The Confidence Shortcut: A Reasoning Failure Mode of Masked Diffusion Models

Dueun Kim [1] **Albert No** [1]

Abstract

Masked diffusion language models (MDMs) uniquely support any-order generation, with confidence-based decoding currently serving as the de facto standard inference policy. To optimize for this, recent training schemes attempt to align training mask patterns directly with those observed during generation. However, we argue that confidence-based decoding is inherently misaligned with the logical-flow trajectories required for complex reasoning, and that confidence-aligned training actively entrenches this misalignment. We make this concrete using multi-digit addition, where the decoding strategy prematurely predicts locally easy digits before resolving their long-range dependencies, producing high-confidence errors on challenging inputs. While traditional random masking keeps the failure rate low on this challenging tail, confidence-aligned training amplifies the error rate by an order of magnitude. Across five distinct reasoning tasks, this same pattern emerges with task-dependent severity: confidence-based decoding induces failures on highly complex inputs, and confidence-aligned training exacerbates them. In contrast, random masking—despite its perceived inefficiency—robustly preserves the reasoning-trajectory conditionals essential for solving the challenging tail.

1. Introduction

Discrete diffusion models (Austin et al., 2021; Lou et al., 2023; Campbell et al., 2022) have emerged as an alternative to autoregressive language modeling. Among discrete diffusion variants, masked diffusion models (MDMs) (Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2024) are particularly prominent: tokens transition between an absorbing `[MASK]` state and the original text, and the model is trained to reconstruct masked positions from a partially observed context. Recent work has shown that MDMs can scale to large-scale language modeling (Nie et al., 2025; Ye et al., 2025b; Gong et al., 2025) and achieve competitive performance on reasoning, planning, and code-generation benchmarks (Ye et al., 2025a; Zhao et al., 2025).

A central appeal of MDMs is their decoding flexibility. Unlike autoregressive models, which commit to a fixed left-to-right factorization, an MDM can support arbitrary generation orders: different positions can be unmasked at different times, and each decoding order induces a different factorization of the same sequence distribution (Chang et al., 2022; Kim et al., 2025; Ye et al., 2025b). The dominant inference heuristic is *confidence-based decoding*, which reveals the position with the largest top-1 probability, margin, or negative entropy score. This heuristic is intuitively appealing for generation, as it first commits to tokens that appear easiest under the current context. It has also motivated recent confidence-aligned training schemes. PAPL (Peng et al., 2026) reweights the per-token loss toward positions the model already predicts confidently, while PUMA (Kim et al., 2026b) modifies the masking process so that training states resemble confidence-based inference trajectories.

For reasoning tasks, however, generation order is not merely a matter of presentation. A reasoning problem usually has a *logical-flow order*: an order in which intermediate facts become justified and later facts become determinate. In multi-digit addition, the stable order is least-significant-digit first, because each carry must be propagated from lower digits before higher digits are fully determined. More generally, the useful generation order is the order in which a competent solver would establish the solution. An MDM decoded in a different order must predict later facts while their prerequisites are still masked, forcing the model to marginalize over unresolved reasoning states. Confidence-based decoding can therefore diverge from the reasoning order: it prefers locally easy tokens, not necessarily tokens whose dependencies have been resolved.

This distinction matters most on the hard tail of reasoning distributions. The inputs on which the dependency order is long or rigid are often rare, but they are not pathological

[1]Department of Artificial Intelligence, Yonsei University, Seoul, Korea. Correspondence to: Albert No <albertno@yonsei.ac.kr>.

Published as a paper at the 1st FoGen workshop, ICML 2026, Seoul, South Korea, 2026.

outliers. They are controlled versions of the cases for which reasoning models are most valuable: long carry chains, narrow maze corridors, deeply nested expressions, or ultimately hard mathematical and scientific problems whose dependency structure cannot be shortened by local heuristics. A model that performs well on common instances by following confidence shortcuts may still fail on precisely the inputs that distinguish complex reasoning from mere interpolation.

We use multi-digit addition as the cleanest setting in which to expose this divergence. Addition is simple enough that the correct dependency structure is known exactly: each digit's value depends on the carry propagated from below, so the unique reasoning order is least-significant-digit first. At the same time, it admits a strong distributional shortcut. Under typical digit sampling, long carry chains are rare, and high-order digits can usually be predicted from a short local window without traversing the full chain. These two ingredients—a known reasoning order and an available shortcut effective on most inputs—allow us to directly measure the decoding behavior. Specifically, we can determine whether confidence-based decoding follows the logical reasoning order or the shortcut, and observe how a model behaves when these two pathways diverge.

Our empirical study compares uniform random masking with two confidence-aligned training schemes, PAPL and PUMA, on five reasoning tasks: addition, maze, ListOps, Countdown, and Sudoku. We hold architecture and compute fixed within each domain; only the training intervention varies. We evaluate confidence-based decoding, random decoding, and task-specific logical-flow or solver-derived orders where available, and stratify results by structural difficulty. The results show that confidence alignment can amplify the mismatch between locally easy predictions and the true reasoning-order dependencies. Depending on the task, this alignment can even produce critical overall failure.

Our contributions are:

- We formulate a reasoning-order view of MDM decoding and explain why confidence-based decoding can be suboptimal when confidence differs from logical-flow dependency order.
- We give a concrete analysis of the confidence shortcut on multi-digit addition, characterizing how confidence-aligned training schemes amplify the failure.
- We provide a controlled five-task empirical study showing that confidence-aligned training can amplify reasoning-order coverage gaps in qualitatively different ways across tasks.

2. Preliminaries

Notation. Let $\mathbf{x} = (x_1, \ldots, x_L) \in \mathcal{V}^L$ denote a clean sequence over vocabulary $\mathcal{V}$, augmented with a special mask token `M`. For a subset $\mathbf{M} \subseteq [L] = \{1, \ldots, L\}$, we write $\overline{\mathbf{M}} = [L] \setminus \mathbf{M}$ for the complement and $\mathbf{x}_{\overline{\mathbf{M}}}$ for the subsequence of $\mathbf{x}$ at the positions in $\overline{\mathbf{M}}$. We refer to $\mathbf{M}$ as the masked indices and $\overline{\mathbf{M}}$ as the visible indices.

2.1. Masked Diffusion Models

A masked diffusion model (MDM) learns to reconstruct tokens from partially masked contexts. Given $\mathbf{x} \sim p_{\text{data}}$, a masking rate $\lambda \sim \mathrm{Unif}(0, 1)$ is sampled, and each position in $[L]$ is independently masked with probability λ. Let $\mathbf{M} \subseteq [L]$ be the resulting masked set. The model observes $\mathbf{x}_{\overline{\mathbf{M}}}$ and assigns a categorical distribution to each masked position:

$$p_\theta(x_i \mid \mathbf{x}_{\overline{\mathbf{M}}}) \in \Delta(\mathcal{V}), \quad i \in \mathbf{M}.$$

The standard MDM denoising objective is

$$\mathcal{L}(\theta) = -\,\mathbb{E}_{\mathbf{x},\,\lambda,\,\mathbf{M}} \left[\frac{1}{\lambda} \sum_{i \in \mathbf{M}} \log p_\theta(x_i \mid \mathbf{x}_{\overline{\mathbf{M}}}) \right],$$

where $1/\lambda$ normalizes for the expected fraction of masked tokens.

Order-agnostic interpretation. The MDM objective is equivalent to a uniform expectation over generation orders (Kim et al., 2025):

$$\mathcal{L}(\theta) \;\propto\; -\,\mathbb{E}_{\pi \sim \mathrm{Unif}(\mathbb{S}_L)} \left[\sum_{j=1}^{L} \log p_\theta\big(x_{\pi(j)} \,\big|\, \mathbf{x}_{\pi(:\,j)}\big) \right],$$

where $\mathbb{S}_L$ denotes the symmetric group of permutations π on $[L]$ and $\pi(:\,j) = \{\pi(1), \ldots, \pi(j-1)\}$ is the prefix unmasked before step j. Every generation order is trained uniformly; the order used at inference time is determined by the decoding policy.

2.2. Decoding policies

MDM inference starts from the fully masked sequence and iteratively unmasks tokens. At each step, let $\mathbf{M} \subseteq [L]$ be the current masked set; the model produces a distribution $p_\theta(\cdot \mid \mathbf{x}_{\overline{\mathbf{M}}})$ for each $i \in \mathbf{M}$. A decoding policy chooses a reveal set $R \subseteq \mathbf{M}$ and fills those positions, typically by greedy prediction:

$$x_i \;\leftarrow\; \arg\max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}_{\overline{\mathbf{M}}}), \quad i \in R.$$

Confidence-based decoding. The most common decoding policy selects positions where the current model is most

confident. For a masked position $i \in \mathbf{M}$, define the top-1 confidence

$$c_\theta^i = \max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}_{\overline{\mathbf{M}}}).$$

Confidence-based decoding selects the highest-scoring positions according to c_θ^i and decodes them first. Variants use related uncertainty measures, such as the margin between the top two probabilities or negative predictive entropy (Chang et al., 2022; Kim et al., 2025; Ye et al., 2025b).

