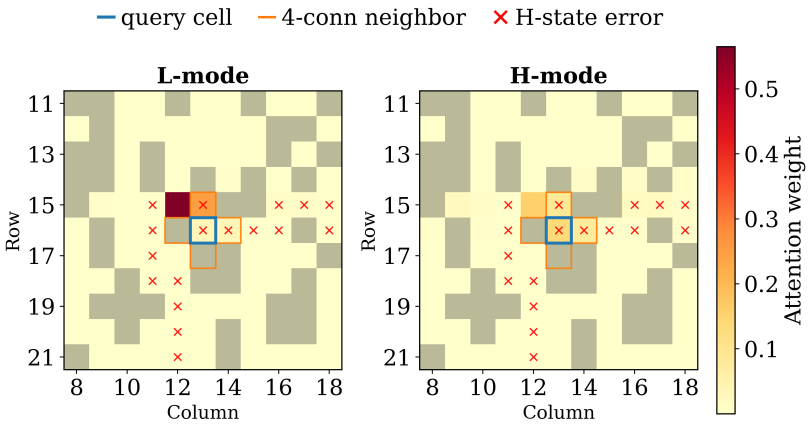
Figure 13a shows an error-adjacent query cell (puzzle `p0040`, cycle 4, head 7, layer 0). For this query, L-update is more concentrated inside $\mathcal{N}_4(q)$, whereas H-update spreads mass over a wider region. Figures 13b and 13c show two additional queries from different puzzles, cycles, and heads with the same qualitative pattern.

G.3 Temporal persistence

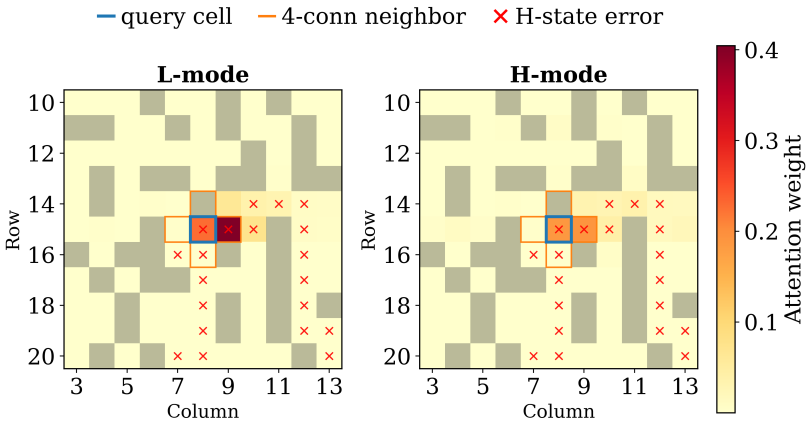
Figure 14 tracks one query cell, head, and layer across multiple cycles. The L-update/H-update separation persists over time: L-update repeatedly concentrates on $\mathcal{N}_4(q)$, while H-update remains broader. This rules out a one-step-artifact interpretation.



(a) Puzzle `p0040`, cycle 4, head 7, layer 0, query $r19c15$. L-update places almost all of its attention mass inside $\mathcal{N}_4(q)$; H-update spreads that mass over a broader region.



(b) Puzzle `p0029`, cycle 12, head 7, layer 0, query $r16c13$. L-update remains sharply local; H-update is broader and less peaked.



(c) Puzzle `p0016`, cycle 12, head 7, layer 0, query $r15c8$. The same local-versus-global split appears again.

Figure 13: Three representative L-update versus H-update attention examples on Maze-30×30.

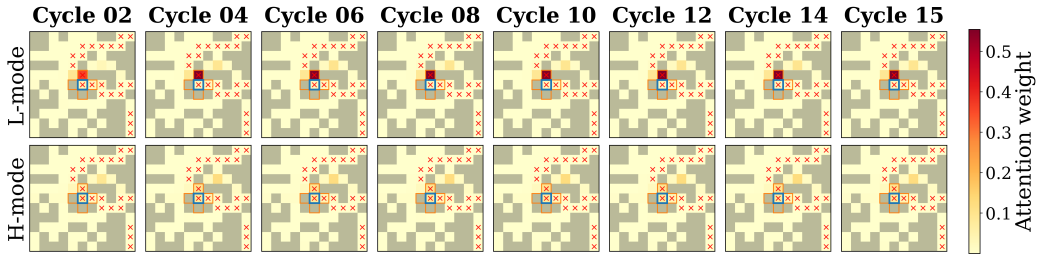


Figure 14: Temporal persistence for a single anchor query (puzzle p0040, head 7, layer 0, query $r19c15$) across cycles 2 through 15.

