RETHINKING REASONING THROUGH MASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning is built from intermediate results: each step of a proof, program or iterative computation rests on what earlier steps established. Masked diffusion models (MDMs) generate by repeatedly filling in blanks, in any order and with the option to revise, and this flexibility has made them a promising alternative to left-to-right language models for reasoning. Yet their gains have fallen short, and it is unclear whether better decoding orders, larger models or stronger revision can close the gap. Here we provably show that none of them removes the fundamental bottleneck: between rounds, an MDM remembers only its tokens, and a token must be either an answer or a blank. Without revision, an MDM giving a short answer is effectively one forward pass deep, however many rounds it runs and however large it grows. Revision lets rounds build on one another, but only by paying for extra rounds or scratch tokens that grow with input length. And when the intermediate results a task needs are not yet answers, no amount of output space can hold them. Passing hidden states from one round to the next removes these limits. This Latent-State MDM, trained on at most 40 steps of the Rule 110 cellular automaton, executes 96 steps without error, where GPT-6 Astra solves only 6 of 10 cases, and lifts Sudoku-Extreme accuracy from 8.95% to 73.95% on the same 6.6M-parameter backbone, surpassing MDMs 46 times larger. Reasoning requires unfinished computation to survive before it becomes an answer.

1 INTRODUCTION

"*The limits of my language mean the limits of my world.*"
Ludwig Wittgenstein

Recently, MDM has drawn considerable attention due to its ability to generate tokens in any order (Lou et al., 2023; Hoogeboom et al., 2021), sparking the perception that it may possess superior reasoning capabilities (Li et al., 2025; Nie et al., 2026; Ye et al., 2025a). This is attributed to iterative refinement and flexible conditioning on multiple positions (Ye et al., 2025b). However, growing evidence suggests their reasoning capabilities fall short of initial expectations (Kang et al., 2026; Feng et al., 2026; Gong et al., 2026). Existing efforts have largely pursued three remedies: better unmasking orders (Kim et al., 2025; Liu et al., 2025), larger models (Nie et al., 2025; 2026; Ye et al., 2025b), and mechanisms that remask or revise earlier decisions (Wang et al., 2026).

But when does an architectural change yield reasoning gains that scale beyond a particular benchmark or problem size? Reasoning in mathematics, code, and planning is ultimately realized through computation. Although these domains differ in semantics, they all require algorithms whose dependencies and resource demands grow with the number of variables, constraints, and decisions. Computational complexity provides a domain-independent framework for studying this growth: it asks which families of computations an architecture can realize, what resources they require, and how those requirements scale with problem size (Strobl et al., 2024; Merrill & Sabharwal, 2023).

Viewed through this framework, the practical gains of these remedies leave a more fundamental question unanswered:

What remains beyond the reach of MDMs,
even when every remedy is pushed to its limit?

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The answer is unexpectedly stark. We allow the model size to grow as any fixed power of the input length, choose the best possible unmasking order for each input, and revision powerful enough to rewrite any earlier prediction. For tasks with short answers, MDMs that cannot rewrite earlier tokens remain unable to solve some elementary problems even with any polynomial number of rounds. Allowing arbitrary rewriting makes these problems solvable, yet the required number of rounds still grows polynomially with the input length. The barrier remains even when the answer contains as many tokens as the input. We construct a simple task that maps one sequence to another of the same length, yet MDMs faithful to masked generation cannot solve it exactly in any polynomial number of rounds, even when every committed token is assumed correct. These are failures of expressivity in the strongest sense: unlimited training data, ideal optimization, and every other favorable training condition cannot overcome them, because the required computation lies beyond what the architecture can realize. Yet expressivity is only the first barrier. Even when an MDM can represent the required computation, standard training may still fail to teach it. Standard mask loss sees only correct partial answers, so cloze-style completion can be rewarded without learning to solve from a fully masked state or recover from the model's own errors. A simple diagnostic example reveals a structural blind spot in the objective: training loss approaches zero as answers grow longer, yet test-time exact-match accuracy from a fully masked state remains at random-guess accuracy even after repeated revision.

These limitations point to a repair at the level of the MDM paradigm itself. Standard masked generation uses the token canvas for two fundamentally different roles: exposing the model's current answer and carrying everything the computation is allowed to remember. Here we propose the **Latent-State MDM** (LS-MDM) framework, which separates these roles by preserving a latent computational state across denoising rounds while allowing tokens to be read out independently at any stage. The framework specifies what must survive across rounds, rather than prescribing a particular memory architecture, feedback rule, readout time, or training procedure. From this perspective, recent approaches based on soft token feedback, deterministic continuous pathways, or persistent auxiliary memory can be viewed as distinct early instantiations of the same broader principle (Hersche et al., 2026; Jo et al., 2026; Xia et al., 2026); many other realizations remain possible. To isolate the principle itself, we study its simplest form: passing the hidden states of one Transformer round directly into the next and reading out tokens only after the final round, without an elaborate memory module or decoding heuristic. A broader discussion of related work appears in Section A.

This minimal instantiation already produces a qualitative change in both theory and practice. Theoretically, it subsumes revision-capable MDMs, turns polynomial-round requirements into logarithmic ones on fundamental computations, and solves sequence tasks that no faithful token-state MDM can compute in any polynomial number of rounds. Our main separations are unconditional under the stated architectural and finite-precision model: they do not rely on conjectured complexity-class separations such as $P \neq NP$. Empirically, the results follow the same progression as the theory. LS-MDM first learns S_5 composition from sequences no longer than 10 and computes every final product correctly through length 64. Standard MDMs remain near random accuracy even within the training data. LS-MDM executes Rule 110 with 66.67% exact match at depth 128 whereas GPT-6 Astra achieves only 50%. Finally, on Sudoku-Extreme, the 6.6M-parameter LS-MDM reaches 73.95% whole-puzzle exact match, compared with 8.95% for the MDM with the same backbone and 55.40% for a 302.9M-parameter MDM. The agreement between unconditional theory and experiments indicates that the gain comes not from a specialized architectural trick, but from correcting the state interface at the heart of masked iteration. It opens a broader path for MDMs: not merely improving which tokens to generate or revise, but redesigning what computation is allowed to survive before it becomes a token.

2 BACKGROUND

Masked diffusion models learn to reconstruct tokens removed by a masking process. At inference, the positions to be generated are initialized as `[MASK]`, and a bidirectional Transformer repeatedly predicts tokens from the current sequence. We view each round as consisting of a **predictor**, which proposes tokens, and a **planner**, which chooses which proposals to accept. The accepted tokens form the canvas for the next round. Unlike autoregressive decoding, this process need not follow a fixed left-to-right order and may update multiple positions in parallel (Lou et al., 2023; Hoogeboom et al., 2021).

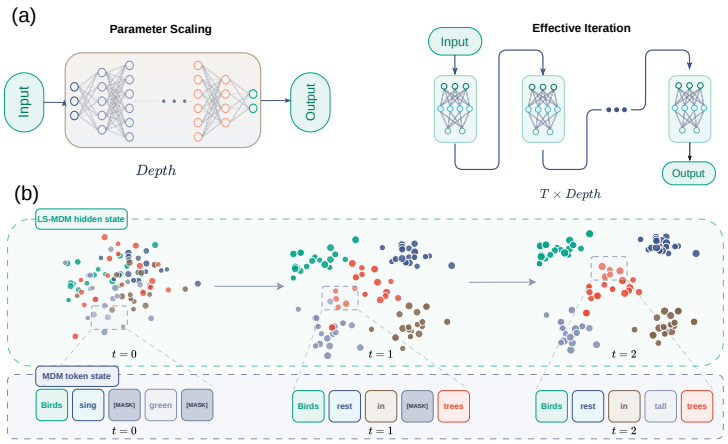


Figure 1: **a**: Enlarging a model strengthens a single forward pass, whereas recurrent application of a fixed backbone can increase the unrolled computational depth to T times the depth of one pass. **b**: Across rounds, an LS-MDM preserves and refines continuous latent representations, while a standard MDM feeds back only the partially decoded token canvas.

In a **monotonic MDM** (Ye et al., 2025b), only masked positions may be filled, and every accepted token is thereafter frozen. In a **revision-capable MDM** (Wang et al., 2026), a writable position may later be replaced or returned to `[MASK]`. We call these models $\mathrm{MDM}_{\mathrm{mono}}$ and $\mathrm{MDM}_{\mathrm{rev}}$.

Let the input have length n, and let V be the fixed output vocabulary. In addition to the n read-only input positions, the model has P auxiliary writable positions and an output region of length m, all initially masked, and runs for T rounds.

To study what repeated rounds add beyond a single forward pass, we use a circuit model of computation. Unless stated otherwise, each round is a constant-depth d, polynomial-size, finite-precision Transformer computation with hidden width $\Theta(\log n)$, and can therefore be represented by a polynomial-size AC^0 circuit under the characterization adopted here (Li et al., 2024).

3 Effective Iteration

Deep learning derives much of its power from being deep. Successive layers compose simple transformations into computations that a single layer cannot perform efficiently. The need for depth becomes more pronounced in code, mathematics, and so on. A program may require the result of one operation before executing the next; a proof may establish one lemma before another step becomes valid. As these dependency chains become more complex, the required computation must continue through more successive stages. Iterative models can provide this depth, with one round building on the result of another. Parameter scaling can strengthen each round, but it cannot supply a missing sequence of dependent steps, as shown in Figure 1a.

An MDM appears deep because it repeatedly runs a Transformer. Yet its rounds are connected only through the token canvas. If a round leaves no new state on that canvas, the next round cannot build on its computation. The model has completed another nominal round without extending its computational depth. We call a round **effective** when it preserves progress for later rounds.

An $\mathrm{MDM}_{\mathrm{mono}}$ remains computationally shallow regardless of model size. Its effective rounds are tied to writable positions. Each position can leave `[MASK]` only once, so a constant number of writable positions can preserve only constantly many successive updates. Neither model scaling nor better planning creates another state transition after the available positions are exhausted. The formal consequence is sharper. Even if $\mathrm{MDM}_{\mathrm{mono}}$ runs for polynomially many nominal rounds, uses a polynomial size predictor, and chooses its planner adaptively, a constant number of writable positions keeps the entire computation at constant depth. In effect, all those rounds can be collapsed

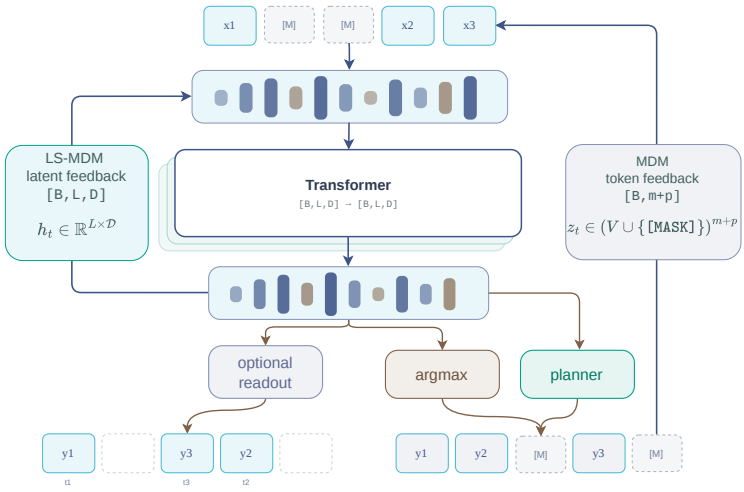


Figure 2: Cross-round feedback in LS-MDM and standard MDM. LS-MDM preserves the full hidden state, whereas standard MDM feeds back only decoded token embeddings.

into one fixed-depth forward pass. Parity, the XOR of a binary sequence, makes this separation exact; the formal result appears in Section B.3.

Revision enables effective rounds. A single rewritable token can carry intermediate state across rounds and compute parity in n rounds, whereas $\mathrm{MDM}_{\mathrm{mono}}$ cannot do so with any polynomial number of nominal rounds.

Efficient revision requires parallel workspace. Reducing the computation to $O(\log n)$ rounds requires many partial results to survive in parallel. Padding provides this token workspace to support multiple intermediate results simultaneously. The formal bounds show that logarithmically many rounds require polynomially many writable positions for parity under the fixed-depth round model. The formal result appears in Section B.4.

Token-state iteration faces a trade-off. A small canvas serializes the computation, while a larger padded canvas exposes more parallelism at a higher cost per forward pass.

4 Preserving Computation

An MDM already computes a hidden state at every position in every round. It then compresses that state into the token canvas and reconstructs a new hidden state in the next round. When the token canvas cannot preserve enough intermediate results across rounds, padding adds more writable positions. Padding uses these extra positions to rebuild the capacity lost during token readout, and every later forward pass must process the longer sequence. A simpler possibility is to preserve the hidden state before it is compressed.

This is the exact principle behind our proposed **Latent-State MDM** (LS-MDM) framework. As shown in Figure 2, LS-MDM carries a persistent latent state across rounds instead of forcing all cross-round information through decoded tokens. Token readout may occur at any round and may participate in later computation, while the latent state remains available to continue the computation. The framework specifies what may survive a round boundary. It does not prescribe a particular latent-state architecture, fusion rule, readout time, or training procedure.

To test whether this framework is effective, we begin with its simplest form. The hidden states produced in one round are passed directly to the next, and token readout occurs only after the final round. We add no padding, latent-state module, gate, or specialized decoding rule. This minimal design lets us test whether preserving latent state alone is enough to change the computation.

This minimal change brings a direct compute advantage. Consider the same Transformer backbone with hidden width $\mathcal{D}$. Without padding, LS-MDM processes a tensor of shape $(n+m) \times \mathcal{D}$ in

every round. An MDM using P padding positions processes $(n + m + P) \times \mathcal{D}$. Padding therefore increases the feedforward cost linearly with sequence length and the attention cost quadratically. LS-MDM keeps the original sequence length, and the separation below shows that it can also require far fewer rounds. The full capacity-matched accounting appears in Section B.5.

LS-MDM retains every computation already available to $\mathrm{MDM}_{\mathrm{rev}}$. Its persistent latent state can encode the complete token canvas, including planner decisions, direct revisions, and remasking operations. LS-MDM can consequently simulate $\mathrm{MDM}_{\mathrm{rev}}$ round for round without auxiliary padding, as shown in Figure 1b. The formal result appears in Section B.6.