Confidence-aligned training. Recent work modifies MDM training so that the training distribution better matches confidence-based inference. *PAPL* (Planner Aware Path Learning) (Peng et al., 2026) keeps the i.i.d. random masking process but reweights the per-token loss across masked positions: positions that the model already predicts confidently receive larger loss weight. *PUMA* (Progressive UnMAsking) (Kim et al., 2026b) replaces the masking process itself: starting from a fully masked sequence, it iteratively unmasks ground-truth tokens in the order selected by the model's confidence scores and uses the resulting intermediate states as denoising contexts. Both schemes shift the training distribution toward the inference-time confidence trajectory, on the rationale that aligning the two should improve generation quality. For the details of each method, please refer to Appendix C.

3. Addition as a Controlled Lens on Reasoning Order

Addition serves as an exceptionally clean diagnostic for reasoning in masked diffusion models. The task is elementary, its exact dependency structure is known, and the gap between genuine reasoning and high-accuracy shortcuts can be characterized in closed form. This makes addition an ideal motivating case before examining tasks with more complex and ambiguous dependencies.

3.1. Task setup

We consider 32-digit addition. Each example consists of two operands and a sum: "$a + b = c$."

The prompt contains the two operands and the equality sign, and the answer region contains the 33 output digits of the sum, including the possible carry-out digit. During generation, the answer region is initially masked and the model must fill in all output digits.

We evaluate three training schemes—standard uniform random masking, PAPL, and PUMA—across two decoding policies. The first is confidence-based decoding, which unmasks the highest-confidence digit at each step. The second is an addition-specific, least-significant-digit-first policy that strictly enforces the arithmetic dependency order. We stratify test instances by the length of their longest carry-propagation chain.

3.2. The optimal reasoning order is LSB-first

Although the input and output strings are conventionally written in most-significant-first order, the arithmetic computation flows in the opposite direction. Indexing from the least significant digit, let a_0, b_0, and c_0 be the least significant digits, with c_{32} representing the final carry-out. Let r_i denote the carry into position i. The arithmetic resolves as follows:

$$\begin{aligned} r_0 &= 0, \quad c_{32} = r_{32}, \\ c_i &= (a_i + b_i + r_i) \bmod 10, \\ r_{i+1} &= \mathbf{1}\{a_i + b_i + r_i \geq 10\}, \quad i = 0, \ldots, 31. \end{aligned} \tag{1}$$

Here, c_i is fully determined once r_i is known, and r_i is established only after resolving the lower-order positions. Therefore, unmasking the answer from c_0 to c_{32} never forces the model to predict a digit before its arithmetic prerequisites are available. This makes least-significant-digit-first (LSB-first) decoding the optimal logical-flow order for addition.

3.3. The tempting shortcut

Addition also admits a powerful shortcut hidden within these same mechanics. Define the position-wise sum $s_i = a_i + b_i$. If $s_i \leq 8$, the carry out of position i is *killed*: $r_{i+1} = 0$ regardless of r_i. If $s_i \geq 10$, a carry is *generated*: $r_{i+1} = 1$ regardless of r_i. Only when $s_i = 9$ does the position *propagate* the incoming carry, yielding $r_{i+1} = r_i$. We classify these as k, g, and p cells, respectively. A maximal sequence of consecutive p cells forms a *carry chain*, which is strictly bounded by the nearest lower-order g or k cell that terminates it.

To determine the carry into a high-order position, it is usually sufficient to look a short distance downward until a non-propagating digit (g or k) is encountered. True long-range dependency arises only across an unbroken run of $s_i = 9$. Under uniform digit sampling, $\mathbb{P}(s_i = 9) = 0.1$. Consequently, a bounded heuristic that looks only at the w preceding lower-order positions fails for a specific digit only if all w positions propagate the carry:

$$\begin{aligned} &\mathbb{P}\big(w\text{-digit lookahead insufficient at position } i\big) \\ &\quad = \mathbb{P}(s_{i-1} = \cdots = s_{i-w} = 9) = 10^{-w}. \end{aligned} \tag{2}$$

Applying a union bound over all 33 answer digits, the probability that any digit requires a lookahead greater than w is bounded by $33 \cdot 10^{-w}$. With just a window of $w = 8$, this probability falls below 10^{-6}. Thus, a model can achieve near-perfect average accuracy by learning a finite-window lookahead heuristic instead of the rigorous LSB-first computation. This shortcut is neither weak nor artificial; it almost perfectly aligns with the standard training distribution.

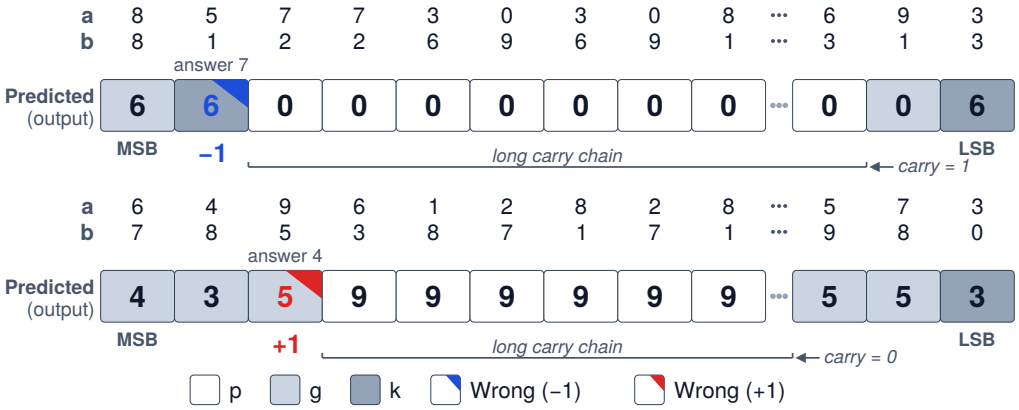

Figure 1. Two confidence-decode failures from random masking on long-carry-chain inputs. The chain-MSB cell is decoded before the chain interior is resolved. The committed value differs from the answer by exactly ± 1, with the direction determined by the cell's role: at k cells the model assumes no carry-in arrives (-1); at g cells the model assumes one does ($+1$).

Table 1. Exact-match accuracy on 32-digit addition by carry-chain length (3-seed average). "Random" denotes standard MDM training with uniform random masking. PAPL is reported with $\alpha = 1$; PAPL with $\alpha = 5$ collapses under both decode policies.

chain $\geq$	Random	PAPL$_{\alpha=1}$	PUMA	PAPL$_{\alpha=5}$
Confidence decode				
4	0.998	0.914	0.868	0.000
12	0.996	0.618	0.864	0.000
20	0.984	0.462	0.826	0.000
24	0.994	0.254	0.868	0.000
28	0.992	**0.026**	0.908	0.000
LSB-first decode				
4	1.000	0.942	1.000	0.000
12	1.000	0.724	1.000	0.000
20	1.000	0.580	1.000	0.000
24	1.000	0.308	1.000	0.000
28	1.000	**0.030**	1.000	0.006

This phenomenon demonstrates exactly why addition is such a useful diagnostic tool. Average test accuracy can appear nearly perfect, even while the model completely fails to implement the true reasoning order required for rare but structurally demanding carry chains.

3.4. Confidence decoding fails on the hard tail

Each model is trained for 300k iterations (batch size 256) on 20,000 random instances of 32-digit addition ($\approx 4{,}000$ epochs). Because this is an elementary task, all three training schemes easily achieve $\geq 99.9\%$ exact-match accuracy on the standard test set ($n = 10{,}000$) under confidence-based decoding. To probe behavior on the rare long-chain inputs, we construct test strata of 500 instances each, guaranteeing a minimum carry-chain length.

Table 1 reports exact-match accuracy by chain length. Random masking remains nearly perfect under confidence-based decoding—scoring 0.992 on the hardest stratum (chain ≥ 28)—and hits exactly 1.000 under LSB-first decoding across all lengths. PUMA, however, drops to 0.908 on the hardest stratum under confidence-based decoding, though it completely recovers to 1.000 when forced to use LSB-first decoding. PAPL suffers a far more severe collapse. Its loss reweighting is controlled by α (where larger values shift more weight to high-confidence positions). Even at a modest setting of $\alpha = 1$, accuracy drops to 0.026 under confidence-based decoding for chain ≥ 28, and LSB-first decoding barely improves this to 0.030. Under the default $\alpha = 5$, accuracy collapses to zero across all chain lengths regardless of the decoding policy.