G.4 Layer robustness

Figure 9 in Section 7 shows the L-update/H-update separation across all four layers; Table 11 gives the corresponding numerical values with 95% confidence intervals.

Table 11: Maze: L-update/H-update attention separation across all Transformer layers, head-averaged over the 8 heads per candidate query (puzzle-level mean $\pm$ 95% CI; n=297 error-adjacent, n=746 control).

Class	Layer	$\Delta\rho^{(4)}$	$\Delta\rho^{(8)}$	$\Delta\rho^{(5\times5)}$	Δ_{ent}	Δ_{viol}
Err.-adj.	0	$0.0037_{\pm 0.0004}$	$0.0028_{\pm 0.0003}$	$0.0019_{\pm 0.0001}$	$0.149_{\pm 0.005}$	$0.006_{\pm 0.001}$
	1	$0.0085_{\pm 0.0007}$	$0.0039_{\pm 0.0004}$	$0.0044_{\pm 0.0002}$	$0.186_{\pm 0.006}$	$0.014_{\pm 0.002}$
	2	$0.0046_{\pm 0.0005}$	$0.0047_{\pm 0.0004}$	$0.0019_{\pm 0.0002}$	$0.145_{\pm 0.006}$	$0.008_{\pm 0.002}$
	3	$0.0412_{\pm 0.0014}$	$0.0286_{\pm 0.0009}$	$0.0126_{\pm 0.0004}$	$0.314_{\pm 0.010}$	$0.081_{\pm 0.005}$
Control	0	$0.0044_{\pm 0.0002}$	$0.0031_{\pm 0.0002}$	$0.0017_{\pm 0.0001}$	$0.152_{\pm 0.002}$	$0.000_{\pm 0.000}$
	1	$0.0115_{\pm 0.0002}$	$0.0057_{\pm 0.0001}$	$0.0045_{\pm 0.0001}$	$0.208_{\pm 0.002}$	$0.000_{\pm 0.000}$
	2	$0.0026_{\pm 0.0002}$	$0.0024_{\pm 0.0001}$	$0.0013_{\pm 0.0001}$	$0.157_{\pm 0.002}$	$0.000_{\pm 0.000}$
	3	$0.0367_{\pm 0.0007}$	$0.0270_{\pm 0.0005}$	$0.0131_{\pm 0.0002}$	$0.401_{\pm 0.003}$	$0.000_{\pm 0.000}$

The local-versus-global split is visible throughout the network: $\Delta\rho^{(k)} > 0$ and $\Delta_{\text{ent}} > 0$ at every layer for both classes, with the gap widening sharply at layer 3. The class-level Δ_{viol} is positive on error-adjacent cells and grows with depth, but the within-neighborhood control reported in Section 7 is null at every layer. We therefore interpret locality on Maze as an early and robust property of L-updates, while the deeper-layer violation-specific role observed on Sudoku does not replicate.

G.5 Causal attention interventions

Intervention protocol. We next test whether the local-versus-global attention split contributes to Maze solving, using five independently trained checkpoints and the same 1,000 test mazes for every condition. We keep model parameters fixed and apply the knockout and diffuse interventions defined in Appendix Section F.5. Here, the local neighborhood is $\mathcal{N}_{5\times5}(q)$: the surrounding 5×5 window, excluding the query itself and clipped at grid boundaries (up to 24 neighbors). Knockout either forbids access to this neighborhood, confines access to it, or permits a random set of the same size. Self-attention and prefix-token access remain intact in each mask; diffuse attention spreads mass uniformly over all sequence positions. Interventions affect all heads and layers of the selected updates, with the final H-update of every recurrent cycle left unchanged to avoid directly masking the update that produces the decoded output.