The stronger consequence concerns efficiency. Concretely, we prove the following theorem:

Theorem 1 (Persistent state and rounds)**.** *Under the fixed-depth AC^0 round model and a fixed vocabulary, computing parity requires*

$$\begin{aligned} \mathrm{MDM}_{\mathrm{mono}}: &\quad (P+m)\log(1+T|V|) = \Omega\left(n^{1/(d+1)}\right), \\ \mathrm{MDM}_{\mathrm{rev}}: &\quad T(P+m) = \Omega\left(n^{1/(d+1)}\right), \end{aligned}$$

while an LS-MDM computes parity with $P = 0$ in $T = \lceil \log_2 n \rceil$ rounds.

The full statement and proof appear in Section B.7. All three models use the same backbone. The first two lines give the persistent-state budgets of $\mathrm{MDM}_{\mathrm{mono}}$ and $\mathrm{MDM}_{\mathrm{rev}}$. Effective rounds and persistent-state capacity are complementary resources, and their joint budget controls how much computation can continue across round boundaries.

Constant writable token space yields an unconditional strict hierarchy in round complexity. $\mathrm{MDM}_{\mathrm{mono}}$ cannot compute parity under any polynomial round budget. $\mathrm{MDM}_{\mathrm{rev}}$ closes the expressivity gap at the cost of polynomial round complexity, while LS-MDM finishes in $\lceil \log_2 n \rceil$ rounds with zero padding. LS-MDM therefore reduces the computational depth without adding the sequence length, activation memory, or per-round FLOPs required by token workspace.

5 When No Answer Is Ready

The previous section derives its sharpest separation when $P + m = O(1)$. Suppose instead that $\mathrm{MDM}_{\mathrm{rev}}$ receives a long output region and may reuse every position. Does enough token space remove the computational difference from LS-MDM? If those positions may store arbitrary temporary symbols, the answer is essentially yes.

The reason is constructive. Extend the vocabulary with temporary symbols that encode latent values. During intermediate rounds, $\mathrm{MDM}_{\mathrm{rev}}$ writes these symbols into the output region and updates them as arbitrary registers. The final round replaces them with answer tokens. By doing so, $\mathrm{MDM}_{\mathrm{rev}}$ can simulate a narrow LS-MDM round for round. The formal result appears in Section B.9.

This simulation, however, reveals a deeper limitation of the standard MDM interface. It succeeds only after the output region is allowed to contain symbols that are not answers.

Sudoku makes the distinction concrete. Before solving a cell, a model may determine that its value lies in $\{3, 7\}$, that assigning 5 creates a contradiction, or that two distant cells obey a coupled constraint. Each conclusion is genuine computational progress and can determine what becomes possible later. A standard output position cannot preserve it. Writing a digit commits to a final answer and may introduce an error. Writing `[MASK]` discards the conclusion, so the next round must begin again without it. The model can guess, or it can remain silent. Neither choice carries the unfinished computation forward.

The model must remain silent about what it cannot say.
The limits of language then become the limits of the MDM.

We formalize this restriction through an **answer-faithful MDM**. Padding is unavailable as token workspace, and every visible output token is already its correct final answer. We further call an output position **ready** when one round can determine its final token from the input and the correct

answers already visible. An **answer derivation chain** repeatedly reveals ready positions until the output is complete.

These conditions are deliberately favorable to the MDM. Every committed token is assumed correct. The planner sees the full input, may choose a different order for every instance, and may reveal every currently ready position in parallel. The only restriction is semantic: the model may preserve answers, while it may not place non-answer computations in the output region. However, we prove that this restriction alone imposes a sharp computational limit:

Theorem 2 (Answer derivation law)**.** *Under the fixed-depth* AC^0 *round model, an answer-faithful MDM that follows the ready-position protocol can compute a task in* T *rounds only if the task admits an answer derivation chain of length at most* T*. If no such chain exists, neither* $\mathrm{MDM}_{\mathrm{mono}}$ *nor* $\mathrm{MDM}_{\mathrm{rev}}$ *succeeds in any finite number of rounds.*

The formal characterization and proof appear in Section B.10.

The theorem identifies an entire class of unreachable tasks for answer-faithful MDMs. Their computation may advance internally, yet this progress does not become a final answer at any output position. Since each round must preserve progress by committing an answer, additional rounds have nothing from which to continue.

This restriction, however, does not apply to LS-MDM. With sufficient latent width, its persistent latent state can preserve progress before any output position is ready, so answer derivation depth is not a general lower bound on LS-MDM round complexity.

The limitation remains strict even when the ready-position protocol is relaxed. There are sequence tasks whose output region provides one position per input token, while every correct answer token depends on an intermediate result that no output position can preserve.

Theorem 3 (Strict separation)**.** *Under the fixed-depth* AC^0 *round model, there exists a sequence task with one output position per input token that no answer-faithful* $\mathrm{MDM}_{\mathrm{mono}}$ *or* $\mathrm{MDM}_{\mathrm{rev}}$ *computes in polynomially many rounds, even when the planner observes the full input and chooses an input-dependent reveal set. LS-MDM computes the same task with zero padding in* $\lceil \log_2 n \rceil + O(1)$ *rounds.*

The task and quantitative statement are given in Section B.11. This separation concerns computational possibility. The output region is already long enough, revision is available, the planner sees the complete input, and every commitment is correct. The answer-faithful MDM still cannot express the intermediate result needed to make progress. LS-MDM preserves that result in its persistent latent state and performs the remaining computation directly.

6 Learning to Revise

The architecture permits revision; the objective must teach it. Our preceding results assumed correct commitments to isolate what the state interface can compute. Standard mask training introduces a separate problem: it conditions on correct partial answers, while inference begins from a fully masked output and may later encounter model-generated errors. This support mismatch leaves revision dynamics underdetermined.

A copy-or-flip task exposes the gap. The target either copies every input bit or flips every bit according to a global property of the input. One revealed answer token identifies the correct choice, making completion trivial, while fully masked inference must infer that choice from the input. Mask loss can therefore approach zero without teaching the decision required at inference.

The complete formal result appears in Section B.12. It separates three abilities that mask loss can conflate. **Solving** derives the answer from the input. **Completing** extends a correct partial answer. **Recovering** repairs an output containing model-generated errors. Standard mask training directly supervises completion; low mask loss alone does not certify solving or recovery.

LS-MDM supplies a state in which intermediate results can survive, but training must still teach the model how to create and update that state along its own trajectories. Our experiments test both architectural capacity and learned state evolution.

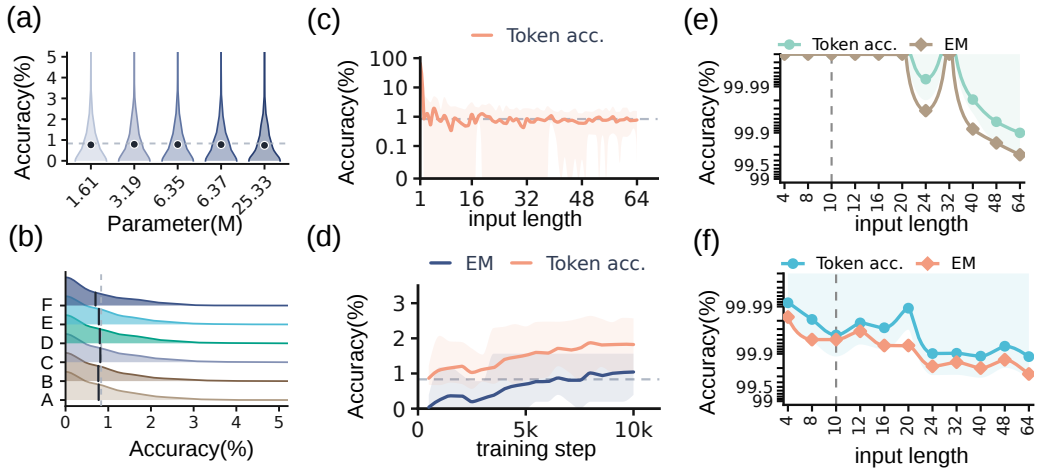


Figure 3: All panels report S_5 composition experiments. **a–b**: Short-output $\mathrm{MDM}_{\mathrm{mono}}$ exact match at the final checkpoint, grouped by model size and planner. Planners A–F are right-to-left, left-to-right, Top-K entropy, Top-K log probability, Top-K margin, and random. Violins and ridges aggregate 1,152 and 960 estimates, respectively; markers show means. **c–d**: Product-conditioned broadcast without an answer derivation chain. Panel c shows token accuracy across input lengths; panel d shows exact match and token accuracy during training. Lines are means over 90 model-seed-planner summaries, and bands span their 2.5th–97.5th percentiles. **e–f**: Prefix-product composition for LS-MDM and $\mathrm{MDM}_{\mathrm{mono}}$, respectively. Lines show mean exact match and token accuracy over three seeds on a logarithmic error scale; bands span the 2.5th–97.5th seed percentiles of token accuracy. Vertical dashed lines mark the maximum training length $n = 10$. Each length uses 2,000 examples per seed. Horizontal dashed lines denote random-guess accuracy.

7 Result

Our experiments ask three questions. First, can persistent latent state help a model learn an iterative computation and continue it beyond the horizons seen during training? The permutation-composition, Rule 90, and Rule 110 experiments test this question under increasingly demanding state transitions. Second, is the advantage explained simply by having more output positions, or does it depend on whether intermediate computation can be represented in those positions? The three permutation-composition interfaces isolate this distinction. Third, does persistent latent state remain useful on a practical constraint-solving task, where useful deductions often arise before any answer is ready? Sudoku-Extreme tests this setting at matched model size and against substantially larger MDMs.

7.1 Permutation composition

Let S_5 denote the 120 permutations of five elements. Composing an ordered sequence from S_5 captures the growing dependency of successive state updates in programs and plans. We test three output formulations for this same computation. The first returns only the final product as a five-token ordering. We vary model size, planner, and random seed, generate training data online for lengths 1 through 64, and evaluate on training data. Every $\mathrm{MDM}_{\mathrm{mono}}$ configuration remains near random-guess accuracy, whereas LS-MDM trained only up to length 10 achieves 100% exact match through length 64 (Figure 3**a–b**). Section E.1 gives the formal definition and complete results.

The first failure might be attributed to having only a constant number of writable positions. The second formulation therefore provides n output positions and n rollout rounds, but makes every position depend on the final global product. Under the same model, planner, seed, and training-set controls, MDMs solve only the smallest lengths and otherwise remain near random-guess accuracy throughout optimization (Figure 3**c–d**). LS-MDM remains near-perfect through length 48, despite training only up to length 10. Section E.2 gives the complete definition, protocol, and results.

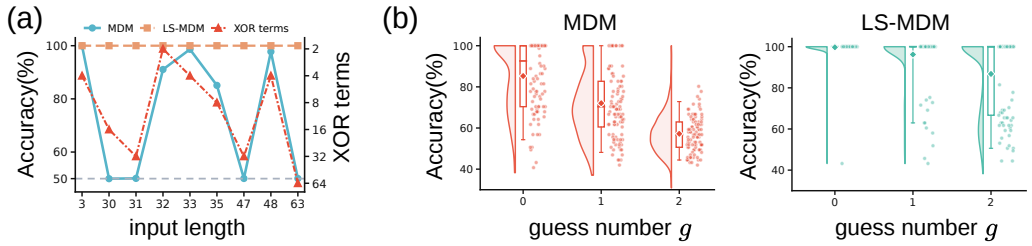


Figure 4: Diagnostics for iterative computation and constraint solving. **a**: Rule 90 mean token accuracy over three seeds on a periodic ring with $n = 256$; triangles show the XOR terms $2^{\mathrm{popcount}(D)}$ on the inverted right axis. The horizontal line marks 50% random-guess accuracy. **b**: Per-puzzle Sudoku token accuracy for the 6.6M MDM and LS-MDM, grouped by minimum correct guesses g. Half-violins show distributions; boxes span the interquartile range, center lines mark medians, whiskers span the 5th–95th percentiles, and diamonds mark means. The $g = 0, 1, 2$ groups contain 279, 290, and 1,431 puzzles, respectively.

The third formulation changes only which correct answers are available along the way. Output position i contains the product of the first i input permutations. This creates an answer derivation chain along which an MDM can carry the computation forward. When both models are trained only up to length 10 and evaluated through length 64, $\mathrm{MDM}_{\mathrm{mono}}$ and LS-MDM both achieve near-perfect accuracy (Figure 3**e–f**). Together, the three experiments show that additional answer positions help an MDM precisely when they expose a usable sequence of intermediate results. Section E.3 gives the complete results.

7.2 Learning State Evolution

Rule 90 and Rule 110 test whether a model can learn a local state-transition rule and apply it repeatedly. Both tasks begin with a binary row arranged on a ring. At each step, every cell is updated from its local neighborhood, and the model must predict the complete row after D steps. An iterative solution retains the row produced at step t and uses it to compute step $t + 1$. These intermediate rows are necessary computational states, but they are not final answers. The two rules provide complementary tests. Rule 90 lets us diagnose whether a model executes the recurrence or learns a direct map to the endpoint, while Rule 110 tests longer and less compressible state evolution.

Rule 90 distinguishes iterative execution from direct endpoint prediction. Each step replaces a cell with the XOR of its two neighbors, yet after D steps each output can depend on between 2 and 64 input cells. MDM accuracy tracks this changing endpoint complexity and falls to random guess at $D = 30, 31, 47$, and 63. LS-MDM remains near-perfect across all evaluated depths, consistent with carrying the evolving row across rounds (Figure 4**a**). Section E.4 gives the exact dependence and complete results.

Rule 110 tests whether learned iteration survives a long run. The model repeatedly applies the same simple local update, but every new row must be computed from the exact row produced one step earlier. The computation therefore forms one continuous trajectory in which an early error can corrupt every subsequent step. We ask whether a model trained only through 40 steps can keep this process correct over substantially longer runs.