Wrong commit profile. Of the 406 confidence-decoding failures (39 from random masking, 367 from PUMA), 99.8% occur at the chain-MSB cell. In these cases, the model misses the true answer by exactly ± 1, predictably reflecting the cell's arithmetic role: overshooting by $+1$ at nearly every g cell and undershooting by -1 at every k cell (Figure 1). Remarkably, the model is highly confident in these errors, assigning the wrong token a top-1 probability of 0.997 on average with the correct token consistently ranked second. Crucially, these wrong commits occur prematurely relative to the dependent chain. At the moment of the erroneous commit, the median fraction of the carry chain's interior that remains masked is 0.68 for random masking and 1.00 for PUMA. On the hardest stratum (chain ≥ 28), this commit happens as early as decoding step 12 of 33 for random masking, and at step 3 for PUMA. In short, confidence-based decoding selects the chain-MSB cell long before the carry chain is resolved, a flaw that PUMA massively exacerbates by committing while the entire chain is completely masked.

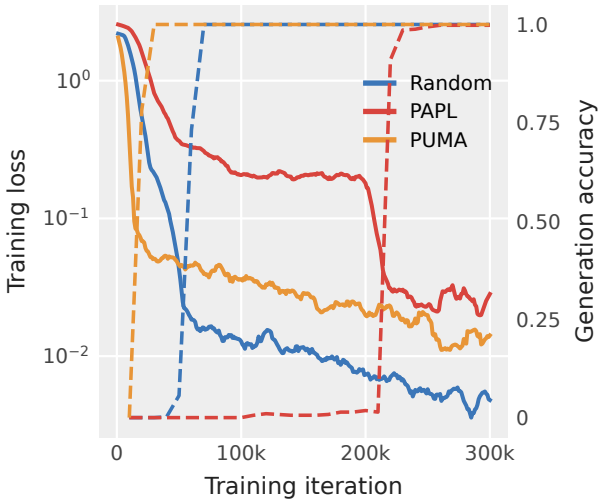

Figure 2. Training loss (solid, left axis) and generation accuracy (dashed, right axis) on the natural test distribution by training scheme.

PAPL exhibits a different failure mode. Its wrong commits are scattered across both g/k cells (exhibiting both over- and under-predictions without a consistent sign) and chain-interior p cells (missing by ± 9, indicative of a wrong-neighbor carry flip). Furthermore, these errors occur very late in the decoding trajectory—at a median stage of 28 out of 33, when only 28% of the chain remains unresolved—indicating that the failure arises after the chain has mostly been decoded.

3.5. Discussion: What addition teaches us

The chain-MSB cell is the worst possible target for confidence-based decoding. While local context ($a_d + b_d$) easily determines the carry-*out*, predicting the cell's exact digit still requires the unresolved carry-*in* from the chain below. Forced to predict prematurely, the model uses the cell's own arithmetic role as a local proxy. Since a k cell does not propagate a carry, the model assumes no carry arrives; conversely, for a g cell, it assumes one does. This substitution explains the predictable ± 1 errors. It is a typical *confidence shortcut*: a locally valid heuristic that succeeds on the bulk of the distribution but fails entirely when a distant anchor dictates the carry-in. Both random masking and PUMA share this exact failure profile—location, direction, and probability collapse—differing only in frequency.

Training dynamics (Figure 2) reveal the trade-off behind this: by aligning training mask states to the confidence-based inference trajectory, PUMA converges on the natural distribution an order of magnitude faster than random masking. However, this alignment rigidly commits the model to the shortcut trajectory, severely amplifying the latent failure rate on long-chain inputs. Importantly, the underlying capacity to compute the correct carry-in remains intact, as LSB-first oracle decoding perfectly recovers 100% accuracy. PUMA does not degrade what the model can represent, but rather biases which representations its inference path queries.

In contrast, PAPL exhibits a fundamentally different, unrecoverable failure mode. Its errors scatter unpredictably across cells and cannot be rescued by oracle decoding. This collapse highlights a fatal flaw in its loss reweighting: positions lacking early confidence (i.e., those with deep sequential dependencies) are continuously downweighted and effectively starved of training signal. PAPL's training loss flatlines for most of the process before rising late (Figure 2), reflecting that these deeply dependent representations are never properly learned, leaving the long-chain conditionals completely broken.

4. Experiments on Other Reasoning Tasks

4.1. Setup

We extend the analysis to four reasoning tasks: maze navigation, ListOps, Countdown, and Sudoku. For each task, all three schemes (random masking, PAPL, PUMA) are trained under a single architecture and compute budget per domain and evaluated under confidence-based decoding and a task-specific dependency-respecting decoding where available. Difficulty stratification is task-specific. For detailed data format and hyperparameter setting, please refer to Appendix B and C.

4.2. Maze

We evaluate a 10×10 maze rendered on a 21×21 wall/corridor grid, with marked start and goal. The model labels every corridor cell as on-path or off-path. The dependency-respecting oracle is dead-end-filling, the standard polynomial-time maze solver: it iteratively eliminates dead-end branches until only the start-to-goal path remains. We stratify the test mazes by the longest corridor in the backbone path; long corridors require information to propagate from both ends. Each stratum contains 300 mazes.

Table 2 reports puzzle-level exact-match accuracy by corridor length. Model trained with random masking is at or above 0.99 under both decode policies across all strata. PUMA's confidence-decode accuracy stays near 0.88 at every corridor length and is recovered to ≥ 0.91 by the dead-end filling, with the gap widening at longer corridors (0.893 confidence vs. 0.987 oracle at corridor ≥ 30). PAPL falls between the two: ~ 0.90 under confidence decoding and ≥ 0.97 under the oracle. Uniform random decoding performs much worse than either confidence or the oracle for all three schemes, with PUMA the weakest by a wide margin. PUMA's training distribution covers only mask states

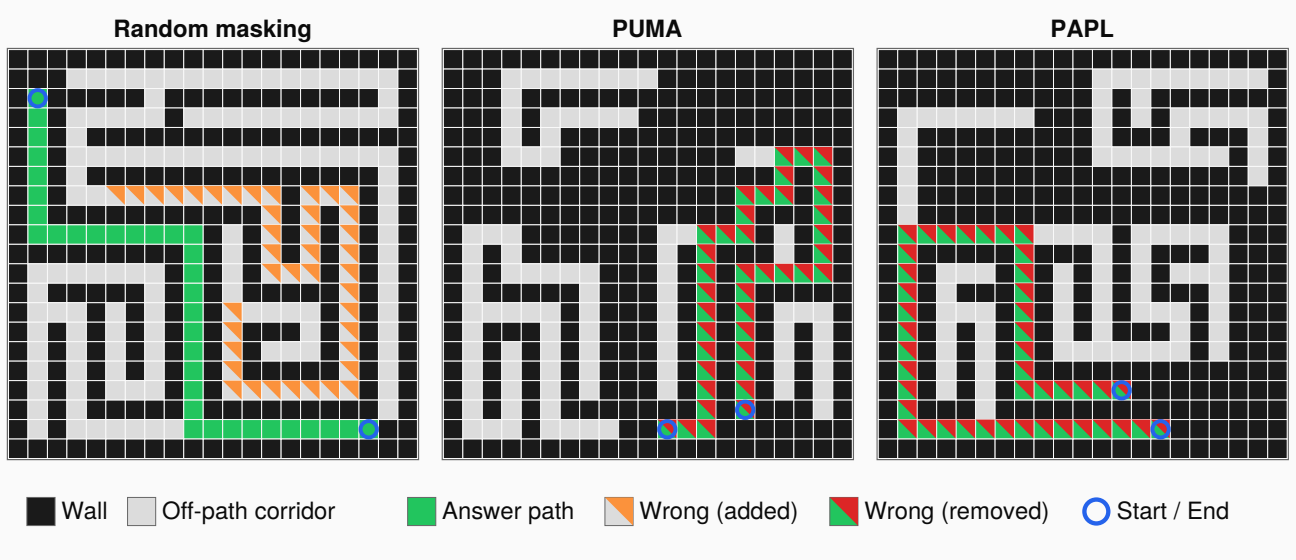

Figure 3. Wrong-commit region geometry on the maze grid. Cells are labeled by the color: walls (black), off-path corridors (light gray), the answer path (green) connecting two start/end cells (blue circles). Wrong commits are overlaid as a diagonal half: an off-path cell labeled on-path (orange) or an on-path cell labeled off-path (red).