Table 12 reports final per-cell accuracy and exact solve as mean $\pm$ sample SD across five seeds. Changes below are in percentage points relative to this table's baseline. We use the same halting procedure and seed-level Welch testing protocol as in the Sudoku study: one-sided tests for predicted degradation from baseline and two-sided tests otherwise.

Table 12: Maze causal attention interventions at inference (five-seed mean ± SD).

Intervention	Per-cell accuracy	Exact solve
None (baseline)	$99.0_{\pm 0.3}$	$71.0_{\pm 6.3}$
L diffuse	$87.9_{\pm 3.8}$	$0.1_{\pm 0.2}$
L excludes neighborhood	$84.5_{\pm 2.2}$	$0.0_{\pm 0.0}$
L local only	$93.8_{\pm 1.8}$	$1.6_{\pm 2.3}$
L random only	$84.6_{\pm 1.9}$	$0.0_{\pm 0.0}$
H diffuse	$98.9_{\pm 0.3}$	$67.3_{\pm 7.8}$
H excludes neighborhood	$98.9_{\pm 0.3}$	$68.5_{\pm 7.2}$
H local only	$98.6_{\pm 0.5}$	$55.4_{\pm 14.0}$
L local, H diffuse	$92.0_{\pm 1.5}$	$0.1_{\pm 0.2}$
L diffuse, H local	$87.8_{\pm 2.8}$	$0.0_{\pm 0.0}$
Both diffuse	$84.3_{\pm 1.9}$	$0.0_{\pm 0.0}$

L's neighborhood access contributes causally to solving. Making L diffuse reduces per-cell accuracy by 11.1 percentage points ($p = 0.001$); excluding its neighborhood reduces it by 14.5 points ($p = 6 \times 10^{-5}$). Both nearly eliminate exact solving (Table 12). As on Sudoku, these interventions provide causal evidence that L's neighborhood access contributes to solving.

Which cells L attends to matters beyond how many. Confining L to the true neighborhood retains higher per-cell accuracy than confining it to a random set of the same size (93.8% versus 84.6%, $p < 10^{-4}$). The corresponding reductions from baseline are 5.2 and 14.4 percentage points. Both conditions expose the same number of cells per query, so the difference supports the importance of their spatial arrangement. Nevertheless, local-only L attention achieves only 1.6% exact solve: preserving neighborhood access helps cell-level predictions but is insufficient to preserve whole-maze solving.

H's broader attention access supports whole-maze solving. Confining H to its neighborhood lowers exact solve by 15.6 percentage points, from 71.0% to 55.4% ($p = 0.033$), while changing per-cell accuracy by only -0.4 points ($p = 0.19$). Making H diffuse produces a smaller exact-solve decrease of 3.7 points (67.3%, $p = 0.43$), and excluding its neighborhood yields 68.5% ($p = 0.57$). The loss in whole-maze solving under local confinement supports a contribution from H's attention beyond the neighborhood. This is consistent with H integrating broader context to coordinate a complete solution, even when most cell predictions remain correct.

The assignment of attention patterns matters. Keeping L local and H diffuse gives higher per-cell accuracy than the swapped assignment (92.0% versus 87.8%, $p = 0.026$). The same two patterns therefore have different effects depending on their assignment to L and H, supporting a directional division of labor. Exact solve is near zero for both assignments, so this comparison establishes a difference in retained per-cell accuracy rather than whole-maze solving.

H Per-seed freeze results

Section 6 reports content changes as mean ± sample SD across three seeds. Table 13 lists each seed's mean total content change over its 1,000 evaluated puzzles, and Figures 15 to 18 show the corresponding per-seed totals and per-sub-step curves. AIR and the symmetric Sudoku control use seeds 0–2; the symmetric Maze control uses seeds 2–4.

The sign of the response is consistent across seeds for five of the eight task–model–direction combinations, including freezing z_H in AIR, which reduces z_L's content changes on Sudoku and increases them on Maze in every seed. It is not consistent for freezing z_L in AIR on Sudoku or freezing z_H in the symmetric control on either task, and magnitudes vary widely throughout (Table 13). In the symmetric control, freezing z_L increases z_H's content changes in every seed on both tasks. Freezing z_H increases z_L's content changes in two Sudoku seeds but decreases them in the third, and changes them by fewer than 30 per puzzle in every Maze seed. These per-seed results reinforce our conclusion in Section 6: the freeze study establishes dynamical interdependence, but the direction and size of the response vary across tasks, intervention directions, and seeds.