LS-MDM remains essentially perfect through $D = 63$, then completes all 200 non-cyclic evaluation trajectories exactly at $D = 64, 80$, and 96. At $D = 128$, it achieves 66.7% exact match. The strongest prompted language model declines from $10/10$ correct examples at $D = 64$ to $6/10$ at $D = 96$ and $5/10$ at $D = 128$. LS-MDM thus continues a learned state evolution for more than twice its training horizon, consistent with preserving each intermediate state for the next update. Table 1 reports the comparison, and Section E.5 gives the complete protocol and results.

7.3 Sudoku-Extreme

Table 1: Rule 110 accuracy on long non-cyclic trajectories. Prompted models use a common 10-example subset of the 200 examples evaluated by trained models.

Model	Samples	$D = 64$		$D = 80$		$D = 96$		$D = 128$	
		EM	Token acc.	EM	Token acc.	EM	Token acc.	EM	Token acc.
Masked MDM	200	0.0	53.4	0.0	55.6	0.0	54.5	0.0	54.1
Revise-MDM	200	0.0	56.5	0.0	55.0	0.0	48.4	0.0	53.1
LS-MDM(Ours)	200	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**66.7**	**84.0**
GPT-6 Astra	10	**100.0**	**100.0**	70.0	92.3	60.0	89.5	50.0	80.5
GPT-5.6 Sol	10	30.0	73.4	0.0	62.3	0.0	51.1	0.0	49.8
GPT-5.6 Luna	10	0.0	24.2	0.0	26.0	0.0	30.7	0.0	34.6
DeepSeek V4.1 Flash	10	20.0	64.5	0.0	45.9	0.0	49.3	0.0	46.7
Llama 4 Maverick	10	0.0	52.3	0.0	50.5	0.0	46.6	0.0	48.7

Sudoku-Extreme tests whether persistent state helps when useful deductions arise before individual cells are ready to be filled. With the same backbone and parameter count, LS-MDM substantially outperforms every MDM decoding schedule. It also remains stronger than all larger MDM baselines, showing that the gain cannot be recovered by changing the decoding schedule or scaling the backbone alone. Table 2 reports the best result at each model scale, and Section E.6 gives the task, evaluation protocol, and full decoding sweep.

Table 2: Best Sudoku-Extreme exact match at each model scale.

Model	Params (M)	EM (%)
MDM	6.6	8.95
MDM	38.1	53.50
MDM	85.5	53.20
MDM	113.8	56.75
MDM	151.7	53.10
MDM	302.9	55.40
LS-MDM	6.6	**73.95**

Note. For each MDM scale, we report the best result across decoding strategies. The full decoding sweep and architecture details appear in Table 12.

To test whether this advantage grows when answer-only progress stalls, we stratify puzzles by the minimum number g of correct solution cells that must be revealed after deterministic propagation reaches a fixed point. Puzzles with $g = 0$ can be completed through answer propagation alone, while larger g indicates that propagation reaches a state from which no answer is immediately available. The matched MDM degrades sharply as g increases and solves no puzzle in the $g = 2$ group. LS-MDM remains substantially more robust across these groups (Figure 4**b**). This pattern connects the empirical result to the state limitation studied in our theory: persistent latent state is most useful when computation must continue before another answer can be committed.

8 Discussion

In this study, we have presented LS-MDM, a general framework that separates persistent computational state from token readout in masked generation. We establish unconditional state and round bounds under an explicit fixed-depth AC^0 model, prove that LS-MDM contains revision-capable MDMs, and identify sequence tasks that faithful token-state MDMs cannot solve within any polynomial round budget. Across permutation composition, Rule 90, Rule 110, and Sudoku-Extreme, the same minimal realization learns reusable state evolution, continues computation beyond its training horizons, and outperforms matched and substantially larger token-state baselines. Together, these results establish the cross-round state interface as a fundamental computational resource alongside model width, parameter count, and per-round depth. Partial reductions, evolving trajectories, and intermediate constraints need a state in which they can persist before becoming answers. LS-MDM provides a foundation for designing iterative generative systems around this principle while remaining independent of any particular memory architecture, feedback rule, or readout schedule. The present results remain bounded by the stated circuit assumptions and controlled evaluation domains; persistent capacity does not itself guarantee learnability, and practical benefits will depend on optimization, memory, hardware, and validation on broader language tasks.

AI USE STATEMENT

We primarily used GitHub Copilot Chat and the GitHub Copilot CLI. During the research process, we used these tools to brainstorm research directions and to assist our interpretation of papers. We also used them to draft and polish portions of the manuscript, including the exposition surrounding mathematical proofs, and to check proofs for possible gaps or inconsistencies. We developed the mathematical claims and proofs ourselves; they were not generated by AI. Generative AI assisted in writing code that was subsequently used to generate synthetic datasets; we reviewed the AI-assisted code and verified the resulting data against the task definitions. We further used AI tools for literature retrieval and discovery. For every work identified or interpreted with AI assistance, we opened the original paper and manually verified the relevant details against its PDF before citing it. We reviewed all AI-assisted text, code, and analyses and take responsibility for the final content of this work. This statement was also drafted with generative AI assistance and reviewed by us.

REPRODUCIBILITY STATEMENT

Code and resources for reproducing our experiments are available here.

REFERENCES

David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 1–5, 1986.

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model. *Advances in Neural Information Processing Systems*, 38: 24415–24459, 2026.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. In *International Conference on Learning Representations*, volume 2026, pp. 15723–15750, 2026.

John Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pp. 6–20, 1986.

Haoyu He, Katrin Renz, Yong Cao, and Andreas Geiger. Mdpo: Overcoming the training-inference divide of masked diffusion language models. *arXiv preprint arXiv:2508.13148*, 2025.

Michael Hersche, Samuel Moor-Smith, Thomas Hofmann, and Abbas Rahimi. Soft-masked diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 105685–105716, 2026.

Emiel Hoogeboom, Alexey A Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive diffusion models. *arXiv preprint arXiv:2110.02037*, 2021.

Haozhe Jiang, Nika Haghtalab, and Lijie Chen. Diffusion language models are provably optimal parallel samplers. In *International Conference on Learning Representations*, volume 2026, pp. 28189–28203, 2026.

Mingyu Jo, Jaesik Yoon, Justin Deschenaux, Caglar Gulcehre, and Sungjin Ahn. Loopholing discrete diffusion: Deterministic bypass of the sampling wall. In *International Conference on Learning Representations*, volume 2026, pp. 93890–93918, 2026.

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

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Koo, Nam Ik Cho, et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. In *International Conference on Learning Representations*, volume 2026, pp. 89900–89938, 2026.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.

Sanghyun Lee, Chunsan Hong, Seungryong Kim, Jonghyun Lee, Jongho Park, and Dongmin Park. Looped diffusion language models. *arXiv preprint arXiv:2605.26106*, 2026.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*, 2025.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, volume 2024, pp. 11911–11943, 2024.

Sulin Liu, Juno Nam, Andrew Campbell, Hannes Stärk, Yilun Xu, Tommi Jaakkola, and Rafael Gómez-Bombarelli. Think while you generate: Discrete diffusion with planned denoising. In *International Conference on Learning Representations*, volume 2025, pp. 81979–82026, 2025.

Yuxi Liu, Konpat Preechakul, Kananart Kuwaranancharoen, and Yutong Bai. The serial scaling hypothesis. In *International Conference on Learning Representations*, volume 2026, pp. 106809–106839, 2026.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

Will Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. *Advances in Neural Information Processing Systems*, 38:112497–112524, 2026a.

Will Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. *Advances in Neural Information Processing Systems*, 38:95315–95339, 2026b.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. In *International Conference on Learning Representations*, volume 2025, pp. 82974–82997, 2025.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646, 2026.

Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. Path planning for masked diffusion model sampling. *arXiv preprint arXiv:2502.03540*, 2025.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Anej Svete and Ashish Sabharwal. On the reasoning abilities of masked diffusion language models. In *International Conference on Learning Representations*, volume 2026, pp. 139646–139681, 2026.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025. doi: 10.48550/arXiv.2506.21734. URL https://arxiv.org/abs/2506.21734.

Guanghan Wang, Yair Schiff, Subham Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *Advances in Neural Information Processing Systems*, 38:147282–147339, 2026.

Kejing Xia, Mingzhe Li, Lixuan Wei, Zhenbang Du, Xiangchi Yuan, Dachuan Shi, Qirui Jin, and Wenke Lee. Metastate: Persistent working memory enhances reasoning in discrete diffusion language models. *arXiv preprint arXiv:2603.01331*, 2026.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *International Conference on Learning Representations*, volume 2025, pp. 77875–77898, 2025a.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025. *URL https://arxiv. org/abs/2508.15487*, 2025b.

Cai Zhou, Chenxiao Yang, Yi Hu, Chenyu Wang, Chubin Zhang, Muhan Zhang, Lester Mackey, Tommi Jaakkola, Stephen Bates, and Dinghuai Zhang. Coevolutionary continuous discrete diffusion: Make your diffusion language model a latent reasoner. *arXiv preprint arXiv:2510.03206*, 2025.

A Related Work

Masked diffusion models and remedies for reasoning. Masked diffusion models (MDMs) learn to reconstruct masked tokens and generate by filling a masked canvas over repeated rounds, in any order (Lou et al., 2023). They have since been scaled to billions of parameters (Nie et al., 2025; 2026; Ye et al., 2025b). Reports of strong reasoning performance (Ye et al., 2025b) coexist with evidence that parallel decoding degrades when output tokens are strongly dependent (Kang et al., 2026) and that low sequence-level error can require many sampling steps (Feng et al., 2026). Existing remedies fall into three families: improved token-selection schedules, ranging from adaptive confidence or margin rules (Kim et al., 2025) to learned selection policies (Liu et al., 2025); larger models (Nie et al., 2025; 2026; Ye et al., 2025b); and revision mechanisms that remask or overwrite previously decoded tokens (Wang et al., 2026; Peng et al., 2025). We use the last term strictly. The low-confidence remasking in LLaDA applies to proposals in the current round and does not revisit committed tokens, so LLaDA and Dream are monotonic MDMs, whereas ReMDM and P2 are revision-capable MDMs (Nie et al., 2026; Ye et al., 2025b; Wang et al., 2026; Peng et al., 2025). Our theory grants these approaches input-dependent token selection, polynomial model size, and unrestricted rewriting, then asks what the token-only state passed between rounds still excludes.

Theory of diffusion language models. Feng et al. (2026) show that MDMs can attain near-optimal token-level perplexity in a number of steps independent of sequence length, while low sequence-level error can require linearly many steps. Jiang et al. (2026) show that revision or remasking increases the power of diffusion language models as parallel samplers: constant-depth circuits can sample the uniform distribution over even-parity strings in $O(1)$ steps with either mechanism, but cannot do so without them. Liu et al. (2026) argue that diffusion models with a TC^0 denoiser remain in TC^0 regardless of the number of denoising steps. Closest to our setting, Svete & Sabharwal (2026) prove an equivalence between finite-precision MDMs that may resample visible positions and polynomially padded looped Transformers. Their results imply that constant-step MDMs with polynomial padding compute exactly AC^0, while regular languages require only logarithmically many steps with linear padding. Zhou et al. (2025) show that continuous diffusion induces richer distribution families than discrete diffusion and can reproduce looped-Transformer rollouts.

Our results address a different computational question. We study the exact computation of a function of the input, while prior sampling results characterize distributions and sampling error (Feng et al., 2026; Jiang et al., 2026). Each round in our model is an arbitrary constant-depth transition with input-dependent token selection, rather than an approximation to a fixed reverse process (Liu et al., 2026). Under this model, revision creates effective iteration and computes parity in n rounds with one rewritable token. Our round-for-round containment of $\mathrm{MDM}_{\mathrm{rev}}$ in LS-MDM and our reverse simulation with temporary symbols parallel the two directions established by Svete & Sabharwal (2026, Theorem 3.1). Constant-round, polynomially padded MDMs are therefore already known to

exclude parity. We add quantitative state bounds, including $T(P+m) = \Omega(n^{1/(d+1)})$ for $\mathrm{MDM}_{\mathrm{rev}}$, and an AC^0 ceiling under every polynomial round budget for $\mathrm{MDM}_{\mathrm{mono}}$ with $O(1)$ writable positions. Answer-faithfulness further shows why the non-answer symbols used by token simulations matter: even with one output position per input token and every commitment correct, some sequence task remains unsolvable in polynomially many rounds. The round separation is sharpest when $P + m = O(1)$. With polynomial padding, $\mathrm{MDM}_{\mathrm{rev}}$ can also achieve logarithmically many rounds (Svete & Sabharwal, 2026), at greater per-round cost. The continuous-versus-discrete inclusion of Zhou et al. (2025) is measure-theoretic, while our separations count rounds and writable positions in an explicit circuit model. Our main results are unconditional under the stated fixed-depth AC^0 round model; only the threshold-circuit extensions assume $\mathrm{TC}^0 \neq \mathrm{NC}^1$.

Circuit complexity of Transformers. Our round model follows the circuit-complexity view of Transformers (Strobl et al., 2024). Constant-depth Transformers with constant precision lie in AC^0 (Li et al., 2024), whereas log-precision Transformers lie in uniform TC^0, which contains parity (Merrill & Sabharwal, 2023). The AC^0 round model is therefore an explicit precision-dependent assumption. Our threshold-circuit extensions cover the TC^0 setting through NC^1-complete problems such as the S_5 word problem (Barrington, 1986). The unconditional lower bounds use the size lower bound for parity in constant-depth circuits (Hastad, 1986). Several mechanisms can take a fixed-depth Transformer beyond one pass. Chain of thought adds serial steps through generated tokens (Li et al., 2024); padding adds parallel workspace, while looping adds compositional depth (Merrill & Sabharwal, 2026a). In particular, a Transformer unrolled $\lceil \log_2 n \rceil$ times recognizes every regular language (Merrill & Sabharwal, 2026b). Our logarithmic-round LS-MDM construction for parity is an instance of this broader phenomenon. Our lower bounds concern the complementary question of what a token-state MDM can preserve between rounds. The trajectory-expansion argument represents T rounds by a constant-depth circuit indexed by possible canvas trajectories, making the number of distinct persistent trajectories, rather than the nominal number of rounds, the controlling resource.