Table 2. Maze puzzle exact-match accuracy by maximum corridor length on 21×21 grids.

corridor $\geq$	Random	PAPL	PUMA
Confidence decode			
4	0.997	0.903	0.873
8	0.983	0.903	0.883
15	0.990	0.880	0.883
20	0.987	0.893	0.877
25	1.000	0.930	0.897
30	0.993	0.907	0.893
Random decode			
4	0.773	0.723	0.417
8	0.723	0.727	0.410
15	0.610	0.617	0.363
20	0.540	0.517	0.250
25	0.430	0.547	0.140
30	0.517	0.517	0.130
Dead-end-filling decode			
4	0.993	0.980	0.920
8	0.983	0.977	0.913
15	0.997	0.973	0.913
20	1.000	0.997	0.987
25	1.000	0.993	0.990
30	1.000	1.000	0.987

reachable along its self-induced confidence trajectory, so the random decoding trajectory exposes the model to mask configurations it has rarely seen during training.

Unlike in the addition task, PAPL does not exhibit a complete collapse on mazes: oracle decoding recovers its performance to ≥ 0.97, indicating that the underlying conditionals distinguishing the labels remain intact. The maze grid imposes redundant constraints on each cell through its neighbors, so the same conditional is reachable through many partial-mask configurations, and a confidence-weighted loss does not strand any single dependency. The addition chain admits no such redundancy.

Wrong commit profile. We examined all 394 confidence-decoding failures (11 from random masking, 209 from PUMA, 174 from PAPL). Without exception, the wrong cells in each failing instance form a single contiguous, 1-cell-wide path rather than a 2D cluster. Topologically, 96% of wrong cells have exactly two wrong neighbors, and none have more than two. Local statistics confirm this spatial propagation: an orthogonal neighbor of a wrong cell is itself wrong with probability 0.45 on average, far exceeding the *i.i.d.* baseline of 0.12–0.15. The nature of the error differs across training schemes. PAPL and PUMA mostly make *path-removal errors*: they take cells that truly lie on the start-to-goal path and label them as off-path. PUMA makes this error in 202 of its 209 failures, and PAPL makes it in all 174 of its failures. Geometrically (red regions, Figure 3), the model cuts out a contiguous segment, severing the start-to-goal connection.

Random masking has far fewer failures. Among its 11 failures, 8 are instead *path-addition errors*: the model labels an off-path corridor as on-path. Geometrically, this creates a spurious side corridor or parallel false path rather than cutting the true path.

This shared shape arises because a cell's label is strongly biased toward extending its committed neighbors. Once the

Table 3. Accuracy on three reasoning tasks. "Recovery" is task-specific: layered post-order decoding for ListOps, 50% answer equation chain reveal for Countdown, and not available for Sudoku (omitted from the Recovery block). "m" for Countdown is solution multiplicity (number of distinct equations reaching the target). ListOps and Countdown report exact-match; Sudoku reports cell-level accuracy on blank cells.

Task	Stratum	Random	PAPL	PUMA
Confidence decode				
ListOps	depth $= 3$	0.426	0.302	0.206
ListOps	depth $= 5$	0.042	0.024	0.002
Countdown	$m \in [1, 3]$	0.182	0.173	0.211
Countdown	$m \geq 11$	0.973	0.981	0.331
Countdown	overall	0.518	0.525	0.294
Sudoku	overall	0.629	0.630	0.753
Sudoku	top 1%	0.357	0.372	0.609
Sudoku	TL4 ≥ 0.95	0.407	0.404	0.539
Random decode				
ListOps	depth $= 3$	0.354	0.282	0.164
ListOps	depth $= 5$	0.034	0.032	0.002
Countdown	$m \in [1, 3]$	0.129	0.134	0.092
Countdown	$m \geq 11$	0.975	0.970	0.474
Countdown	overall	0.488	0.494	0.265
Sudoku	overall	0.565	0.570	0.599
Sudoku	top 1%	0.370	0.365	0.432
Sudoku	TL4 ≥ 0.95	0.382	0.376	0.414
Recovery decode				
ListOps	depth $= 3$	0.406	0.322	0.174
ListOps	depth $= 5$	0.054	0.028	0.004
Countdown	$m \in [1, 3]$	0.995	0.993	0.985
Countdown	$m \geq 11$	1.000	1.000	0.999
Countdown	overall	0.997	0.998	0.992

model makes an incorrect commit, this flawed state acts as a confident local proxy for adjacent cells, cascading the error along the corridor until a dead end. This deeply parallels the chain-MSB substitution in addition: both exploit a locally consistent heuristic—carry symmetry there, corridor extension here—as a proxy for a globally determined value, propagating the error along the very structure that makes the shortcut tempting.

4.3. ListOps, Countdown, Sudoku

ListOps. ListOps consists of nested arithmetic expressions over MIN, MAX, MEDIAN, and SUM-MOD-10 operators (Nangia & Bowman, 2018). Its reasoning structure is hierarchical: inner sub-expressions must be evaluated before the outer operators become resolvable. We therefore stratify test instances by expression depth and use a layered post-order decoding policy as a bottom-up diagnostic order.

As depth increases, confidence-aligned training increasingly underperforms standard random masking. At depth 3, accuracy under confidence-based decoding is 0.426 for random masking, 0.302 for PAPL, and 0.206 for PUMA. At depth 5, the gap becomes even more pronounced: random masking reaches 0.042, PAPL 0.024, and PUMA only 0.002 (Table 3). Thus, the same pattern observed on long carry chains emerges in a hierarchical domain: as the dependency structure deepens, confidence-aligned training loses coverage faster than standard random masking.

Unlike in the addition and maze tasks, layered post-order decoding does not substantially recover model performance. At depth 5, random masking improves only from 0.042 to 0.054, PAPL from 0.024 to 0.028, and PUMA from 0.002 to 0.004. Uniform random decoding yields similar numbers (0.034, 0.032, and 0.002). These results suggest that the failure is not merely due to a bad confidence trajectory. In this small-model regime, the deep bottom-up conditionals themselves appear underlearned, especially for PUMA.

Countdown. Countdown asks the model to combine four numbers with $\{+, -, \times, /\}$ to reach a target. We stratify examples by solution multiplicity m, the number of distinct equation chains that reach the target; low m indicates a highly constrained instance, while high m indicates that many valid solutions exist (Katz et al., 2025).

The aggregate results reveal a surprising trend. PUMA is much weaker overall under confidence-based decoding: achieving 0.294 exact-match accuracy, compared with 0.518 for random masking and 0.525 for PAPL. However, this deficit is not concentrated on the rare, highly constrained cases. On the low-multiplicity stratum ($m \in [1, 3]$), PUMA slightly exceeds random masking (0.211 vs. 0.182), although this small gap should not be over-interpreted. The decisive failure occurs on the common, high-multiplicity stratum: for $m \geq 11$, PUMA drops to 0.331, while random masking and PAPL reach 0.973 and 0.981, respectively.

Uniform random decoding does not repair this gap: on $m \geq 11$, PUMA improves only to 0.474, remaining far below random masking and PAPL. In contrast, unmasking 50% of the answer equation chain recovers all three methods to at least 0.99 overall. Thus, PUMA has not lost the arithmetic conditionals needed to complete a solution once a useful scaffold is visible. Its failure is instead a mask-state coverage failure: the self-induced confidence training trajectory has narrowed away from the partial equation states needed to solve many common, easy Countdown instances from scratch.

Sudoku. We evaluate standard 9×9 Sudoku puzzles and report cell-level accuracy on initially blank cells. We stratify test instances by puzzle rating tier—where the top-1% denotes the hardest one percent of puzzles based on a solver's guess count—and by the fraction of TL4 cells, which require search beyond standard deductive techniques (t-dillon, 2019; Stuart, 2008).

Sudoku represents a success case for confidence-aligned training. Under confidence-based decoding, PUMA performs best across every difficulty level: overall accuracy is 0.753 for PUMA versus 0.629 for random masking, and on the top-1% tier, PUMA reaches 0.609 versus 0.357 for random masking (Table 3). This contrasts with the addition and maze tasks. In Sudoku, highly confident predictions often correspond to logically ready cells: because each blank cell is constrained by its row, column, and 3×3 box, unmasking one correct cell immediately strengthens many other predictions. Confidence-based decoding therefore resembles true constraint propagation rather than a flawed shortcut that bypasses unresolved dependencies.