Table 13: Per-seed freeze results: total content changes per puzzle in the unfrozen state, averaged over 1,000 evaluated puzzles per seed. Δ = frozen $-$ normal.

			Freeze z_H (change in z_L)			**Freeze z_L (change in z_H)**		
Task	**Model**	**Seed**	Normal	Frozen	Δ	Normal	Frozen	Δ
Sudoku	AIR	0	1396.4	177.2	-1219.2	250.3	243.0	-7.3
		1	1317.5	291.8	-1025.7	242.1	162.0	-80.1
		2	1060.6	249.5	-811.1	257.8	2349.0	$+2091.2$
	Symmetric	0	474.7	1847.5	$+1372.8$	274.4	734.1	$+459.7$
		1	456.6	384.1	-72.5	306.3	613.9	$+307.6$
		2	473.1	604.9	$+131.8$	320.5	465.0	$+144.5$
Maze	AIR	0	1265.3	2453.8	$+1188.5$	899.4	3600.0	$+2700.6$
		1	1840.2	11318.6	$+9478.4$	819.5	900.0	$+80.5$
		2	931.9	2058.8	$+1126.9$	896.7	2700.0	$+1803.3$
	Symmetric	2	4.8	32.7	$+28.0$	786.9	1557.0	$+770.1$
		3	1.4	10.3	$+8.9$	779.3	902.3	$+123.0$
		4	27.2	8.0	-19.2	724.3	738.8	$+14.5$

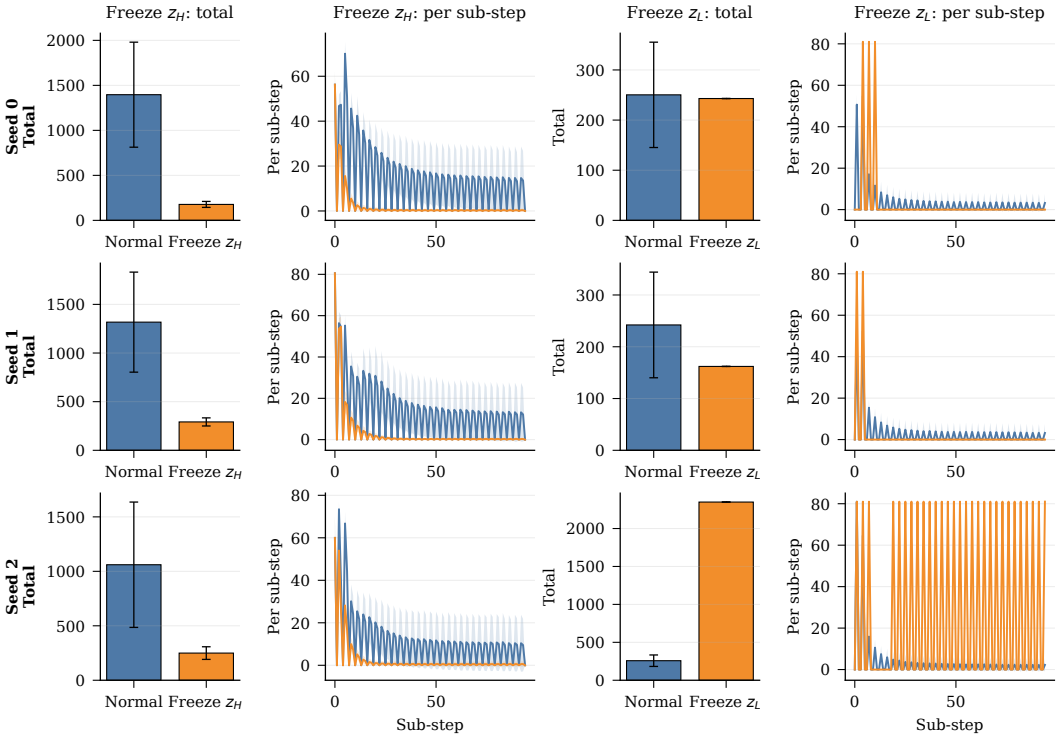


Figure 15: Per-seed freeze experiments for AIR (`Lx_H`) on Sudoku. Each row is one seed. Bars show total content changes per puzzle and curves show per-sub-step content changes for normal (blue) and frozen (orange) rollouts; error bars and bands denote $\pm$ SD across the 1,000 evaluated puzzles of that seed.