Looped Transformers and latent state in diffusion language models. Looped Transformers repeatedly apply a shared block and can simulate chain-of-thought steps in latent space (Saunshi et al., 2025). LoopMDM loops layers inside each denoising pass, increasing the depth of one round while still passing only tokens between rounds (Lee et al., 2026). Other methods widen the cross-round state interface. Soft-Masked DLM feeds back distributions over likely tokens at masked positions (Hersche et al., 2026); Loopholing carries the final hidden state across the sampling operation (Jo et al., 2026); MetaState adds persistent recurrent working memory (Xia et al., 2026); and co-evolutionary diffusion jointly evolves tokens and continuous representations (Zhou et al., 2025). We view these methods as instances of a broader principle: intermediate state that is not yet an answer must be able to survive a round. Our framework characterizes why this principle changes computational capacity without prescribing a particular memory architecture, feedback rule, readout time, or training objective. The minimal instance studied here carries the complete hidden state, samples no intermediate tokens, and adds no memory module, gate, padding, or specialized decoding rule. Soft-Masked DLM and Loopholing instead retain a token readout at each step and use stop-gradient training for the carried state (Hersche et al., 2026; Jo et al., 2026). The LS-MDM framework is defined by the requirement that non-answer latent state can persist across rounds, independent of any particular recurrent architecture, feedback rule, or readout schedule. This framework admits a broad range of realizations, including looped Transformers, explicit memory modules, and hybrid token-latent feedback. To preserve theoretical generality and isolate the role of persistent state, we study its simplest realization: a looped Transformer that passes the full hidden state between rounds and reads out tokens only after the final round. We do not present the looped architecture itself as the contribution. We use it as the minimal realization needed to isolate the framework's state principle. Which realization best supports learning remains open.

Training revision dynamics. Standard mask training conditions on correct partial answers, while inference begins from a fully masked output and may later encounter model-generated errors. MDPO addresses the mismatch between random training masks and the progressive inference schedule by optimizing over denoising trajectories (He et al., 2025). Our mask-risk theorem isolates a distinct support mismatch that remains even under a matched schedule. Predictors with identical mask loss can differ in whether they correct or preserve an erroneous state, and mask loss can approach zero while exact match from the fully masked state remains at most $1/2$. Low mask loss therefore

certifies completion from correct partial answers without certifying independent solving or recovery from errors.

Benchmarks. S_5 composition instantiates the canonical NC^1-complete bounded-width branching-program construction (Barrington, 1986). Cellular automata provide canonical examples of inherently serial computation (Liu et al., 2026), which motivates our Rule 90 and Rule 110 evaluations. Sudoku-Extreme was introduced with the Hierarchical Reasoning Model (Wang et al., 2025); recurrent latent-state solvers such as Tiny Recursive Models also report strong accuracy at model sizes comparable to ours (Jolicoeur-Martineau, 2025). These results are consistent with the value of carrying latent state across iterations. Our Sudoku experiment is a controlled architectural comparison rather than a state-of-the-art claim: the MDM and LS-MDM share the same backbone, isolating the cross-round state interface as the experimental variable.

B Formal Model and Proofs

B.1 Circuit and state conventions

All circuit classes in this appendix are nonuniform. An AC^0 circuit has unbounded fan-in AND and OR gates, negated input literals, constant depth, and polynomial size. Depth counts AND and OR layers. By De Morgan dualization, both polarities of every internal output can be supplied with constant-factor size overhead and no additional asymptotic depth. The unconditional results below concern an explicitly assumed AC^0 round model. Finite precision alone does not imply this circuit upper bound for an arbitrary Transformer. The conditional threshold-circuit counterparts are stated separately in Appendix C.

Unless a theorem states otherwise, T, P, and m are polynomially bounded in n; every round transition may depend on the input length and round index, has depth at most a fixed constant d, and has size at most $s(n) = n^{O(1)}$. The transition includes both the predictor and planner.

Let $\mathcal{A} = V \cup \{\perp\}$ and $q = P + m$. A token configuration is an element of $\mathcal{A}^q$, with initial configuration $\perp^q$. For each round $t \in \{1, \dots, T\}$, a token model applies a circuit

$$F_t : \Sigma^n \times \mathcal{A}^q \longrightarrow \mathcal{A}^q.$$

A monotonic transition may change a coordinate from $\perp$ to an element of V and may never change that coordinate again. A revision transition may assign any element of $\mathcal{A}$ at every round.

An $\mathrm{LSMDM}_d[T, P, m]_w$ has a recurrent state of exactly $(n + m + p)w$ bits, arranged as w bits at each of the positions. Formally,

$$h_0 = E(x), \qquad h_t = F_t(x, h_{t-1}), \qquad y = R(x, h_T).$$

The input x remains available read-only at every round. The initializer E, every transition F_t, and the final readout R have polynomial size and constant depth; each F_t has depth at most d. This recurrent state convention is the one used by every theorem below. Adding a fixed number of encoding, decoding, initialization, and readout layers changes total depth by $O(1)$.

Write

$$b_{\mathrm{tok}} = \lceil \log_2 |\mathcal{A}| \rceil .$$

For the reverse simulation of arbitrary latent bits by a fixed alphabet, $\log_2 |\mathcal{A}|$ bits per token is the information-theoretic rate. We uses b_{tok} to suppress a vocabulary-dependent constant. The stated bound is exact when $|\mathcal{A}|$ is a power of two. For arbitrary fixed $|\mathcal{A}|$, the information-theoretic position count is

$$P' \geq \left\lceil \frac{nw}{\log_2 |\mathcal{A}|} \right\rceil$$

but need not admit the local constant-depth coding used below. A constructive sufficient bound is

$$P' \geq n \left\lceil \frac{w}{\lfloor \log_2 |\mathcal{A}| \rfloor} \right\rceil .$$

All asymptotic conclusions are unchanged.

For each fixed d, choose $c_d > 0$ small enough that every depth $d+2$ AC^0 circuit computing PARITY_n has size at least $2^{2\theta_d(n)}$, where

$$\theta_d(n) = c_d n^{1/(d+1)}.$$

We also use the standard consequence that a polynomial size AC^0 circuit for parity must have depth $\Omega(\log n / \log\log n)$.

B.2 Trajectory expansion

Definition 1 (State trajectory). *Consider a deterministic T round model with one input-independent initial state z_0 and with outputs given by projections of its final state. Its trajectory on input x is*

$$\tau(x) = (z_0, z_1, \ldots, z_T), \qquad z_t = F_t(x, z_{t-1}).$$

Let $\mathfrak{T}_n$ be any set containing every trajectory realized on inputs of length n, with the same fixed initial coordinate z_0 in every candidate, and write $N(n) = |\mathfrak{T}_n|$.

Theorem 4 (Trajectory expansion). *Suppose every round transition has depth at most d and size at most $s(n)$, the initial state is input-independent, and every output bit is a projection of the final state. Every output bit of the T round model is computed by a depth at most $d+2$ circuit of size*

$$O(N(n)T(s(n) + |z_t|)).$$

The statement holds for both AC^0 and TC^0 round circuits.

Proof. Fix a candidate trajectory $\tau = (z_0, \ldots, z_T) \in \mathfrak{T}_n$. Since every z_t is fixed within this branch, the predicate

$$\mathrm{Valid}_\tau(x) = \bigwedge_{t=1}^{T} [F_t(x, z_{t-1}) = z_t]$$

is computed by placing all T transition circuits in parallel and applying one unbounded fan-in AND to their output equality tests. Equality with the fixed string z_t requires only literals and one unbounded fan-in AND. These AND gates can be merged with the validity layer, giving depth at most $d+1$.

For an output bit indexed by j, output

$$\bigvee_{\tau \in \mathfrak{T}_n :\, (z_T)_j = 1} \mathrm{Valid}_\tau(x).$$

Determinism guarantees exactly one valid trajectory for every input. The final OR adds one layer. Replicating the transition circuits for all candidates gives the stated size. Unbounded AND and OR are available in both circuit classes. □

Corollary 1 (Trajectory counts). *For $q = P + m$ writable token positions,*

$$N_{\mathrm{mono}}(n) \le (1 + T|V|)^q, \qquad N_{\mathrm{rev}}(n) \le |\mathcal{A}|^{Tq}.$$

If the complete persistent state contains at most b bits at each round, then $N(n) \le 2^{Tb}$.

Proof. In a monotonic trajectory, each coordinate either remains masked or is assigned one pair consisting of a first write time in $\{1, \ldots, T\}$ and a token in V. Revision permits an arbitrary symbol in $\mathcal{A}$ at each coordinate and round. The final claim counts all length T sequences of b bit states. □

B.3 Monotonic ceiling

Theorem 5 (Monotonic ceiling). *Under the fixed-depth AC^0 round model, let $T \le \mathrm{poly}(n)$ and $P + m = O(1)$. Then*

$$\mathrm{MDM}_{\mathrm{mono}}[T, P, m] \subseteq \mathrm{AC}^0.$$

Consequently, no such model computes PARITY_n.

Proof. Let $q = P + m = O(1)$ and $T \le \mathrm{poly}(n)$. Corollary 1 gives

$$N_{\mathrm{mono}}(n) \le (1 + T|V|)^q = \mathrm{poly}(n).$$

Theorem 4 therefore produces a polynomial size, constant depth circuit for every output bit. Hence every function computed by the model lies in AC^0. Since $\mathrm{PARITY} \notin \mathrm{AC}^0$, no such model computes parity. □

B.4 Revision time-space tradeoff

Theorem 6 (Revision time-space tradeoff)**.** *Under the fixed-depth* AC^0 *round model and a fixed vocabulary:*

1. $$\mathrm{PARITY}_n \in \mathrm{MDM}_{\mathrm{rev}}[n, 0, 1].$$

2. *If* $T = O(\log n)$*,* $P = 0$*, and* $m = O(1)$*, then*
$$\mathrm{MDM}_{\mathrm{rev}}[T, 0, m] \subseteq \mathrm{AC}^0,$$
so no such model computes parity.

3. *Any* $\mathrm{MDM}_{\mathrm{rev}}$ *computing parity in* $T = O(\log n)$ *rounds satisfies*
$$P + m = \Omega\left(\frac{n^{1/(d+1)}}{\log n}\right).$$

Proof. For the first claim, use the single output token as a bit register. The token canvas initially contains $\perp$. At round 1, write x_1. At each round $t \in \{2, \ldots, n\}$, replace the register by its XOR with x_t. XOR of two bits is computed by a depth-two Boolean circuit. After n rounds the register equals $\bigoplus_{t=1}^{n} x_t$.

For the second claim, $P = 0$, $m = O(1)$, and $T = O(\log n)$. Therefore

$$N_{\mathrm{rev}}(n) \le |\mathcal{A}|^{Tm} = n^{O(1)}.$$

Theorem 4 gives a polynomial-size AC^0 circuit for every output bit, which excludes parity.

For the third claim, let $q = P + m$. If the model computes parity, Theorem 4 and the parity size lower bound imply

$$2^{2\theta_d(n)} \le O\big(|\mathcal{A}|^{Tq} T(s(n) + q)\big).$$

Taking binary logarithms and using polynomial resource bounds gives

$$Tq \log_2 |\mathcal{A}| \ge \theta_d(n) - O(\log n).$$

The vocabulary is fixed, so $\log_2 |\mathcal{A}| = \Theta(1)$. Substituting $T = O(\log n)$ yields the stated lower bound. □

B.5 Compute and capacity accounting

Let $L_0 = n + m$, and let

$$C(L) = \Theta(L^2 D + L D^2)$$

denote the dominant cost of one dense Transformer layer with hidden width D and sequence length L. If an MDM uses P padding positions, while LS-MDM uses none, then

$$\frac{\mathrm{FLOPs}_{\mathrm{MDM}}}{\mathrm{FLOPs}_{\mathrm{LSMDM}}} \asymp \frac{T_{\mathrm{MDM}}}{T_{\mathrm{LSMDM}}} \frac{C(L_0 + P)}{C(L_0)}.$$

One token position carries at most $b_{\mathrm{tok}} = \lceil \log_2 |\mathcal{A}| \rceil$ representation bits. A D-dimensional latent vector with p-bit coordinates contains $w = Dp$ bits. Under the power-of-two alphabet convention, directly matching the nw latent bits requires

$$P' = n \left\lceil \frac{w}{b_{\mathrm{tok}}} \right\rceil$$

token positions. For an arbitrary fixed alphabet, the constructive sufficient bound in Appendix B.1 applies. Substituting P' into the FLOPs ratio gives the capacity-matched accounting.

B.6 ROUND-FOR-ROUND CONTAINMENT

Theorem 7 (Round-for-round containment)**.** *Under the finite-precision circuit model,*

$$\mathrm{MDM}_{\mathrm{rev}}[T, P, m] \subseteq \mathrm{LSMDM}[T, 0, m]_w$$

whenever

$$w \geq b_{\mathrm{tok}} \left\lceil \frac{P+m}{n} \right\rceil .$$

Proof. The writable token canvas has $q = P + m$ coordinates and therefore admits an injective binary encoding of length at most qb_{tok}. The hypothesis

$$w \geq b_{\mathrm{tok}} \left\lceil \frac{q}{n} \right\rceil$$

implies $nw \geq qb_{\mathrm{tok}}$. Store the encoded canvas across the n persistent latent positions. One LS-MDM round decodes this state, applies the original predictor and planner transition, and re-encodes the updated canvas. Encoding and decoding are fixed wiring operations with $O(1)$ additional depth. The final readout decodes the m output coordinates. This simulates every revision and remasking operation and preserves the number of rounds. □

B.7 PERSISTENT STATE AND ROUNDS

Theorem 8 (Persistent-state bounds)**.** *Under the fixed-depth* AC^0 *round model, every model computing* PARITY_n *satisfies*

$$\begin{aligned} \mathrm{MDM}_{\mathrm{mono}}: &\quad (P+m)\log_2(1+T|V|) \geq \theta_d(n) - O(\log n), \\ \mathrm{MDM}_{\mathrm{rev}}: &\quad T(P+m)b_{\mathrm{tok}} \geq \theta_d(n) - O(\log n). \end{aligned}$$

Moreover,

$$\mathrm{PARITY}_n \in \mathrm{LSMDM}_2[\lceil \log_2 n \rceil, 0, 1]_1,$$

and every polynomial-size $\mathrm{LSMDM}_d[T, P, m]_w$ *computing parity satisfies*

$$Td = \Omega\left(\frac{\log n}{\log \log n} \right).$$

Proof. Let $q = P + m$. For a monotonic MDM, Corollary 1 and Theorem 4 give a depth $d + 2$ circuit of size

$$O((1 + T|V|)^q T(s(n) + q)) .$$

If the model computes parity, the lower bound defining $\theta_d(n)$ yields

$$q \log_2(1 + T|V|) \geq \theta_d(n) - O(\log n).$$

For a revision MDM the same argument with $N_{\mathrm{rev}} \leq |\mathcal{A}|^{Tq}$ gives

$$Tq \log_2 |\mathcal{A}| \geq \theta_d(n) - O(\log n).$$

Since $\log_2 |\mathcal{A}| \leq b_{\mathrm{tok}}$, the displayed revision inequality in the main text follows.