Uniform random decoding confirms that PUMA's gain is primarily trajectory-driven. On the top-1% tier, PUMA's lead over random masking shrinks from $+0.252$ under confidence-based decoding (0.609 vs. 0.357) to $+0.062$ under random decoding (0.432 vs. 0.370). For instances with a TL4 fraction ≥ 0.95, the lead similarly shrinks from $+0.132$ to $+0.032$. Thus, PUMA does not simply learn a uniformly stronger global representation; rather, it benefits because its confidence-trained trajectory is inherently well aligned with Sudoku's constraint-propagation structure.

5. Discussion

Confidence is a readiness heuristic, not a guarantee. Confidence-based decoding treats high token probability as evidence that a position is ready to be unmasked. This heuristic is useful when local predictability matches logical readiness, but it fails when a token is predictable via a shortcut while its true dependencies remain masked. The addition and maze tasks make this distinction clear: a digit or corridor cell can appear highly confident well before the underlying carry state or global path connectivity has been resolved. Sudoku represents the opposite case, where high-confidence predictions often perfectly align with the cells a constraint solver would logically fill next.

Confidence alignment narrows the states the model learns. Standard random masking trains the model across a diverse range of partial-observation states. In contrast, PAPL and PUMA bias training toward states that are likely to occur under confidence-based decoding. This bias can improve efficiency when the confidence trajectory is task-aligned, as seen in Sudoku. However, when confidence follows a shortcut, this same narrowing drastically reduces coverage of critical reasoning states. The result is either a *trajectory failure*, where a better decoding order can recover the model's performance, or a *representation failure*, where the underlying conditionals were never learned sufficiently to be recovered.

The main lesson. The core issue is not confidence-based decoding itself, but whether its decoding order faithfully matches the task's dependency structure. It succeeds when confidence correctly tracks logical readiness, but fails when confidence latches onto a locally correct yet globally incomplete shortcut. Therefore, evaluation should look beyond average accuracy under a single confidence decoder. Models must be tested on structurally hard cases, using solver-derived unmasking orders as diagnostics wherever possible.

6. Limitations

Two methodological assumptions deserve mention. First, we train small task-specific transformers ($\approx$ 0.4M to $\approx$ 21M parameters) and use greedy (deterministic) decoding throughout. The confidence-shortcut mechanism plausibly persists at larger scales and under stochastic decoding—locally-easy positions whose values depend on long unresolved chains pervade extended reasoning—but in those settings the dependency structure is harder to identify exactly and individual wrong commits become probabilistic rather than deterministic, so the failure mode is harder to characterize cleanly.

Second, our quantitative comparisons assume access to a clean dependency-respecting reveal order. This is exact for addition and maze, but for Sudoku and Countdown we approximate it with solver-derived orders involving search and backtracking, so the gap between confidence decoding and these orders conflates the confidence shortcut with the inherent suboptimality of backtracking-based sequences.

7. Conclusion

Masked diffusion language models offer flexible, any-order generation, but complex reasoning tasks fundamentally require the decoding order to follow the logical flow of resolving facts. While confidence-based decoding serves as a useful local proxy, it catastrophically fails when reflecting a superficial shortcut rather than true logical readiness. We observe this failure across multiple domains: on long carry chains in addition, on long corridors in maze, on deep expression trees in ListOps, and through narrowed mask-state coverage in Countdown. Our findings demonstrate that confidence-aligned training is a double-edged sword: it amplifies these failures by overfitting to local shortcuts when the trajectory diverges from the task's dependency structure, yet provides a benefit when they coincide, as in Sudoku. These results highlight a critical blind spot in current evaluation paradigms, where high average accuracy can mask profound reasoning deficits. Ultimately, the inference policy must align with the underlying reasoning order to unlock the full potential of masked diffusion models.

Impact Statement

This paper presents foundational research on the reasoning reliability of masked diffusion models (MDMs). By analyzing failure modes in complex tasks, our work provides technical insights essential for building more robust and predictable generative AI systems. These findings contribute to mitigating the risk of incorrect model outputs, supporting the broader goal of AI safety. As this is a technical study on model stability, we do not anticipate any direct negative societal consequences.

References

Aman, S. Logicdiff: Logic-guided denoising improves reasoning in masked diffusion language models. *arXiv preprint arXiv:2603.26771*, 2026.

Asano, H., Kozuno, T., Saito, K., and Baba, Y. Where-to-unmask: Ground-truth-guided unmasking order learning for masked diffusion language models. *arXiv preprint arXiv:2602.09501*, 2026.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Van Den Berg, R. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, 2021.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *ICML*, 2009.

Cai, C. and Li, G. Confidence-based decoding is provably efficient for diffusion language models. *arXiv preprint arXiv:2603.22248*, 2026.

Campbell, A., Benton, J., De Bortoli, V., Rainforth, T., Deligiannidis, G., and Doucet, A. A continuous time framework for discrete denoising models. In *NeurIPS*, 2022.

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. Maskgit: Masked generative image transformer. In *CVPR*, 2022.

Gandhi, K., Lee, D., Grand, G., Liu, M., Cheng, W., Sharma, A., and Goodman, N. D. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020.

Gong, S., Zhang, R., Zheng, H., Gu, J., Jaitly, N., Kong, L., and Zhang, Y. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Huang, Z., Chen, Z., Wang, Z., Li, T., and Qi, G.-J. Reinforcing the diffusion chain of lateral thought with diffusion language models. *arXiv preprint arXiv:2505.10446*, 2025.

Ivanitskiy, M. I., Shah, R., Spies, A. F., Räuker, T., Valentine, D., Rager, C., Quirke, L., Mathwin, C., Corlouer, G., Behn, C. D., et al. A configurable library for generating and manipulating maze datasets. *arXiv preprint arXiv:2309.10498*, 2023.

Katz, M., Kokel, H., and Sreedharan, S. Seemingly simple planning problems are computationally challenging: The countdown game. *arXiv preprint arXiv:2508.02900*, 2025.

Kim, B., Jeon, D., Kim, D., Jeung, W., and No, A. Rainbow padding: Mitigating early termination in instruction-tuned diffusion LLMs. In *ICLR*, 2026a.

Kim, J., Shah, K., Kontonis, V., Kakade, S. M., and Chen, S. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In *ICML*, 2025.

Kim, J., Geuter, J., Alvarez-Melis, D., Kakade, S., and Chen, S. Stop training for the worst: Progressive unmasking accelerates masked diffusion training. *arXiv preprint arXiv:2602.10314*, 2026b.

Lee, N., Sreenivasan, K., Lee, J. D., Lee, K., and Papailiopoulos, D. Teaching arithmetic to small transformers. In *ICLR*, 2024.

Lou, A., Meng, C., and Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.

McLeish, S., Bansal, A., Stein, A., Jain, N., Kirchenbauer, J., Bartoldson, B. R., Kailkhura, B., Bhatele, A., Geiping, J., Schwarzschild, A., et al. Transformers can do arithmetic with the right embeddings. In *NeurIPS*, 2024.

Nangia, N. and Bowman, S. Listops: A diagnostic dataset for latent tree learning. In *NAACL*, 2018.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Ou, J., Nie, S., Xue, K., Zhu, F., Sun, J., Li, Z., and Li, C. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

Peng, F. Z., Bezemek, Z., Rector-Brooks, J., Zhang, S., Bronstein, M. M., Zhang, A., Bose, J., and Tong, A. Planner aware path learning in diffusion language models training. In *ICLR*, 2026.

Sahoo, S. S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J. T., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models. In *NeurIPS*, 2024.

Shah, K., Dikkala, N., Wang, X., and Panigrahy, R. Causal language modeling can elicit search and reasoning capabilities on logic puzzles. In *NeurIPS*, 2024.

Shi, J., Han, K., Wang, Z., Doucet, A., and Titsias, M. K. Simplified and generalized masked diffusion for discrete data. In *NeurIPS*, 2024.

St Sauver, E. J. Think first, diffuse fast: Improving diffusion language model reasoning via autoregressive plan conditioning. *arXiv preprint arXiv:2603.13243*, 2026.

Stuart, A. SudokuWiki: Solving strategies. Online resource, 2008. URL https://www.sudokuwiki.org/Strategy_Families.

t-dillon. Tdoku: A fast sudoku solver and generator. GitHub repository, 2019. URL https://github.com/t-dillon/tdoku.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C., Wu, Y., Lu, M., Song, S., and Yadkori, Y. A. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025a.

Wang, G., Turok, G., Schiff, Y., Arriola, M., and Kuleshov, V. d2: Improved techniques for training reasoning diffusion language models. *arXiv preprint arXiv:2509.21474*, 2025b.