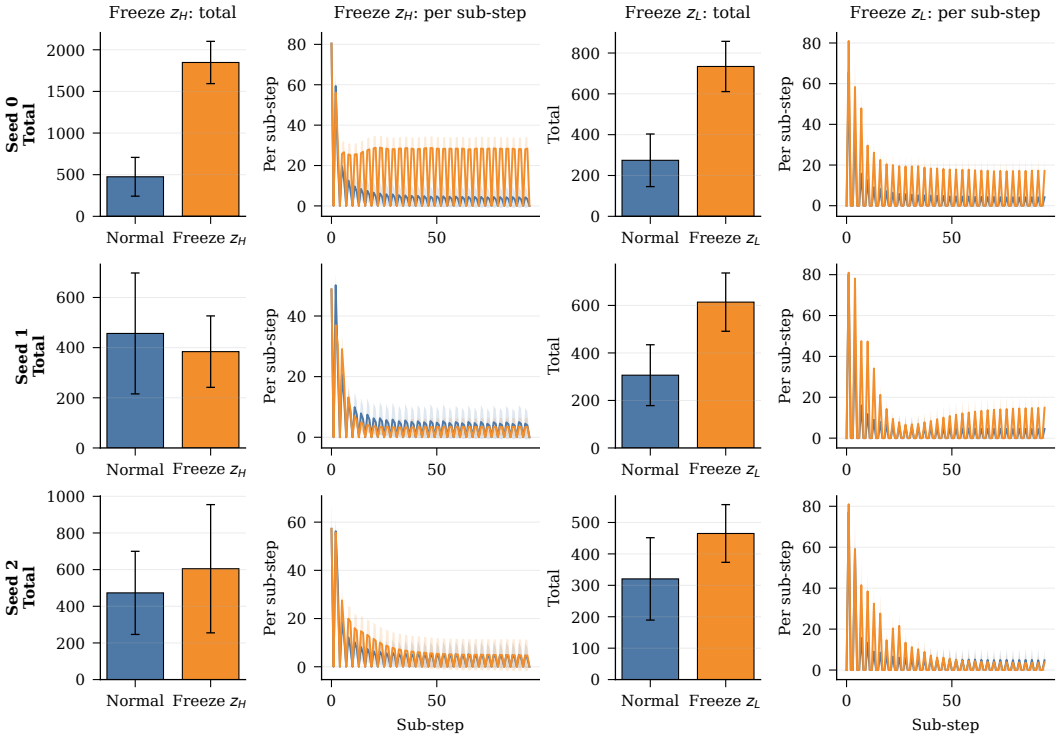


Figure 16: Per-seed freeze experiments for the symmetric `Lx_Hx` control on Sudoku, shown as in Figure 15.