For the LS-MDM upper bound, initially position i stores x_i. At round r, one designated position for each nonempty block of at most 2^r consecutive inputs stores the XOR of the designated values for its two nonempty half blocks; a missing half contributes the constant 0. Two-input XOR has depth two. After $\lceil \log_2 n \rceil$ rounds the first designated position stores $\mathrm{PARITY}_n(x)$. This uses one persistent bit per input position and no auxiliary padding.

For the LS-MDM lower bound, unroll its T round computation directly. The resulting circuit has polynomial size and depth at most $Td + O(1)$. The polynomial size depth lower bound for parity gives

$$Td = \Omega\left(\frac{\log n}{\log \log n} \right).$$

□

B.8 EFFICIENCY SEPARATION

Theorem 9 (Efficiency separation). *For fixed d, every revision MDM with $P + m = O(1)$ that computes parity requires $T = n^{\Omega(1)}$, whereas $\mathrm{LSMDM}_2[\lceil \log_2 n \rceil, 0, 1]_1$ computes parity.*

Proof. Substitute $P + m = O(1)$ and fixed b_{tok} into the revision inequality of Theorem 8. The LS-MDM upper bound is also established in Theorem 8. □

B.9 TOKEN SIMULATION OF LATENT STATE

Theorem 10 (Token simulation of latent state). *Fix $T \geq 1$. For a fixed alphabet $\mathcal{A} = V \cup \{\perp\}$,*

$$\mathrm{LSMDM}_d[T, 0, m]_w \subseteq \mathrm{MDM}_{\mathrm{rev},d+O(1)}[T, P', m]$$

whenever

$$P' \geq n \left\lceil \frac{w}{\lfloor \log_2 |\mathcal{A}| \rfloor} \right\rceil .$$

If $m = \Theta(n)$ and the output alphabet is extended with symbols encoding $O(w)$ temporary bits per position, the same simulation uses $P' = 0$.

Proof. Assume $T \geq 1$. For $T = 1$, the single token transition directly computes $R(x, F_1(x, E(x)))$ and writes the final answer.

For the fixed vocabulary construction, first assume the power of two coding convention in Appendix B.1. Partition the nw latent bits into blocks of b_{tok} bits and encode each block by one token. The displayed bound supplies enough padding positions. Encoding and decoding are independent constant-size lookups on each block and therefore add $O(1)$ depth. For arbitrary fixed alphabet size, use blocks of $\lfloor \log_2 |\mathcal{A}| \rfloor$ bits and the corrected sufficient bound stated in Appendix B.1.

At round 1, the token model uses its read-only access to x to compute $h_1 = F_1(x, E(x))$ directly and writes its encoding. At each intermediate round it decodes h_{t-1}, computes $h_t = F_t(x, h_{t-1})$, and writes the new encoding. In the final round it computes both h_T and

$$R(x, h_T) = R(x, F_T(x, h_{T-1})) ,$$

then writes the final answer tokens into the output region. The depth is $d + O(1)$ and the number of rounds remains T.

For the second construction, assume $m = \Theta(n)$ and extend the output alphabet with enough temporary symbols to encode $O(w)$ bits at each position. The m output positions then hold the complete nw bit latent state. The same transition schedule stores h_t as temporary symbols for $t < T$ and computes $R(x, h_T)$ in the final transition. Both constructions preserve the number of rounds. □

B.10 DERIVATION DEPTH

Definition 2 (Sequence task and one-round derivability). *A sequence task is a family $\mathbf{f} = (f_n)_{n \geq 1}$ with*

$$f_n : \Sigma^n \longrightarrow A^{m(n)}, \qquad A \subseteq V.$$

Fix a constant per round depth bound d and one polynomial p shared by all input lengths, premise sets, and output coordinates. Write $f = f_n$, $m = m(n)$, and $[m] = \{1, \ldots, m\}$. For $R \subseteq [m]$ and $j \in [m]$, write $R \vdash_{d,p} j$ when a circuit of depth at most d and size at most $p(n)$ computes $f(x)_j$ from x and the correct partial answer $f(x)|_R$ for every $x \in \Sigma^n$.

Definition 3 (Derivation closure and depth). *Define*

$$\mathrm{Cl}_{d,p}(R) = R \cup \{j \in [m] : R \vdash_{d,p} j\}, \qquad R^{(0)} = \emptyset, \qquad R^{(t+1)} = \mathrm{Cl}_{d,p}(R^{(t)}).$$

The derivation depth is

$$\mathrm{dd}_{d,p}(f) = \min\{t : R^{(t)} = [m]\}.$$

If no such t exists, set $\mathrm{dd}_{d,p}(f) = \mathsf{ND}$.

Definition 4 (Canonical answer process)**.** *A token process for f is canonical relative to (d, p) when padding remains masked, every nonmask output token written at every round equals its final value in $f(x)$, and every newly written coordinate j satisfies $J_{t-1}(x) \vdash_{d,p} j$, where*

$$J_t(x) = \{j \in [m] : y_t(x)_j \neq \bot\}.$$

The planner may depend on the full input and on the current canvas.

Theorem 11 (Derivation depth characterization)**.** *For every canonical monotonic or revision process computing f in T rounds,*

$$\mathrm{dd}_{d,p}(f) \neq \mathsf{ND}, \qquad T \geq \mathrm{dd}_{d,p}(f).$$

Conversely, if $\mathrm{dd}_{d,p}(f) = k < \infty$, a canonical monotonic MDM with per round size $O(m(n)p(n))$ computes f in $k \leq m$ rounds. Revision neither enlarges the canonical function class nor decreases the minimum round count. The same task is computable by an LS-MDM in k rounds with

$$w = b_{\mathrm{tok}} \left\lceil \frac{m}{n} \right\rceil.$$

Proof. Derivability is monotone in its premise: if $R \subseteq R'$ and $R \vdash_{d,p} j$, then $R' \vdash_{d,p} j$, because a circuit may ignore the additional answers. We prove by induction that

$$J_t(x) \subseteq R^{(t)}$$

for every input x. The claim is trivial at $t = 0$. Suppose it holds at $t - 1$. Every newly written j satisfies $J_{t-1}(x) \vdash_{d,p} j$. By monotonicity and the induction hypothesis, $R^{(t-1)} \vdash_{d,p} j$, so $j \in R^{(t)}$. A model that finishes in T rounds has $J_T(x) = [m]$, hence $R^{(T)} = [m]$ and $T \geq \mathrm{dd}_{d,p}(f)$.

Conversely, at round t reveal every coordinate in $R^{(t)} \setminus R^{(t-1)}$. By definition, each such coordinate has a valid one-round circuit from x and the answers in $R^{(t-1)}$. The coordinate circuits may be run in parallel with total size $O(m(n)p(n))$ and the same asymptotic depth. This gives a canonical monotonic computation in exactly k rounds. The increasing chain can add a coordinate at most m times, so $k \leq m$. A canonical revision process can only remask already correct answers in addition to these writes; this cannot reveal a coordinate before it enters the closure.

Finally, apply Theorem 7 to the canonical monotonic construction with $P = 0$ and m writable answer positions. □

B.11 No valid first token

Define

$$\mathrm{GPAR}_n(x)_i = x_i \oplus \bigoplus_{j=1}^{n} x_j, \qquad i = 1, \ldots, n.$$

An MDM is **answer-faithful** for this task when padding remains masked and every nonmask output token written at every round equals the corresponding final token of $\mathrm{GPAR}_n(x)$. The planner may inspect the full input and choose an input-dependent reveal set.

Theorem 12 (Global flip is not derivable)**.** *For every fixed (d, p), there is $n_0(d, p)$ such that for all $n \geq n_0(d, p)$,*

$$\mathrm{dd}_{d,p}(\mathrm{GPAR}_n) = \mathsf{ND}.$$

Proof. For every coordinate i,

$$\mathrm{GPAR}_n(x)_i = x_i \oplus \mathrm{PARITY}_n(x).$$

If $\emptyset \vdash_{d,p} i$, then XORing the resulting output with x_i would give a depth $d + O(1)$ circuit of size $O(p(n))$ for parity. For all sufficiently large n, this contradicts the parity lower bound. Hence $\mathrm{Cl}_{d,p}(\emptyset) = \emptyset$, and all iterates of the closure remain empty. □

Theorem 13 (No valid first token)**.** *Assume the fixed-depth AC^0 round model, polynomial per-round size, and $P = 0$.*

1. *No answer-faithful monotonic or revision MDM computes GPAR_n in any $T \leq \mathrm{poly}(n)$ rounds.*

2.

$$\mathrm{GPAR}_n \in \mathrm{LSMDM}_2[\lceil \log_2 n \rceil + O(1), 0, n]_1.$$

Proof. Assume first that an answer-faithful token model computes GPAR_n within $T \leq \mathrm{poly}(n)$ rounds. For each round t and coordinate i, construct a circuit $A_{t,i}(x)$ indicating that round t is the first round in which any output is written and that i is the least coordinate written in that round. Before the first write the canvas is fixed to $\perp^n$. Therefore every $A_{t,i}$ and the corresponding proposed token value are computed by evaluating the round circuits on the fixed all-mask canvas. All earlier no-write conditions can be checked in parallel, so $A_{t,i}$ has constant depth and polynomial size.

On every input, exactly one pair (t, i) is selected. Answer faithfulness gives

$$\mathrm{value}_{t,i}(x) \oplus x_i = \mathrm{PARITY}_n(x)$$

on the inputs selected by $A_{t,i}$. Taking the OR over the polynomially many guarded values

$$A_{t,i}(x) \wedge (\mathrm{value}_{t,i}(x) \oplus x_i)$$

produces a polynomial size, constant depth circuit for parity, a contradiction. The argument permits input dependent planners, simultaneous first writes, monotonic updates, and revision.

For the LS-MDM upper bound, compute parity by the binary reduction from the proof of Theorem 8. Retain the resulting bit and, in $O(1)$ additional rounds, XOR it with every x_i in parallel and read out all n coordinates. □

B.12 Mask risk and revision dynamics

Let $X \sim \mathcal{D}_n$, let $S \subseteq \{1, \ldots, m\}$ be the visible answer coordinates sampled from a masking distribution μ, and let $M = \{1, \ldots, m\} \setminus S$. With zero loss assigned when $M = \emptyset$, define

$$\mathcal{L}_{\mathrm{mask}}(p) = \mathbb{E}_{X,S}\left[\frac{1}{|M|}\sum_{j \in M} \ell(p(\cdot \mid X, f(X)|_S, S), f(X)_j)\right].$$

For the first part of the theorem below, revision uses the fixed synchronous rule that writes the predictor proposal at every designated coordinate. For the second part, ℓ is cross entropy, X is uniform on $\{0,1\}^n$, the output length is $m = n$, point-mass output distributions are permitted in the finite-precision predictor model, and the predictor specifies proposals at both visible and masked positions. Greedy ties between the copy and flip hypotheses are broken in favor of copying.

Theorem 14 (Mask risk does not identify revision dynamics)**.** *Assume that the predictor class permits independent behavior outside the joint support of the masked training distribution.*

1. *There exist two predictors with identical $\mathcal{L}_{\mathrm{mask}}$ such that, under the same synchronous update rule, one corrects a fixed erroneous canvas in one round and the other preserves that error indefinitely.*

2. *There exists a unique-answer task and a finite-precision predictor family satisfying*

$$\mathcal{L}_{\mathrm{mask}} = \frac{\log 2}{n+1} \longrightarrow 0,$$

while every permitted greedy revision process initialized from the fully masked output has sequence accuracy at most $1/2$, for every number of rounds.

Proof. For the first claim, let $\mathcal{S}_{\mathrm{train}}$ be the joint support of input-canvas pairs under the masked training distribution. Every pair in this support contains only answers that are correct for its input. Fix an input x_0 and a completed canvas $z \neq f(x_0)$. Then $(x_0, z) \notin \mathcal{S}_{\mathrm{train}}$. By the hypothesis on the predictor class, start from any predictor p_0 and construct two predictors that agree with p_0 on $\mathcal{S}_{\mathrm{train}}$. Under the synchronous write-all rule, define p_{good} at (x_0, z) to propose the complete correct answer $f(x_0)$, and define p_{bad} to propose z itself. Thus p_{good} corrects the canvas in one round, while z is an erroneous fixed point of p_{bad}. The two predictors have identical mask risk.