Ye, J., Gao, J., Gong, S., Zheng, L., Jiang, X., Li, Z., and Kong, L. Beyond autoregression: Discrete diffusion for complex reasoning and planning. In *ICLR*, 2025a.

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025b.

Zhao, S., Gupta, D., Zheng, Q., and Grover, A. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.

A. Related Works

Discrete and masked diffusion models. Discrete diffusion models provide an alternative to autoregressive language modeling by iteratively denoising categorical sequences (Austin et al., 2021; Campbell et al., 2022; Lou et al., 2023). Masked diffusion models (MDMs) instantiate this with an absorbing `[MASK]` state and a denoiser trained to reconstruct masked tokens (Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2024), and have recently scaled to language-model settings competitive on reasoning, planning, and code-generation benchmarks (Nie et al., 2025; Ye et al., 2025b; Gong et al., 2025; Ye et al., 2025a; Zhao et al., 2025).

Decoding policies and order-aware inference. The de facto MDM inference default is confidence-based parallel decoding—revealing positions by top probability (Chang et al., 2022), top-two margin (Kim et al., 2025), or negative entropy (Ye et al., 2025b). Kim et al. (2025) relate this to the order-agnostic training objective, and Cai & Li (2026) show an entropy-sum variant is provably efficient under conditions unrelated to logical-flow structure. Structural alternatives to confidence decoding have also been proposed: logic-role classifiers that unmask premises first (Aman, 2026), planners that imitate ground-truth oracles (Asano et al., 2026), and autoregressive plans prepended as frozen scaffolds (St Sauver, 2026). We treat these decoding policies as fixed inference choices and ask how training-time alignment with them shapes the trained representation.

Planner-aligned training. The closest methods bias the MDM loss or mask distribution toward an inference-time planner. PAPL (Peng et al., 2026) reweights the per-token loss by the model's own confidence; PUMA (Kim et al., 2026b) replaces the i.i.d. forward process with a teacher-forced confidence-based chain, asymptotically matching inference-time mask-state marginals. Reinforcement-learning approaches likewise concentrate compute along inference-aligned trajectories (Zhao et al., 2025; Wang et al., 2025b; Huang et al., 2025). Our results identify a complementary failure mode: when the inference policy diverges from the task's logical-flow order, planner alignment trades away the exploration random masking provides, and the conditionals lost are precisely those needed on adversarial reasoning inputs. Curriculum-style training distributions (Bengio et al., 2009) face an analogous coverage-vs-concentration trade-off in a different setting.

Algorithmic reasoning and shortcut learning. The lookahead shortcut on addition instantiates the broader pattern of shortcut learning (Geirhos et al., 2020), where models exploit distributionally-correlated cues that fail out-of-distribution. Multi-digit addition is a long-standing length-generalization probe for transformers (Lee et al., 2024; McLeish et al., 2024), and recent benchmarks expose long-range dependency failures in masked generation (Wang et al., 2025a; Shah et al., 2024). We identify training-distribution narrowing under confidence-aligned objectives as a specific mechanism producing such shortcut-like failures, and provide diagnostic signatures distinguishing failure modes within this class.

B. Data formats and generation

This appendix specifies the token-level format, generation procedure, and difficulty stratification for each domain. For each domain we indicate which positions of the sequence constitute the *prompt region* (visible to the model from the start of generation, never masked) and which constitute the *answer region* (masked at training time and predicted by the model). Probe NLL and generation accuracy are computed over the answer region only.

B.1. Addition

Task and source. We follow the addition task formulation of Lee et al. (2024): given two operands, the model produces their sum digit-by-digit. We use 32-digit operands.

Vocabulary. The vocabulary consists of digits 0–9, the symbols + and =, plus `M` and `PAD`.

Sequence format. A single training example has the form

$$\underbrace{a_0 a_1 \cdots a_{31} + b_0 b_1 \cdots b_{31} =}_{\text{prompt}} \underbrace{c_0 c_1 \cdots c_{32}}_{\text{answer}},$$

where digits are written in standard most-significant-first order with zero-padding to fixed width. The prompt occupies 66 characters; the answer occupies 33 characters (one extra digit accommodates the carry-out of the most significant addition). For example, $a = 47$, $b = 38$ (displayed at 4-digit width) yields the sequence `0047+0038=00085`.

Generation. Operands a, b are sampled uniformly from $\{0, \ldots, 10^{32} - 1\}$. The sum $c = a + b \mod 10^{32}$ is computed deterministically.

Difficulty stratification. For each digit position i, let $s_i = a_i + b_i$. By Equation 1, the carry dependency propagates through position i if and only if $s_i = 9$; for $s_i \leq 8$ the carry is killed (becomes 0) and for $s_i \geq 10$ the carry is generated (becomes 1), regardless of the incoming carry. The carry chain length of an instance is the longest run of consecutive positions with $s_i = 9$, bounded by a generate or kill event. This equals the length of the dependency window required to resolve the carry at the end of the chain, matching the rare-event probability in Equation 2. Under uniform operand sampling, chains of length ≥ 28 appear in well under 1% of training samples.

Set sizes. Train: 20,000 instances. Test: 10,000 random instances plus 500 instances per stratification level for the carry-chain sweep.

B.2. Maze

Task and source. We adapt the maze-completion task of Ivanitskiy et al. (2023): given the start and goal positions of a 10×10 logical maze, the model fills in the wall/corridor assignment of every cell so that a single connected path links start to goal.

Vocabulary. Cell-content tokens `#` (wall), `.` (corridor cell, in puzzle), `S` (start), `E` (end), and `0`/`1` (solution: 0 = corridor not on path, 1 = corridor on path), plus =, `[MASK]`, `[PAD]`.

Sequence format. A maze is encoded as a 21×21 wall-and-corridor grid (every other row and column is a wall layer between corridor cells), serialized in row-major order without separators. The prompt half shows the maze structure with all corridor cells unmarked (`.`); the answer half shows the same grid with each corridor cell labeled 0 or 1 depending on whether it lies on the start-to-end path. A short example illustrates the format on a 3×3 logical maze (rendered in 7×7 grid, linebreaks added for clarity—actual sequences are flat strings):

```
prompt:     answer:
#######     #######
#S.#..#     #S0#00#
#.#.#.#     #1#0#0#
#.#.#.#     #1#0#0#
#.....#     #11111#
#####.#     #####1#
#....E#     #0000E#
#######     #######
```

The full sequence is `puzzle=solution` where `puzzle` and `solution` are the row-major flattened grids (441 cells each for the 10×10 logical maze used in our experiments).

Generation. Mazes are generated by randomized depth-first search starting from a random corridor cell, producing a connected tree of corridors. Start and goal are chosen as the most distant pair of corridor cells along the resulting tree, ensuring a unique simple path between them.

Difficulty stratification. The backbone is the simple corridor path from start to goal. Branching points are corridor cells on the backbone with three or more accessible neighbors. Corridors are maximal runs of consecutive backbone cells between branching points. We stratify test instances by maximum corridor length.

Set sizes. Train: 10,000 mazes. Test: 5000 mazes with 300 mazes per corridor-length stratum.

B.3. ListOps

Task and source. We use the ListOps task introduced by Nangia & Bowman (2018) with operators MIN, MAX, MEDIAN and SUM-MOD-10, restricted to operands 0–9. We modify the original formulation to require the model to produce

intermediate values for every sub-expression, not only the root, so the answer region exposes the full computation graph.

Vocabulary. Digits 0–9, operator characters `X` (MAX), `N` (MIN), `D` (MEDIAN), `S` (SUM-MOD-10), brackets `[`, `]`, equality marker =, plus `[MASK]` and `[PAD]`. The trace rainbow-pads to a fixed maximum length (Appendix B.6).

Sequence format. The expression appears in prefix-bracketed form as the prompt, followed by the post-order evaluation trace (one digit per sub-expression value, inner-to-outer). For the input `[X 3 [N 2 5] 7]` (max of $\{3, 7, \min(2, 5)\} = 7$):

$$\underbrace{\texttt{[X 3 [N 2 5] 7] =}}_{\text{prompt}} \quad \underbrace{\texttt{27}}_{\text{trace: inner=2, outer=7}} .$$

The trace gives the result of each sub-expression in evaluation order: the inner `[N 2 5]` evaluates to 2, then the outer `[X ...]` to 7. The model predicts every digit of the trace.

Generation. Trees are sampled recursively: at each non-leaf, an operator is chosen uniformly; the number of children is sampled from a fixed distribution peaked at 2–4. Leaf operands are sampled uniformly from 0–9. We generate trees of target depth 1–6 to match the stratification axis.

Difficulty stratification. Tree depth (longest leaf-to-root path).