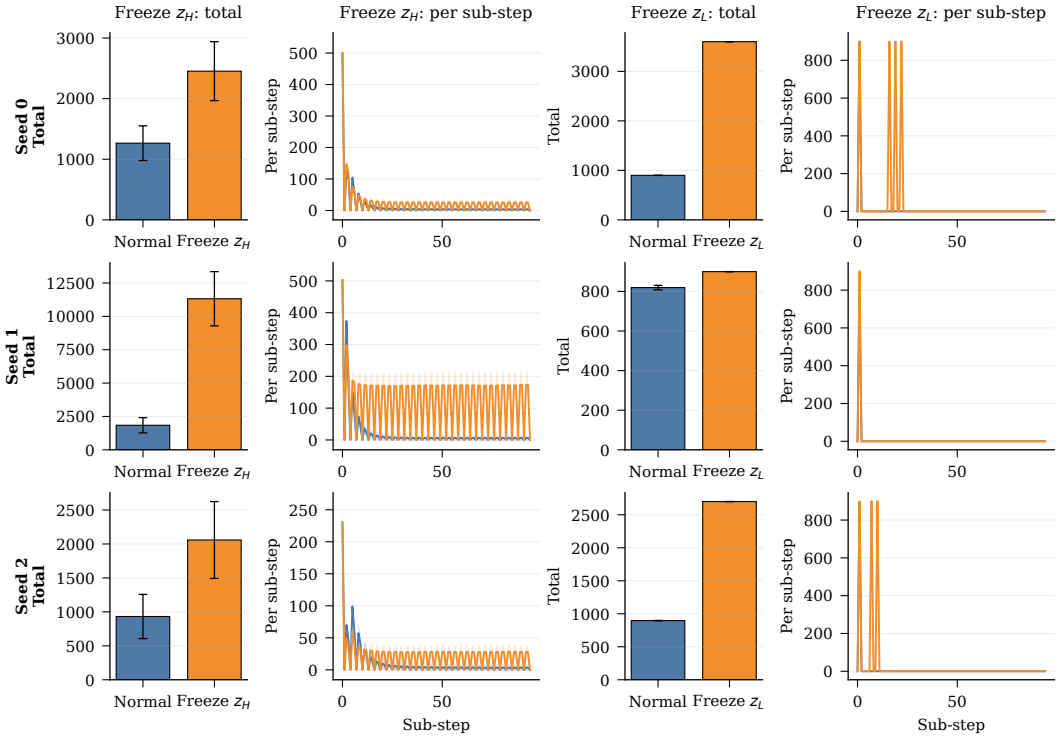


Figure 17: Per-seed freeze experiments for AIR (`Lx_H`) on Maze, shown as in Figure 15.

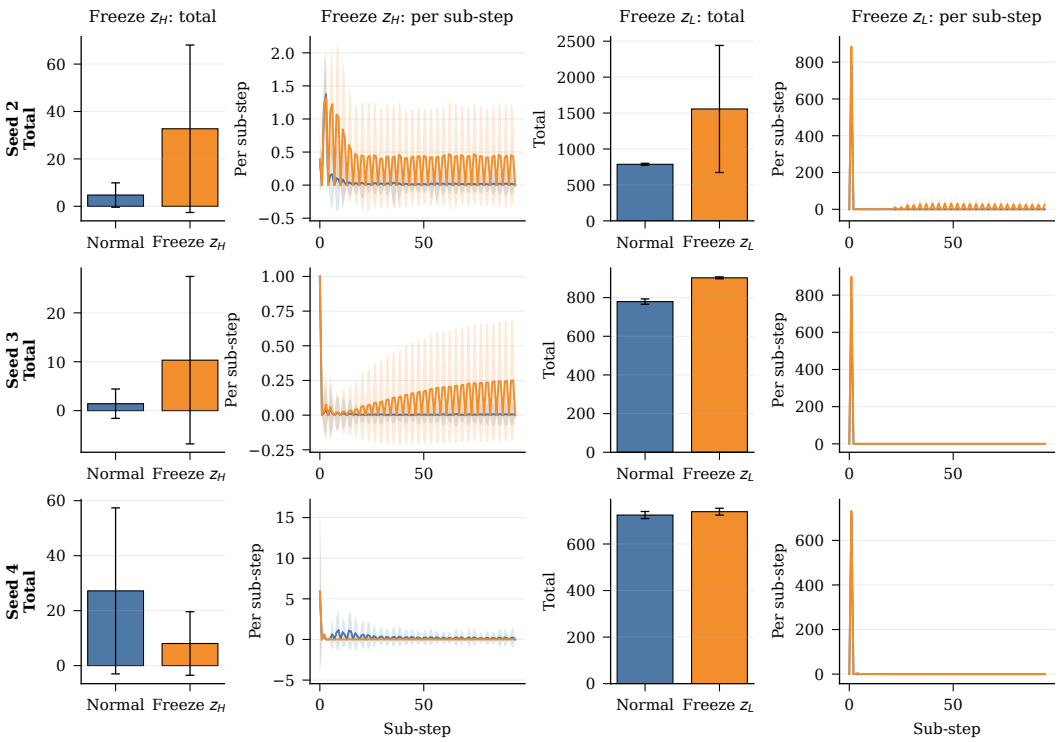


Figure 18: Per-seed freeze experiments for the symmetric `Lx_Hx` control on Maze (seeds 2–4), shown as in Figure 15.