For the second claim, take X uniformly distributed on $\{0,1\}^n$ and let

$$f_n(x)_j = x_j \oplus \mathrm{PARITY}_n(x), \qquad j = 1, \ldots, n.$$

Thus the target either copies x or flips every input bit, according to the global parity. Sample α uniformly from $[0,1]$ and mask each answer coordinate independently with probability α. If coordinate i is visible, the predictor recovers the global choice as

$$b = f_n(x)_i \oplus x_i$$

and proposes $x_j \oplus b$ at every coordinate j. If all coordinates are masked, it assigns equal probability to the copy and flip hypotheses and greedily chooses copy. Whenever at least one coordinate is visible, every proposal is correct and the cross entropy loss is zero. The probability that all coordinates are masked is

$$\mathbb{E}_{\alpha \sim U[0,1]}[\alpha^n] = \int_0^1 \alpha^n \, d\alpha = \frac{1}{n+1}.$$

The loss conditional on this event is $\log 2$, proving

$$\mathcal{L}_{\mathrm{mask}} = \frac{\log 2}{n+1}.$$

At inference, the fully masked state proposes the copy hypothesis, so coordinate j receives proposal x_j. Once any proposal is written, comparing it with the corresponding input again yields the copy hypothesis. Remasking every position returns to the same fully masked state and the same proposal. Hence every completed answer equals x. This is correct exactly when $\mathrm{PARITY}_n(x) = 0$, which occurs with probability $1/2$. Sequence accuracy is therefore at most $1/2$ for every number of rounds and every planner that can only select or remask proposed values. □

C Conditional Threshold-Circuit Extensions

Definition 5 (Threshold-circuit hypothesis)**.** *The conditional results in this section assume the nonuniform separation*

$$\mathrm{TC}^0 \subseteq \mathrm{NC}^1 \qquad \textit{and} \qquad \mathrm{TC}^0 \neq \mathrm{NC}^1.$$

Fix a Boolean function family $g = (g_n)_{n \geq 1}$ in $\mathrm{NC}^1 \setminus \mathrm{TC}^0$.

Write $\mathrm{MDM}^{\mathrm{TC}}$ and $\mathrm{LSMDM}^{\mathrm{TC}}$ for the corresponding models when every round transition and readout is a polynomial size, constant depth threshold circuit.

C.1 Short-output separation

Theorem 15 (TC^0 version of the short-output separation)**.** *Assume Definition 5. The following token models compute only functions in TC^0:*

1. *monotonic models satisfying*
$$(P+m)\log_2(1+T|V|) = O(\log n);$$
2. *revision models satisfying*
$$T(P+m)b_{\mathrm{tok}} = O(\log n).$$

The function family g is excluded from both classes and is computed by an LS-MDM in $O(\log n)$ rounds using polynomial total persistent state. Consequently, under the hypothesis, latent iteration with polynomial state is strictly stronger than polynomial-round constant-space monotonic iteration and logarithmic-round constant-space revision iteration.

Proof. Under either displayed budget, Corollary 1 gives $N(n) = n^{O(1)}$. The trajectory expansion proof uses only constant-depth threshold circuits and unbounded AND and OR gates, so it yields a polynomial size TC^0 circuit for the output. Hence neither token class computes g. The two stated special cases satisfy the respective trajectory budgets.

It remains to construct the LS-MDM computation without introducing a separate result. Let a bounded fan-in circuit of polynomial size $s(n)$ and depth $O(\log n)$ compute g_n. Store one bit for every gate, distributed across the n latent positions using $\lceil s(n)/n \rceil$ bits per position. At latent round r, retain all previously computed gate values and evaluate in parallel every gate whose circuit depth is r. Each gate has only a constant number of predecessors, so this update has constant depth and polynomial size. After $O(\log n)$ rounds, the readout returns the output gate. □

C.2 Long-output separation

Theorem 16 (TC^0 version of no valid first token)**.** *Assume Definition 5 and define*

$$\mathrm{GF}_{g,n}(x)_i = x_i \oplus g_n(x), \qquad i = 1, \ldots, n.$$

No answer-faithful monotonic or revision $\mathrm{MDM}^{\mathrm{TC}}$ *with polynomially many rounds computes* $\mathrm{GF}_{g,n}$*. An* $\mathrm{LSMDM}^{\mathrm{TC}}$ *computes it in* $O(\log n)$ *rounds with polynomial total persistent state.*

Proof. If an answer-faithful token model computed $\mathrm{GF}_{g,n}$, repeat the first-commitment construction from Appendix B.11. Constant-depth threshold circuits are closed under the polynomially many parallel first-commitment tests and the final guarded OR. The resulting circuit computes

$$\mathrm{GF}_{g,n}(x)_i \oplus x_i = g_n(x)$$

in TC^0, contradicting the choice of g.

For the upper bound, use the layered circuit simulation from the proof of Theorem 15 to compute $g_n(x)$ in latent state, retain the resulting bit, and XOR it with all input coordinates in one additional round. □

D Experimental Details

D.1 Data construction

We use three complementary benchmarks that isolate distinct computational demands: sequential composition (S_5), repeated state evolution (Rule 90/110), and search-intensive constraint satisfaction (Sudoku-Extreme). The S_5 tasks share the same permutation sequences but vary the output from a five-token final product to sequence-length broadcast and prefix targets; the cellular-automaton tasks vary update depth under exact simulation; and Sudoku-Extreme comprises difficult puzzles that require multi-step reasoning and search. Dataset construction details for each benchmark follow.

D.1.1 S_5 composition

Inputs. An instance of length n is a sequence $x = (g_1, \ldots, g_n) \in S_5^n$, where each g_i is drawn uniformly from the 120 permutations of $[5] = \{1, \ldots, 5\}$. Prefix products follow the input-order convention $p_0 = e$ and $p_i = g_i \circ p_{i-1}$, that is, $p_i(j) = g_i(p_{i-1}(j))$. Each permutation g is one token $P_{\kappa(g)}$, where $\kappa(g) \in \{0, \ldots, 119\}$ is its lexicographic rank, so the input is $X_n = P_{\kappa(g_1)} \| \cdots \| P_{\kappa(g_n)}$, with $\|$ denoting concatenation.

Targets and canvas. Each stored input carries the three targets in Table 3, so the three experiments differ only in the output interface. Values $v \in [5]$ use the zero-based tokens O_{v-1}. All methods place the input at fixed source coordinates on a 64-slot canvas, packing each example as

$$X_n \parallel \texttt{[PAD]}^{64-n} \parallel \texttt{[SEP]} \parallel Y \parallel S,$$

where the target Y and the suffix S are given in Table 3. The `[EOS]` position is not supervised.

Splits. The training split contains 100,000 sequences at each length $n \in \{1, \ldots, 10\}$ for LS-MDM, the validation split contains 5,000 sequences over the same lengths, and the test split contains 2,000 sequences at each $n \in \{4, 8, 10, 12, 16, 20, 24, 32, 40, 48, 64\}$; lengths $n > 10$ are length OOD. We sample the validation and test sequences first and exclude them from training, so the three splits are disjoint.

Table 3: Targets for the three S_5 experiments on a shared input X_n.

Experiment	Target	Serialization $\boldsymbol{Y}$	Suffix $\boldsymbol{S}$	$\lvert\boldsymbol{Y}\rvert$
Global product (§E.1)	p_n	$O_{p_n(1)-1} \,\|\, \cdots \,\|\, O_{p_n(5)-1}$	`[EOS]`	5
Product-conditioned broadcast (§E.2)	$z_i = g_i \circ p_n$	$P_{\kappa(z_1)} \,\|\, \cdots \,\|\, P_{\kappa(z_n)}$	`[PAD]`$^{64-n}$	n
Prefix scan (§E.3)	$p_i = g_i \circ p_{i-1}$	$P_{\kappa(p_1)} \,\|\, \cdots \,\|\, P_{\kappa(p_n)}$	`[PAD]`$^{64-n}$	n

D.1.2 Rule 90 and Rule 110

An instance consists of a depth D and an initial row $x^{(0)} \in \{0,1\}^n$ with i.i.d. uniform bits. The target is the complete row $x^{(D)}$ after D synchronous updates with periodic boundary conditions,

$$x_i^{(t+1)} = \phi_r\left(x_{i-1}^{(t)}, x_i^{(t)}, x_{i+1}^{(t)}\right), \qquad i \in \mathbb{Z}_n, \quad r \in \{90, 110\},$$

where $\phi_{90}(a,b,c) = a \oplus c$ and ϕ_{110} is the standard Rule 110 lookup table. Targets are generated exactly by iterating the automaton, and supervision is applied only to the final row. We report cell accuracy.

Data regimes. For both Rule 90 and Rule 110, the matched setting uses $n = 256$. Models train at $D \in \{1,\ldots,10\} \cup \{30,\ldots,40\}$; depths $D \in \{11,\ldots,29\}$ are unseen interpolation, and $D \in \{41,\ldots,63\}$ are depth OOD. Final evaluation reports $D \in \{1,2,4,8,10,11,16,20,29,30,32,40,41,48,63\}$, with 2,000 examples per depth. The separate Rule 110 long-depth benchmark uses $n = 128$, $D \in \{64,80,96,128\}$, and 200 examples per depth.

Split separation. All rows are generated programmatically. Training and final evaluation use independent random streams; although online samples are not explicitly deduplicated, the 2^{256} possible initial rows make accidental train–test overlap negligible ($< 10^{-60}$).

D.1.3 Sudoku-Extreme

Following HRM (Wang et al., 2025) and TRM (Jolicoeur-Martineau, 2025), we use the Sudoku-Extreme dataset[1], a unique-solution 9×9 benchmark encoded as row-major 81-token puzzle–solution pairs, with 0 denoting blanks. Our fixed seed-0 evaluation subset contains 2,000 unaugmented puzzles from the 422,786-puzzle test pool; each has 55.7 blanks and requires 21.9 `tdoku` solver backtracks on average, indicating substantial search. For training, we expand 1,000 base puzzles to 1,001,000 pairs by retaining each original and generating 1,000 random Sudoku-preserving variants through digit relabeling, valid row/column permutations, and optional transposition.

D.2 Training Details

To enable fair comparisons between MDM and LS-MDM and facilitate reproducibility, we describe the S_5 and cellular-automaton configurations below.

D.2.1 S_5 composition

Model and Training Configuration. For a fair comparison, all S_5 experiments use the same Transformer architecture with eight signed-relative-attention heads and a fixed 64-slot source layout. We also match the optimization setup across methods, using AdamW with global batch 256, learning rate 3×10^{-4}, weight decay 0.1, and EMA decay 0.999. To enable token revision, Revise-MDM additionally corrupts 10% of visible tokens and supervises their correction. MDM, Revise-MDM, and LS-MDM all use $T = n$ rounds during training and inference.

Global product experiment. MDM and Revise-MDM retain the standard corruption-based formulation: a random subset of answer slots is masked, and the model predicts the clean sequence x_0 directly from the corrupted sequence. At inference, both start from a fully masked answer. Standard MDM predicts x_0, decodes only the subset selected by the unmasking schedule, and feeds the partially decoded sequence into the next round. Revise-MDM additionally permits corrections

[1] `https://huggingface.co/datasets/sapientinc/sudoku-extreme`

to previously decoded tokens: each round rewrites all answer slots with the current x_0, remasks the lowest-confidence subset under the Top-K schedule, and refines those positions in subsequent rounds. LS-MDM uses a one-step soft cursor that points to g_t at round t, providing an explicit inductive bias for sequential composition. During training, it starts from fully masked answer slots and performs all recurrent updates in hidden space. To reduce backpropagation cost, earlier recurrent states are detached and gradients flow only through the final four rounds; only the final hidden state is decoded and supervised. At inference, LS-MDM again starts from fully masked answer slots, keeps all intermediate computation in hidden space, and decodes only after the final round. MDM and LS-MDM apply token-wise cross-entropy to the final decoded product, with $\mathcal{L}_{\text{global}} = \sum_{j=1}^{5} \text{CE}(\hat{y}_j, y_j)$.

Product-conditioned broadcast experiment. Because $z_i = g_i \circ p_n$, revealing any answer token z_i exposes the global product through $p_n = g_i^{-1} \circ z_i$, after which every answer token $z_j = g_j \circ p_n$ is determined. To prevent this shortcut, both MDM and Revise-MDM replace random partial corruption with a fully masked answer canvas and predict the clean sequence x_0 directly. MDM, Revise-MDM, and LS-MDM all use the same one-step cursor, which marks g_t at round t and tells the shared backbone which permutation to incorporate next. This inductive bias encodes the known left-to-right composition order, ensuring that the comparison tests how intermediate products are represented and preserved rather than whether the model can discover a traversal policy. All three methods receive the same two supervision signals, a global-product loss and a product-conditioned sequence loss, with $\mathcal{L}_{\text{broadcast}} = \text{CE}(\hat{p}_n, p_n) + \sum_{i=1}^{n} \text{CE}(\hat{z}_i, z_i)$. The explicit p_n loss suppresses shortcut solutions that fit the output tokens without learning the underlying state transition.

Prefix-product scan experiment. The target prefixes satisfy $p_0 = e$ and $p_i = g_i \circ p_{i-1}$, so each position depends on its predecessor and the natural decoding order is left to right. We mirror this autoregressive factorization during training by corrupting from right to left: a suffix is masked while the correct prefix remains visible, and the model predicts the next frontier. We do not train Revise-MDM in this setting. Revising an earlier prefix p_i would change the conditioning state for every downstream token p_j, $j > i$, requiring the entire suffix to be recomputed; selective token revision is therefore incompatible with the left-to-right decoding process. LS-MDM follows the same left-to-right structure in hidden space: at recurrent step i, completed positions $j < i$ are frozen while the remaining positions continue to update. It decodes only after the final round and applies cross-entropy to the complete prefix chain, with $\mathcal{L}_{\text{prefix}} = \sum_{i=1}^{n} \text{CE}(\hat{p}_i, p_i)$.

D.2.2 Cellular-Automaton Experiments

Model and Training Configuration. For a controlled comparison, Rule 90 and Rule 110 use the same 9,198-parameter backbone: a single weight-tied Transformer block with hidden width 32, four attention heads, a 64-dimensional MLP, periodic left/center/right ring-local attention, and four Fourier frequencies for the update depth. All methods use AdamW with learning rate 10^{-3}, weight decay 10^{-4}, betas $(0.9, 0.99)$, batch size 64, gradient clipping at 1.0, and bf16 precision. For a fair comparison, MDM, Revise-MDM, and LS-MDM all run $T = D$ recurrent rounds for an instance of depth D.