Set sizes. Train: 20,000 trees, depths 1–5. We apply depth-based decay with a 0.5 ratio to intentionally make difficult examples rare in the training distribution. Test: 1,000 trees per depth bin.

B.4. Countdown

Task and source. The countdown task asks the model to combine four input numbers using $\{+, -, \times, /\}$ to reach a target value. We use the dataset of Ye et al. (2025a), with the random generation procedure and multiplicity annotations following Gandhi et al. (2024).

Vocabulary. Digits 0–9, operators $+, -, *, /$, equality marker =, comma `,`, plus `[MASK]` and `[PAD]`, and rainbow padding characters.

Sequence format. Inputs and target form the prompt; the equation chain forms the answer. The format follows Ye et al. (2025a): four input numbers and one target, comma-separated, followed by an equation chain of three binary operations:

$$\underbrace{\texttt{86,28,13,31,96}}_{\text{prompt: 4 inputs, target}} = \underbrace{\texttt{86+28=114,31-13=18,114-18=96}}_{\text{answer: 3-step equation chain}}.$$

Each step in the chain consumes two operands (drawn from the input pool or freshly produced intermediates) and writes a new number. The final step's result equals the target. Sequence length varies across instances and is rainbow-padded.

Difficulty stratification. Solution multiplicity m is the number of distinct equation chains that reach the target, computed by the dataset's annotator. We use bins $m \in [1, 3]$ (rare-hard), $m \in [4, 10]$ (medium), and $m \geq 11$ (common easy).

Sub-sampling. Following the rare-hard framing, we retain only 10% of training puzzles with $m \in [1, 3]$, while higher-multiplicity puzzles are kept at natural frequency. The test set retains the natural distribution. This puts the rare-hard stratum at $\sim 5\%$ of training samples but $\sim 48\%$ of test samples, isolating each method's ability to generalize from under-represented training patterns.

Set sizes. Train: approximately 400,000 puzzles after sub-sampling. Test: 1,000 puzzles at natural distribution.

B.5. Sudoku

Task and source. Standard 9×9 sudoku. We use the `sudoku-extreme` dataset (Wang et al., 2025a), originally compiled for the Hierarchical Reasoning Model. The dataset contains approximately 3.8 million puzzles spanning a wide

range of difficulties, each annotated with a rating computed by the tdoku solver (t-dillon, 2019), defined as the number of guesses (binary decision points along the search tree) required by an MRV-heuristic backtracking solver. Rating zero indicates the puzzle is solvable by constraint propagation alone; the maximum rating in the dataset is 465, with 99% of puzzles below rating 101.

Vocabulary. Digits 1–9, blank token `.`, equality marker =, plus `[MASK]` and `[PAD]`. Total 13 tokens.

Sequence format. The puzzle and its solution are flattened in row-major order. The prompt shows the puzzle with blank cells marked by a placeholder; the answer is the complete 81-digit solution:

```
prompt:  53..7....6..195....98....6.
         8...6...34..8.3..17...2...6.
         6....28....419..5....8..79
answer:  534678912672195348198342567859
         761423426853791713924856961537
         284287419635345286179
```

where the prompt's 81 cells include blanks marked `.` and the answer is the corresponding 81-cell solution. The model sees the puzzle layout including which cells are blank, and predicts the complete solution. Generation accuracy is measured over the 81-cell solution; probe NLL and stratified accuracy are restricted to the originally blank cells (averaging ~ 56 per puzzle).

Difficulty stratification. We use two complementary axes.

Rating tiers. Six tiers determined from the rating distribution of `sudoku-extreme` via quantile boundaries on non-zero ratings. The median rating is 17; the 75th, 90th, 95th, and 99th percentiles are 34, 51, 65, and 101 respectively. Tier boundaries are placed at these quantiles, yielding:

- *easy*: rating 0 (singles only)
- *medium*: rating 1–17 ($\leq p_{50}$ of non-zero)
- *hard*: rating 18–34 (p_{50}–p_{75})
- *very hard*: rating 35–51 (p_{75}–p_{90})
- *extreme*: rating 52–101 (p_{90}–p_{99})
- *top 1%*: rating ≥ 102 ($> p_{99}$, up to dataset max 465).

Per-cell technique levels. For decoding-strategy analysis, we classify each blank cell by the deepest solving technique required to deduce its value, following the standard Sudoku-solving technique hierarchy (Stuart, 2008; Shah et al., 2024). We use a five-level hierarchy:

- **TL0 (singles)**: Naked singles and hidden singles—cells whose value is forced by direct row/column/box elimination.
- **TL1 (subsets and intersections)**: Naked/hidden pairs and triples, pointing pairs, box-line reductions.
- **TL2 (fish and wings)**: X-Wing, Swordfish, XY-Wing patterns.
- **TL3 (forcing chains)**: Single-step hypothesis-contradiction chains.
- **TL4 (search)**: Cells that cannot be deduced by any of the above and require bifurcation. This is the true "no stepping stone" regime.

The technique-difficulty decoding strategy reveals cells in increasing TL order; the constraint-propagation reveals them in the order an implementation of the above hierarchy uncovers them.

Set sizes. Train: a balanced sample with exponential decay across rating tiers (decay factor 0.01, yielding approximately 570,000 puzzles dominated by *easy* and *medium* but with at least one puzzle from each tier). Test: 1,000 each for *easy* and *medium*; 500 for *hard*; 200 for *very hard*; 100 for *extreme*; 500 for *top 1%*.

B.6. Rainbow padding for variable-length answers

For domains with variable answer length (ListOps, countdown), the answer region is padded to a fixed length to support batched training. Rather than a single repeated pad token, we use a deterministic cyclic pattern over a small set of distinct pad tokens, following Kim et al. (2026a). The cyclic structure distributes probability mass across the padding region, preventing pad positions from dominating early steps of confidence-based decoding.

C. More Details for the Experiments

C.1. Architecture

Table 4 describes the model architectures used in the experiments. All architectures are pre-norm transformers with bidirectional self-attention, GELU activations, learned absolute positional embeddings, weight-tied input/output projections, and an MLP hidden dimension of $3 \times$ embedding dim.

domain	layers	heads	embed dim	total params
addition	2	2	128	~ 0.4M
maze	3	3	192	~ 1.4M
ListOps	3	3	192	~ 1.4M
countdown	12	12	384	~ 21M
sudoku	8	8	256	~ 6.4M

Table 4. Model architecture per domain.

C.2. Training

Table 5 describes the training configuration used in the experiments.

domain	iterations	batch	epochs	LR
addition	300,000	256	$\approx 4{,}000$	$1 \cdot 10^{-3}$
maze	50,000	256	$\approx 1{,}300$	$3 \cdot 10^{-4}$
ListOps	300,000	256	$\approx 4{,}000$	$3 \cdot 10^{-4}$
countdown	200,000	256	≈ 130	$3 \cdot 10^{-4}$
sudoku	300,000	256	≈ 130	$3 \cdot 10^{-4}$

Table 5. Training hyperparameters.

All models trained with AdamW ($\beta = (0.9, 0.95)$), weight decay 0.01, gradient clipping at 1.0. Early stopping is disabled across all runs to remove convergence speed as a confound between methods. Training was performed on a single A100 GPU, and each experiment took approximately 4 hours for addition, maze, and ListOps, and 10 hours for Countdown and Sudoku.

C.3. PAPL

PAPL (Peng et al., 2026) is a one-line modification to the standard MDM loss. Given a batch of partially masked sequences and the model's predicted log-probabilities, PAPL replaces the uniform per-token weighting with

$$w_i = \frac{1}{|\mathbf{M}|}\Big(1 + \alpha \cdot \tilde{w}_i\Big), \quad \tilde{w}_i = \mathrm{softmax}_{j \in \mathbf{M}}\left[\frac{1}{\tau}\,\log p_\theta\big(x_j \mid \mathbf{x}_{\overline{\mathbf{M}}}\big)\right]_i,$$

where the softmax is taken over masked positions only, $\alpha \geq 0$ controls the strength of confidence-aligned reweighting (recovering standard MDM at $\alpha = 0$), and τ controls the sharpness of the planner distribution. The reweighting requires only the per-position predicted log-probability of the ground-truth token—already available during the standard loss computation—so the entire change consists of constructing $\tilde{w}_i$ from these log-probabilities and multiplying the per-position cross-entropy by $1 + \alpha\tilde{w}_i$ before averaging. We use $\alpha = 5$ and $\tau = 1$, the values used in the main experiments of Peng et al.

(2026), for maze, ListOps, countdown, and sudoku. For addition, $\alpha = 5$ collapses representation to zero accuracy at every chain length, so we report PAPL with $\alpha = 1$ for addition.