I Interpretation of task and layer differences

Sections 6 and 7 show differences between the two update types, across layers, and between tasks. We offer the following interpretation; it is consistent with our measurements but not directly tested.

L- versus H-updates. Only L-updates receive the encoded input $\tilde{\mathbf{x}}$, so L stays anchored to the given clues and the constraints around each cell. H receives no direct input and integrates information across the whole grid through the recurrent states. This matches the more local attention of L-updates and the broader attention of H-updates (Section 7).

Across layers. On both tasks, L-updates are already more local and more concentrated than H-updates in the first layer, and the gap widens in deeper layers (Table 8 in Appendix F.4 for Sudoku; Table 11 in Appendix G.4 for Maze). The tasks differ in what emerges with depth. On Sudoku, extra attention to violated cells appears only in the deeper layers, both in Δ_{viol} and in the within-neighborhood control of Section 7. On Maze, Δ_{viol} also grows with depth, but the within-neighborhood control stays near zero at every layer, so this growth reflects stronger locality rather than attention to the error cells themselves. General locality therefore emerges early on both tasks, whereas attention to the specific cells in a conflict emerges in deeper layers only on Sudoku.

Sudoku versus Maze. The tasks differ in how their rules connect cells. A Sudoku cell must agree with every cell in its row, column, and box, so finding which of those cells conflict is useful, consistent with the violation-specific attention in deeper layers. A Maze path cell only needs to connect to its neighbors, which local attention already covers. Accordingly, L-updates attend more than H-updates to erroneous cells near the query only because they attend more locally overall; within the neighborhood, they favor erroneous cells no more than H-updates do. Many small detours also remain valid paths in Maze, so when one state is frozen the other can keep rerouting around it; this is consistent with freezing either state increasing content changes on Maze, whereas on Sudoku freezing z_H suppresses z_L's changes (Section 6).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the empirical claims on specialization, state relatedness, input/state-identity signals, and attention patterns; the scope is explicitly limited to Sudoku-Extreme and Maze.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A dedicated Limitations and Broader Impact section discusses the restriction to synthetic grid tasks, the 4-layer shared-Transformer setting, fixed recurrence schedule, and the partial nature of attention-based mechanistic evidence.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results or proofs; the equations define the model recurrence and analysis metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The recurrence and variants are defined in the Preliminaries, and architecture details, hyperparameters, training procedures, data splits, error-bar conventions, and compute are provided in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The Sudoku-Extreme and Maze datasets are public and cited, and the code is available via supplementary material. We made our best efforts to ensure code is runnable and results are reproducible, but if there're any errors or bugs in the code that make it unable to run or produce very interesting results, please flag those in the comments/reviews and we're happy to help.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix A specifies the architecture, recurrence schedule, optimizer, learning rate, batch size, training duration, data augmentation, test splits, and evaluation protocol.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Main accuracy and ablation results report mean ± standard deviation across 5 independent seeds; freeze experiments report mean ± sample standard deviation across 3 seeds, and correlational attention analyses report puzzle-level mean ± 95% confidence intervals. Causal attention interventions report mean ± sample standard deviation across 5 seeds, and the paper reports Welch tests, with Holm correction for the injection, level-token, and operator-form ablation comparisons.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: GPU type, approximate training time per run, and total reported GPU-hours are provided in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses synthetic/public algorithmic tasks and small research models, and does not involve human subjects, private data, or deployed systems.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The Limitations and Broader Impact section states that the work is foundational research on synthetic tasks, identifies no foreseeable direct harmful societal impact, and notes possible positive implications for efficient and interpretable reasoning systems.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper studies small models on synthetic/public algorithmic tasks and does not release high-risk models, scraped datasets, or deployed systems.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: Prior papers and public datasets are cited, but the current paper does not explicitly list licenses or terms of use for all existing assets.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new datasets or models.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not used as a component of the core methodology in this research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.