Rule 90. Both MDM and LS-MDM use left/center/right attention on a ring, matching the periodic boundary condition under which the first and last cells are adjacent. To measure MDM's computational ceiling without requiring it to learn Rule 90's analytic dependency pattern, we additionally evaluate a hard Lucas router. Lucas' theorem gives $x_i^{(D)} = \bigoplus_{k \subseteq_{\text{bit}} D} x_{i-D+2k}^{(0)}$, where $k \subseteq_{\text{bit}} D$ denotes a bitwise submask; the router assigns its eight heads to the corresponding circular offsets $-D + 2k$. To strengthen early-round prediction, both MDM variants use fully masked inputs for 25% of training examples and random mask corruption for the remaining 75%; at inference, they always start from an all-mask canvas. Because LS-MDM has sufficient recurrent computational capacity to learn the transition rule directly, we do not provide it with the Lucas router or any other Rule-90-specific prior. LS-MDM starts from an all-mask state during both training and inference.

Rule 110. Rule 110 uses the same MDM and LS-MDM settings as Rule 90. Because no analogous Lucas prior is available, the Rule 110 MDM uses no Lucas routing and relies solely on learned ring-local attention. As an additional baseline, we evaluate external language-model APIs using a frozen, tool-free prompt that specifies the depth, cyclic width, Rule 110 lookup table, and initial row, and

requests only the final contiguous 128-bit state. All requests use `reasoning_effort=high`, while temperature, sampling seed, and maximum output tokens remain unspecified.

D.2.3 SUDOKU-EXTREME

Model and Training Configuration. For the matched-capacity comparison, MDM and LS-MDM use the same two-layer, width-512, eight-head non-causal GPT-2 backbone with zero dropout and 6.6M parameters. Both use AdamW with global batch 256, weight decay 1.0, a 2,000-step warmup, bf16 precision, and EMA decay 0.999; MDM uses learning rate 10^{-3}, whereas LS-MDM uses 10^{-4}. To test whether scale closes the gap, we additionally train MDM backbones with 38.1M, 85.5M, 113.8M, 151.7M, and 302.9M parameters, each using learning rate 3×10^{-4}.

Training Protocol. MDM trains with a 20-step masked-diffusion objective, predicting the clean grid from answer cells masked at a randomly sampled noise level. LS-MDM initializes the answer canvas as fully masked and refines a continuous latent state without intermediate decoding. Both use token-wise cross-entropy: MDM supervises only masked cells, whereas LS-MDM supervises all answer cells at each segment boundary.

D.2.4 VOCABULARY-STATE ABLATION

To isolate the effect of latent representation from corruption and denoising, all variants use the same recurrent architecture, optimization, curriculum, and task loss without diffusion corruption; they differ only in the state carried between rounds. LS-MDM retains an unrestricted continuous hidden state. Vocabulary-Simplex instead feeds back the convex combination $\sum_{v \in V} p_{i,v}^{(t)} e_v$ of token embeddings during both training and inference. Vocabulary-Vertex reuses the same Vocabulary-Simplex checkpoint but replaces this soft mixture at inference with the single embedding $e_{\arg\max_v p_{i,v}^{(t)}}$, restricting the recurrent state to a vocabulary vertex without retraining.

E EXTENDED EXPERIMENTAL RESULTS

Let $[5] = \{1, 2, 3, 4, 5\}$ and let S_5 be the group of bijections from $[5]$ to itself. An instance of length n is a sequence

$$(g_1, \ldots, g_n) \in S_5^n.$$

We use the input-order convention. Define $p_0 = e$ and

$$p_i = g_i \circ p_{i-1} \qquad \text{for } i = 1, \ldots, n.$$

Equivalently, for every $j \in [5]$,

$$p_n(j) = g_n\big(g_{n-1}(\cdots g_2(g_1(j)) \cdots)\big).$$

E.1 SHORT-OUTPUT S_5 COMPOSITION

The target is the one-line representation

$$y = (p_n(1), p_n(2), p_n(3), p_n(4), p_n(5)) \in [5]^5.$$

Each coordinate of y is emitted as one token, so the output length is $m = 5 = O(1)$, independent of n. Exact match is one if and only if all five output tokens equal this target in order. Table 4 reports LS-MDM performance across all evaluation lengths.

Each final evaluation uses 2,000 examples at every length and a rollout budget of $T = n$. The metric is exact match on the complete five-token permutation. Across three seeds and eleven evaluation lengths, all 33 evaluations achieve 100% exact match.

E.2 PRODUCT-CONDITIONED BROADCAST S_5 COMPOSITION

The product-conditioned broadcast target is a sequence of n permutation tokens,

$$z = (z_1, \ldots, z_n) \in S_5^n, \qquad z_i = g_i \circ p_n.$$

Table 4: LS-MDM exact match (%) on S_5 permutation composition.

Setting	n	Regime / definition	Seed 0	Seed 1	Seed 2	Mean ± std
Training lengths	1–10	Lengths in the training split	—	—	—	—
LS-MDM training lengths	$\{1, 2, 4, 8, 10\}$	Lengths used by the staged curriculum	—	—	—	—
Final evaluation	4	In range	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	8	In range	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	10	In range	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	12	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	16	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	20	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	24	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	32	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	40	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	48	Length OOD	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	64	Length OOD	100.00	100.00	100.00	100.00 ± 0.00

Table 5: Product-conditioned broadcast S_5 composition results for LS-MDM.

Setting	n	Regime / definition	Exact match (%)				Token accuracy (%)			
			Seed 0	Seed 1	Seed 2	Mean ± std	Seed 0	Seed 1	Seed 2	Mean ± std
Training lengths	1–10	Lengths in the training split	—	—	—	—	—	—	—	—
LS-MDM training lengths	$\{1, 2, 4, 8, 10\}$	Lengths used by the staged curriculum	—	—	—	—	—	—	—	—
Final evaluation	4	In range	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	8	In range	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	10	In range	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	12	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	16	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	20	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	24	Length OOD	100.00	99.95	100.00	99.98 ± 0.02	100.00	99.99	100.00	100.00 ± 0.01
Final evaluation	32	Length OOD	100.00	99.80	100.00	99.93 ± 0.09	100.00	99.93	100.00	99.98 ± 0.03
Final evaluation	40	Length OOD	100.00	99.85	100.00	99.95 ± 0.07	100.00	99.98	100.00	99.99 ± 0.01
Final evaluation	48	Length OOD	99.95	99.40	99.95	99.77 ± 0.26	100.00	99.85	99.95	99.93 ± 0.06
Final evaluation	64	Length OOD	0.00	95.20	97.95	64.38 ± 45.54	81.37	99.64	99.96	93.66 ± 8.69

Thus output position i contains the full input-order product followed by g_i. Each element of S_5 is represented by one token, so the output length is $m = n$. Exact match requires all n permutation tokens to be correct, while token accuracy is the fraction of correct output positions.

One correct target token would reveal the global product through $p_n = g_i^{-1} \circ z_i$. We therefore mask every output position on every training example. This removes the shortcut of recovering p_n from a revealed answer and directly trains the model to solve from the fully masked state used at inference. Table 5 reports exact match and token accuracy across lengths.

Training runs for 30,000 steps, and we report the checkpoint with the highest validation exact match. The staged curriculum advances at steps 300, 1,500, 4,500, 10,500, and 16,500 for lengths 1, 2, 4, 8, and 10. Final evaluation uses 2,000 examples at each length and rollout budget $T = n$.

E.3 Prefix-product S_5 composition

The prefix-product target is

$$r = (p_1, \ldots, p_n) \in S_5^n,$$

where $p_i = g_i \circ p_{i-1}$ is the product of the first i input permutations under the input-order convention defined above. Each p_i is represented by one token, so the output length is $m = n$. This target admits an answer derivation chain: $p_1 = g_1$, and after p_{i-1} is visible, $p_i = g_i \circ p_{i-1}$ is obtained by one group multiplication. A monotonic MDM may therefore reveal the target from left to right in $T = n$ rounds. Tables 6 and 7 report the LS-MDM and monotonic-MDM results, respectively.

All three LS-MDM runs train for 30,000 steps, and the table reports the final checkpoint. The staged curriculum advances at steps 300, 1,500, 4,500, 10,500, and 16,500 for lengths 1, 2, 4, 8, and 10, while replaying examples from earlier stages.

All three MDM runs train for 60,000 steps, and the table reports the checkpoint with the highest validation exact match. The curriculum covers every integer length from 1 through 10. Each final evaluation for both models uses 2,000 examples per length and seed.

Table 6: Prefix-product S_5 composition results for LS-MDM.

Setting	n	Regime / definition	Exact match (%)				Token accuracy (%)			
			Seed 0	Seed 1	Seed 2	Mean ± std	Seed 0	Seed 1	Seed 2	Mean ± std
Training lengths	1–10	Lengths in the training split	—	—	—	—	—	—	—	—
LS-MDM training lengths	$\{1, 2, 4, 8, 10\}$	Lengths used by the staged curriculum	—	—	—	—	—	—	—	—
Final evaluation	4	In range	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	8	In range	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	10	In range	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	12	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	16	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	20	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	24	Length OOD	100.00	100.00	99.90	99.97 ± 0.05	100.00	100.00	99.98	99.99 ± 0.01
Final evaluation	32	Length OOD	100.00	100.00	100.00	100.00 ± 0.00	100.00	100.00	100.00	100.00 ± 0.00
Final evaluation	40	Length OOD	100.00	99.95	99.80	99.92 ± 0.08	100.00	99.98	99.96	99.98 ± 0.02
Final evaluation	48	Length OOD	100.00	100.00	99.50	99.83 ± 0.24	100.00	100.00	99.83	99.94 ± 0.08
Final evaluation	64	Length OOD	100.00	99.90	99.20	99.70 ± 0.36	100.00	100.00	99.70	99.90 ± 0.14

Table 7: Prefix-product S_5 composition results for $\mathrm{MDM}_{\mathrm{mono}}$.

Setting	n	Regime / definition	Exact match (%)				Token accuracy (%)			
			Seed 0	Seed 1	Seed 2	Mean ± std	Seed 0	Seed 1	Seed 2	Mean ± std
Training lengths	1–10	Lengths in the training split	—	—	—	—	—	—	—	—
MDM training lengths	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$	All integer training lengths	—	—	—	—	—	—	—	—
Final evaluation	4	In range	100.00	99.95	100.00	99.98 ± 0.02	100.00	99.98	100.00	99.99 ± 0.01
Final evaluation	8	In range	100.00	99.85	100.00	99.95 ± 0.07	100.00	99.94	100.00	99.98 ± 0.03
Final evaluation	10	In range	100.00	99.85	100.00	99.95 ± 0.07	100.00	99.88	100.00	99.96 ± 0.06
Final evaluation	12	Length OOD	100.00	99.90	100.00	99.97 ± 0.05	100.00	99.93	100.00	99.98 ± 0.03
Final evaluation	16	Length OOD	100.00	99.80	100.00	99.93 ± 0.09	100.00	99.92	100.00	99.97 ± 0.04
Final evaluation	20	Length OOD	100.00	99.80	100.00	99.93 ± 0.09	100.00	99.97	100.00	99.99 ± 0.02
Final evaluation	24	Length OOD	100.00	99.45	100.00	99.82 ± 0.26	100.00	99.70	100.00	99.90 ± 0.14
Final evaluation	32	Length OOD	100.00	99.55	100.00	99.85 ± 0.21	100.00	99.71	100.00	99.90 ± 0.14
Final evaluation	40	Length OOD	100.00	99.40	100.00	99.80 ± 0.28	100.00	99.66	100.00	99.89 ± 0.16
Final evaluation	48	Length OOD	100.00	99.60	100.00	99.87 ± 0.19	100.00	99.79	100.00	99.93 ± 0.10
Final evaluation	64	Length OOD	100.00	99.20	100.00	99.73 ± 0.38	100.00	99.66	100.00	99.89 ± 0.16

E.4 Rule 90

Let the initial state be $x^{(0)} \in \{0,1\}^{256}$, with cell indices in the cyclic group $\mathbb{Z}_{256}$. Rule 90 evolves the state by

$$x_i^{(t+1)} = x_{i-1}^{(t)} \oplus x_{i+1}^{(t)}, \qquad i \in \mathbb{Z}_{256}.$$

An instance consists of $x^{(0)}$ and a rollout depth D; the target is the complete state $x^{(D)} \in \{0,1\}^{256}$. Token accuracy is the fraction of the 256 output bits predicted correctly.

Expanding the recurrence over $\mathbb{F}_2$ gives

$$x_i^{(D)} = \bigoplus_{\substack{0 \le k \le D \\ \binom{D}{k} \bmod 2 = 1}} x_{i+D-2k}^{(0)},$$

where indices are taken modulo 256. By Lucas's theorem, the number of odd coefficients in row D of Pascal's triangle is

$$2^{\mathrm{popcount}(D)},$$

where $\mathrm{popcount}(D)$ is the number of ones in the binary representation of D. All evaluated depths satisfy $D \le 63$, so the offsets $D - 2k$ are distinct modulo 256. Thus every output bit is the XOR of exactly $2^{\mathrm{popcount}(D)}$ distinct input bits and requires $2^{\mathrm{popcount}(D)} - 1$ binary XOR operations. This range also precedes the finite-ring collapse at $D = 128$, after which Rule 90 on a ring of size 256 reaches the all-zero fixed point. Table 8 reports the corresponding three-seed accuracies and dependency sizes.

E.5 Rule 110

Let $x^{(0)} \in \{0,1\}^n$ be a binary row with indices in $\mathbb{Z}_n$. Rule 110 applies the synchronous update

$$x_i^{(t+1)} = \phi\left(x_{i-1}^{(t)}, x_i^{(t)}, x_{i+1}^{(t)}\right), \qquad i \in \mathbb{Z}_n,$$

where the local rule is

Table 8: Rule 90 token accuracy at $n = 256$.