C.4. PUMA

PUMA (Kim et al., 2026b) replaces the i.i.d. forward process with teacher-forced trajectories generated under the model's current confidence-based policy. Concretely, given an integer hyperparameter K, the answer length L is partitioned into K stages, each unmasking roughly L/K tokens; smaller K yields coarser stages (more tokens unmasked per step), and larger K yields finer stages closer to the inference-time confidence-based reveal trajectory.

Streaming buffer. A naive implementation of PUMA would require generating each teacher-forced trajectory from scratch per training example, costing K forward passes per gradient step. Instead, following the original implementation, we maintain a streaming buffer of B teacher-forced chains (one per batch slot). At each gradient step, the model is trained on the current state of each chain, after which one stage of unmasking is applied to advance each chain to its next state. When a chain reaches full unmasking (end of its K stages), it is replaced with a fresh fully-masked sequence. This amortizes the chain-generation cost across K training steps, making PUMA's per-step compute essentially identical to standard MDM training—one forward pass per gradient step, with the chain-advancement forward pass being the same one used to compute confidence scores. The training distribution thus consists of intermediate states sampled uniformly across the K stages of confidence-based reveal trajectories.

Stochastic stage size. The number of tokens unmasked at each stage is not fixed at $\lfloor L/K \rfloor$. Following the original implementation, we add a small amount of randomness to avoid imbalanced exposure to particular mask patterns: at each stage the actual number of tokens revealed is drawn from a small range around L/K.

K-schedule. Because the model's confidence policy is unreliable early in training, the original PUMA paper uses a K-schedule that ramps from a smaller K_{start} to a larger K_{end} over the first portion of training and is held constant thereafter. The original paper uses $K = 8$ fixed for sudoku ($L = 81$, so ~ 10 tokens revealed per step) and a ramp from $K = 12$ to $K = 42$ for TinyGSM (max length $L = 512$, with the upper end giving ~ 12 tokens per step). The authors note these settings trade some training-inference alignment for compute savings during pretraining. Since our experiments are designed to study the consequences of such alignment rather than to optimize compute, we use K values that more faithfully realize PUMA's design intent (a few tokens per step), scaled to each domain's answer length. Due to the computational constraint, maze follows approximately same scheduling with TinyGSM from the original paper.

domain	answer length L	K_{start}	K_{end}	L/K_{end} (tokens/step)
addition	33	3	16	~ 2
maze	441	10	40	~ 10
ListOps	20	2	10	~ 2
countdown	40	4	20	~ 2
sudoku	81	8	40	~ 2

Table 6. PUMA K-schedule per domain. The K-schedule ramps over the first third of training and is held constant thereafter, following the original PUMA recommendation. The L/K_{end} column gives the asymptotic tokens revealed per step at the schedule's tightest setting.

C.5. Decoding strategies

Given the model's predicted distribution $p_\theta(\cdot \mid \mathbf{x}_{\overline{\mathbf{M}}})$ at each masked position $i \in \mathbf{M}$, we evaluate the following strategies for selecting the next position i' to reveal:

$$\begin{aligned} \text{Confidence:} \quad & i' = \arg\max_{i \in \mathbf{M}} c_\theta^i, \\ \text{Random:} \quad & i' \sim \mathrm{Unif}(\mathbf{M}), \\ \text{Algorithmic Optimal:} \quad & i' = \pi_{\text{task}}\big(\mathbf{M}, \mathbf{x}_{\overline{\mathbf{M}}}\big), \end{aligned}$$

where $c_\theta^i = \max_{v \in \mathcal{V}} p_\theta(v \mid \mathbf{x}_{\overline{\mathbf{M}}})$ is the top-1 confidence at position i (Section 2.2) and π_{task} is a polynomial-time deterministic function for addition (LSB-first) and maze (dead-end-filling). For ListOps, countdown, and sudoku, where the

underlying task is NP-hard or admits multiple valid orderings, we evaluate solver-derived strategies—layered post-order for ListOps (depth-first, ties broken by confidence), step-sequential for countdown (left-to-right within the equation chain), and constraint-propagation order or technique-difficulty order for sudoku.

Once the position i' is chosen, the token to commit is the argmax of the predictive distribution at that position:

$$x_{i'} \leftarrow \arg\max_{v \in \mathcal{V}} p_\theta\big(v \mid \mathbf{x}_{\overline{\mathbf{M}}}\big).$$

We do not introduce stochasticity into token selection (no Gumbel noise, top-p/top-k sampling, or temperature) in order to isolate the effect of the training scheme from the confounding effect of decoding randomness. All decoding produces one sample per puzzle; generation accuracy is exact-match against the answer, except for sudoku (cell accuracy).

D. Additional results

The main body reports each task under its most informative decode pair: typically confidence-based decoding versus an efficient algorithmic optimal where one exists, or versus uniform random reveal where it does not. For three tasks we report additional decoding results here.

For addition, we extend Table 1 with uniform random reveal to align the presentation with the other domains; the same three schemes trained for the addition experiment are evaluated under random decode (Table 7).

For countdown and sudoku, we report *solver-order* decoding strategies—reveal orders derived from a backtracking solver's traversal of the solution rather than from an efficient algorithmic optimal. Because the underlying tasks are NP-hard or admit multiple valid orderings, no polynomial-time oracle exists; the solver orders we use are deterministic and reproducible, but their reveal sequence corresponds to a search-with-backtracking traversal, not an efficient one. Empirically they produce strictly worse accuracy than confidence-based decoding for every training scheme we test, at every difficulty stratum. We include them as diagnostics: they confirm that the cross-scheme ordering observed under confidence decoding is not an artifact of the decode policy, and they make explicit what goes wrong when the reveal order is structurally constrained but inefficient.

D.1. Addition: uniform random decoding

Random decoding collapses accuracy for all three schemes. Even random masking—which trains on the full i.i.d. mask distribution—drops from 0.992 under confidence decode to 0.424 at chain ≥ 4 and to near zero at longer chains. The collapse is structural: addition's carry chain has no redundancy, so any reveal order that commits a chain-MSB cell before its carry-in is resolved produces a wrong digit that propagates through the rest of the answer.

D.2. Countdown: step-sequential reveal

The step-sequential reveal resolves one equation step at a time, left-to-right within the equation chain. This is a task-natural generation order: high-multiplicity instances admit multiple valid chains, so left-to-right is one consistent choice among several. The order tracks how a forward solver would write out the chain once it has committed to a set of operands at each step, and is backtracking-based in the sense that an arbitrary commit at one step may require a different chain to reach the target on harder instances. Results across all multiplicity bins are within a few points of confidence decode.

D.3. Sudoku: solver-order reveals

We evaluate two solver-derived reveal orders, neither of which is an efficient algorithmic oracle. The *constraint-propagation order* reveals cells in the order a backtracking solver uncovers them. The *technique-difficulty order* reveals cells in increasing order of the deepest solving technique required (TL0–TL4; Appendix B), which is itself derived from a backtracking-based solver.

Both reveal orders produce strictly worse accuracy than confidence decode for every training scheme at every difficulty tier (Table 8b vs. Table 3), with constraint-propagation order ahead of technique-difficulty order throughout. The relative pattern across training schemes is preserved: PUMA leads random masking at every tier above easy.

Table 7. Accuracy on 32-digit addition under uniform random decoding, by carry-chain length.

chain $\geq$	Random	PAPL$_{\alpha=1}$	PUMA	PAPL$_{\alpha=5}$
4	0.424	0.000	0.538	0.000
12	0.002	0.000	0.010	0.000
20	0.000	0.000	0.000	0.000
24	0.002	0.000	0.000	0.000
28	0.032	0.000	0.004	0.000

Table 8. Solver-order decoding results, compared with confidence and random-reveal numbers in Table 3. (a) Countdown under step-sequential decode, by solution multiplicity. (b) Sudoku under constraint-propagation and technique-difficulty orders, by rating tier; cell-level accuracy.

multiplicity	Random	PAPL	PUMA
$m \in [1, 3]$	0.127	0.077	0.109
$m \in [4, 10]$	0.468	0.449	0.397
$m \geq 11$	0.962	0.956	0.482
overall	0.485	0.456	0.290

(a) Countdown step-sequential decode.

	Constr.-prop.			**Tech.-diff.**		
rating tier	Random	PAPL	PUMA	Random	PAPL	PUMA
overall	0.578	0.581	0.631	0.581	0.584	0.624
rating = top1%	0.369	0.378	0.452	0.364	0.385	0.425
TL4 frac ≥ 0.95	0.385	0.388	0.430	0.373	0.380	0.409

(b) Sudoku solver-order reveals.