Depth D	MDM token accuracy (%)				LS-MDM token accuracy (%)				XOR terms
	Seed 0	Seed 1	Seed 2	Mean ± std	Seed 0	Seed 1	Seed 2	Mean ± std	
3	99.9953	99.9994	99.9146	99.9698 ± 0.0390	100.0000	100.0000	100.0000	100.0000 ± 0.0000	4
30	49.9623	50.0174	49.9789	49.9862 ± 0.0231	100.0000	100.0000	100.0000	100.0000 ± 0.0000	16
31	50.0445	50.0451	50.0943	50.0613 ± 0.0233	100.0000	100.0000	100.0000	100.0000 ± 0.0000	32
32	100.0000	100.0000	91.0543	97.0181 ± 4.2170	100.0000	100.0000	99.9992	99.9997 ± 0.0004	2
33	100.0000	100.0000	98.6357	99.5452 ± 0.6431	100.0000	100.0000	100.0000	100.0000 ± 0.0000	4
35	50.0795	50.0773	85.0711	61.7426 ± 16.4957	100.0000	100.0000	100.0000	100.0000 ± 0.0000	8
47	49.9984	49.9043	50.0607	49.9878 ± 0.0643	100.0000	100.0000	99.9984	99.9995 ± 0.0008	32
48	100.0000	100.0000	97.7486	99.2495 ± 1.0613	100.0000	100.0000	100.0000	100.0000 ± 0.0000	4
63	50.0424	50.0197	50.0588	50.0403 ± 0.0160	100.0000	100.0000	99.9941	99.9980 ± 0.0028	64

Table 9: Rule 110 standard evaluation at row length $n = 256$.

D	Split	Masked MDM (%)		Revise-MDM (%)		LS-MDM exact match (%)			
		EM	Cell acc.	EM	Cell acc.	Seed 0	Seed 1	Seed 2	Mean ± std
1	Train	100.00	100.00	100.00	100.00	100	100	100	100.00 ± 0.00
2	Train	0.00	66.94	0.00	78.46	100	100	100	100.00 ± 0.00
4	Train	0.00	54.60	0.00	51.84	100	100	100	100.00 ± 0.00
8	Train	0.00	53.14	0.00	54.02	100	100	100	100.00 ± 0.00
10	Train	0.00	54.21	0.00	54.48	100	100	100	100.00 ± 0.00
11	Unseen	0.00	54.34	0.00	54.99	100	100	100	100.00 ± 0.00
16	Unseen	0.00	52.62	0.00	55.66	100	100	100	100.00 ± 0.00
20	Unseen	0.00	54.03	0.00	55.77	99.95	100	100	99.98 ± 0.02
29	Unseen	0.00	56.11	0.00	56.11	100	100	100	100.00 ± 0.00
30	Train	0.00	55.99	0.00	55.99	100	100	100	100.00 ± 0.00
32	Train	0.00	56.15	0.00	56.15	100	100	100	100.00 ± 0.00
40	Train	0.00	56.37	0.00	56.37	100	100	100	100.00 ± 0.00
41	Depth OOD	0.00	56.24	0.00	56.24	100	100	100	100.00 ± 0.00
48	Depth OOD	0.00	56.36	0.00	56.36	100	100	100	100.00 ± 0.00
63	Depth OOD	0.00	53.29	0.00	56.43	100	100	100	100.00 ± 0.00

Input neighborhood (a, b, c)	**111**	**110**	**101**	**100**	**011**	**010**	**001**	**000**
Rule 110 output $\phi(a, b, c)$	0	1	1	0	1	1	1	0

An instance consists of the initial row $x^{(0)}$ and rollout depth D; the target is the complete row $x^{(D)}$. Exact match requires all n target bits to be correct, and cell accuracy is their mean bit accuracy.

Every evaluated instance is non-cyclic over its requested horizon:

$$x^{(s)} \neq x^{(t)} \qquad \text{for all } 0 \leq s < t \leq D.$$

This condition prevents a model from solving the task by detecting a repeated row and skipping the intervening updates.

Standard evaluation. The standard protocol uses $n = 256$ and 2,000 examples at every depth. LS-MDM is trained at depths $D \in \{1, \ldots, 10\} \cup \{30, \ldots, 40\}$; depths $11, 16, 20, 29$ are unseen interpolation points, and depths above 40 are depth OOD. LS-MDM means and standard deviations use the population standard deviation over three runs, labeled seeds $0, 1, 2$. Masked MDM and Revise-MDM use seed 0 only, so their columns report single-checkpoint results without a seed standard deviation. Table 9 reports the standard-depth results.

Long-depth evaluation. The long-depth protocol uses $n = 128$, depths $D \in \{64, 80, 96, 128\}$, and 200 shared non-cyclic examples per depth. This protocol is distinct from the standard $n = 256$ evaluation and is not pooled with it. Table 10 reports this long-depth comparison.

Table 10: Rule 110 long-depth evaluation.

D	n	Samples	Masked MDM (%)		Revise-MDM (%)		LS-MDM (%)					Majority baseline (%)
			EM	Cell acc.	EM	Cell acc.	Seed 0	Seed 1	Seed 2	EM mean ± std	Cell acc. mean ± std	
64	128	200	0.00	53.36	0.00	56.54	100	100	100	100.00 ± 0.00	100.00 ± 0.00	56.54
80	128	200	0.00	55.61	0.00	54.95	100	100	100	100.00 ± 0.00	100.00 ± 0.00	56.45
96	128	200	0.00	54.53	0.00	48.36	100	100	100	100.00 ± 0.00	100.00 ± 0.00	56.72
128	128	200	0.00	54.14	0.00	53.14	100	0	100	66.67 ± 47.14	83.99 ± 22.64	56.56

Table 11: Prompted language-model exact match (EM) and token accuracy (%) on Rule 110. Each model is evaluated on 10 examples per depth; Invalid counts failed responses across all 40 calls.

Model	Samples	$D = 64$		$D = 80$		$D = 96$		$D = 128$		Invalid
		EM	Token acc.	EM	Token acc.	EM	Token acc.	EM	Token acc.	
GPT-6 Astra	10	100.0	100.0	70.0	92.3	60.0	89.5	50.0	80.5	0
GPT-5.6 Sol	10	30.0	73.4	0.0	62.3	0.0	51.1	0.0	49.8	0
GPT-5.6 Luna	10	0.0	24.2	0.0	26.0	0.0	30.7	0.0	34.6	17
DeepSeek V4.1 Flash	10	20.0	64.5	0.0	45.9	0.0	49.3	0.0	46.7	2
Llama 4 Maverick	10	0.0	52.3	0.0	50.5	0.0	46.6	0.0	48.7	0

Prompted language models. Each language model receives the same 10 non-cyclic instances per depth, drawn from the 200-example LS-MDM evaluation set, and is asked to solve them with long chain-of-thought. Exact match is the percentage of correct instances out of 10, while token accuracy is averaged over the ten target rows. All calls set the reasoning effort to high. Invalid responses receive zero exact match and zero token accuracy; the Invalid column reports their total across all 40 calls. Only models with all 40 responses scored are included. Table 11 reports the prompted-model results.

E.6 SUDOKU-EXTREME

Each instance is a partially filled 9×9 Sudoku grid with a unique solution. The model receives the given digits and predicts the complete grid. A prediction receives whole-puzzle exact match if and only if every one of the 81 cells equals the unique solution. We evaluate all configurations on the same 2,000 puzzles.

To measure where deterministic answer propagation stalls, let P be a puzzle and S its complete reference solution. Let $\mathrm{R2}(B)$ denote running our deterministic propagation solver from board B until it either completes the puzzle or reaches a fixed point. Define

$$B_0 = \mathrm{R2}(P),$$

and let $U(B_0)$ be the set of cells left empty in B_0. For $C \subseteq U(B_0)$, let $B_0 \oplus S_C$ be the board obtained by filling each cell in C with its value in S. The minimum correct-guess count is

$$g^*(P, S) = \min_{C \subseteq U(B_0)} \{|C| : \mathrm{R2}(B_0 \oplus S_C) \text{ completes the puzzle}\}.$$

A guess therefore means revealing one correct reference-solution digit. It does not count incorrect attempts, backtracks, or search-tree nodes.

We exhaustively test the empty set, every singleton, and every unordered pair of cells. The buckets $g = 0$, $g = 1$, and $g = 2$ therefore identify $g^* = 0$, 1, and 2 exactly. The bucket $g = 3+$ certifies only $g^* \geq 3$. For this final bucket, a deterministic greedy oracle records one continuation path, which need not be shortest. If the successive guesses on this path have candidate counts $c_1, \ldots, c_k$, its luck envelope is

$$\prod_{j=1}^{k} \frac{1}{c_j}.$$

The evaluated set contains 279, 290, and 1,431 puzzles in the exact $g = 0$, 1, and 2 buckets, respectively. Figure 4**b** summarizes per-puzzle token accuracy using all puzzles; displayed dots are a

Table 12: Whole-puzzle exact match (%) on 2,000 Sudoku-Extreme puzzles.

Model	Architecture			Random Monotonic	Det. linear revision		Gumbel-linear revision		Det. cosine revision
	Params	Layers	Width		Prob.	Margin	Prob.	Margin	Prob.
MDM	6.6M	2	512	1.65	**8.95**	8.50	2.70	2.85	6.00
MDM	38.1M	12	512	13.45	53.15	**53.50**	27.60	30.00	49.65
MDM	85.5M	12	768	16.55	**53.20**	53.15	29.10	29.55	50.90
MDM	113.8M	36	512	23.30	**56.75**	56.10	36.30	38.30	54.70
MDM	151.7M	12	1024	14.10	52.35	**53.10**	24.55	26.55	49.90
MDM	302.9M	24	1024	18.15	**55.40**	54.70	29.45	31.45	52.95
LS-MDM	6.6M	2	512	**73.95**					

Note. Random monotonic decoding reveals positions stochastically and freezes revealed tokens. Prob. ranks positions by top-1 probability; Margin ranks them by the top-1–top-2 probability gap.

deterministic subsample of at most 100 puzzles per bucket. The plot reports descriptive distributions only, without inferential error bars or hypothesis tests.

For the controlled architectural comparison, LS-MDM and the smallest MDM use the same two-layer Transformer backbone with width 512 and 6.6M parameters. LS-MDM carries a persistent hidden state across recurrences and uses a puzzle-dependent recurrence depth, allowing different instances to receive different amounts of computation. The remaining MDM baselines scale depth and width from 6.6M to 302.9M parameters.

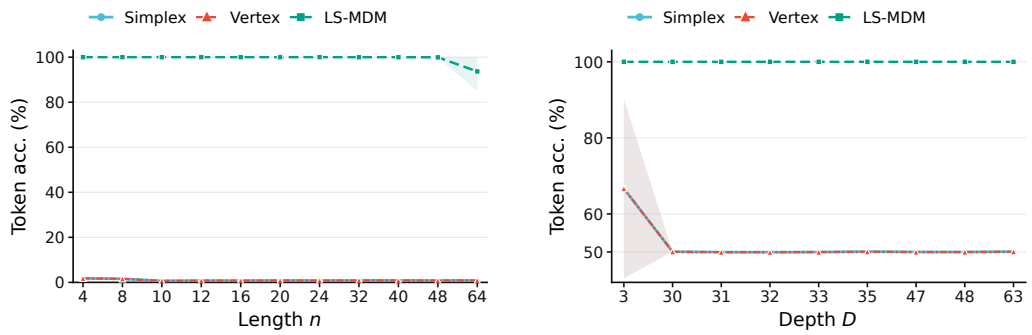
For each MDM, we evaluate random monotonic decoding and five confidence-based remasking-and-revision strategies. Random monotonic decoding reveals positions stochastically and freezes each revealed token. In the other variants, token values are greedy argmax predictions, low-confidence positions are remasked, and all target positions remain revisable at later steps. Deterministic linear revision ranks positions by either predicted probability or logit margin. Gumbel-linear revision perturbs those position scores with Gumbel noise under the same linear mask schedule; it does not sample token values. Deterministic cosine revision uses predicted probability with a cosine mask schedule. LS-MDM reads out the completed grid after its recurrent computation, so the MDM planner variants do not apply. Table 12 reports all whole-puzzle exact-match results.

E.7 Latent-state representation ablation

To isolate the effect of latent state from diffusion's corruption-and-denoising paradigm, all variants use the same recurrent framework without diffusion corruption and differ only in the representation carried between rounds. Let $e_v \in \mathbb{R}^d$ be token v's embedding, let $r_i^{(t)} \in \mathbb{R}^d$ be the pre-readout latent representation at position i in round t, and define $p_i^{(t)} = \mathrm{softmax}(W_{\mathrm{out}} r_i^{(t)} + b_{\mathrm{out}})$. The next-round latent representation is:

Variant	Training representation	Inference representation
Vocabulary-Simplex	$\sum_{v \in V} p_{i,v}^{(t)} e_v$	$\sum_{v \in V} p_{i,v}^{(t)} e_v$
Vocabulary-Vertex	$\sum_{v \in V} p_{i,v}^{(t)} e_v$	$\sum_{v \in V} \delta_{v, \arg\max p_i^{(t)}} e_v$
LS-MDM	$r_i^{(t)}$	$r_i^{(t)}$

Figure 5 shows that the vocabulary-constrained variants approach chance performance as the S_5 composition length or Rule 90 depth increases, whereas LS-MDM remains near-perfect. On Sudoku-Extreme, Table 13 shows that Vocabulary-Simplex and Vocabulary-Vertex reach only 4.70% and 0.15% exact match, compared with 73.95% for LS-MDM. These results show that recurrence alone is insufficient on more demanding reasoning tasks when intermediate computation must be projected back into the vocabulary. Effective cross-round reasoning requires a state space that can preserve partial computations before they correspond to output tokens.



(a) S_5 product-conditioned broadcast.

(b) Rule 90.

Figure 5: **Continuous latent state outperforms vocabulary-constrained feedback.** All variants share the same recurrent backbone and training setup, differing only in the state passed between rounds: expected token embeddings (Vocabulary-Simplex), argmax token embeddings at inference (Vocabulary-Vertex), or continuous hidden states (LS-MDM). **(a)** S_5 product-conditioned broadcast across sequence lengths. **(b)** Rule 90 across depths on a periodic 256-cell ring. Curves show three-seed mean token accuracy; shading denotes ± 1 population standard deviation.

Table 13: Latent-state representation ablation on Sudoku-Extreme. The metric is whole-puzzle exact-match accuracy (%) on 2,000 puzzles.

Latent representation	**Vocabulary-Simplex**	**Vocabulary-Vertex**	**LS-MDM**
EM (%)	4.70	0.15	73.95