Learned Relay Representations for Forward-Thinking Discrete Diffusion Models

Benjamin Rozonoyer [*1] **Jacopo Minniti** [*2] **Dhruvesh Patel** [*1] **Neil Band** [3] **Avishek Joey Bose** [4 5]
Tim G. J. Rudner [2 6] **Andrew McCallum** [1]

Website Code

Abstract

When Masked Diffusion Models (MDMs) generate sequences through iterative refinement, the rich internal computation over masked positions is discarded—forcing every subsequent refinement step to recompute the valuable internal information stored as model representations. To avoid a hard reset between denoising rounds, we propose Learned Relay Representations (RELAY), a method that allows MDMs to be "forward-thinking" when denoising—*explicitly learning how to propagate latent information for the benefit of future denoising steps*. RELAY introduces a differentiable per-token channel that passes information between forward passes and is trained via truncated backpropagation through time (BPTT). We show that this framework can be scaled to state-of-the-art Diffusion Language Models (DLMs), and is seamlessly compatible with techniques like block diffusion and KV caching. We first provide a thorough justification of the design choices in RELAY on a challenging Sudoku-based planning task. We then scale RELAY to Fast-dLLM v2, a state-of-the-art DLM, outperforming standard supervised finetuning on coding tasks while reducing the inference latency by up to 32%. Our empirical results demonstrate that state-of-the-art DLMs can be explicitly trained to *relay* latent information forward across decoding steps, advancing the performance-latency Pareto frontier. We provide code for all our experiments.

1. Introduction

Masked Diffusion Models (MDMs) generate discrete sequences via iterative denoising (Austin et al., 2021; Campbell et al., 2022; Sahoo et al., 2024; Shi et al., 2024): starting from a fully masked canvas, each forward pass unmasks a fraction of the remaining positions. The Transformer computes hidden states at every position—including those still masked—but discards them at the end of each step, beginning the next pass from the partially unmasked sequence alone. We call this the *hard reset* problem: the only information that persists across steps is the discrete tokens just committed, leaving MDMs with no way to accumulate intermediate continuous computation.

This matters because recurrent computation—unrolling a fixed-parameter model across many steps—is exactly the structural property that recent work has tied to improved performance on difficult reasoning tasks, as it effectively expands the function class the model can approximate (Gatmiry et al., 2024; Saunshi et al., 2025; Li et al., 2024). MDMs already perform many forward passes per generation; the hard reset is what prevents any of that compute from being reused.

This raises a natural question: How can the sequential unmasking structure of MDMs support recurrent computation that carries richer information across steps?

Our answer is Learned Relay Representations (RELAY), a method that makes discrete diffusion models *forward-thinking*: at each denoising step, alongside any newly unmasked tokens, the model carries its last-layer hidden states forward as a learned *relay*, giving the next forward pass direct access to the prior step's continuous computation. Simply piping these states forward, however, does not by itself ensure they encode anything useful for what follows. RELAY therefore trains

[*]Equal contribution

Accepted to FoGen 2026: Foundations of Deep Generative Models: Understanding Memorization, Generalization, and Reasoning, an ICML 2026 workshop (non-archival).

the relay end-to-end with truncated backpropagation through time (BPTT; Werbos, 1990), shaping it to be maximally informative for the next several denoising steps and enabling a form of latent chain-of-thought across the unmasking trajectory.

Contributions. We introduce RELAY, which equips MDMs with learned relay representations—continuous latent states passed forward across decoding steps and trained end-to-end via truncated BPTT. RELAY is architecture-agnostic and leaves the inference-time decoding procedure of MDMs (unmasking schedule, sampling) unchanged; the only addition at inference is forwarding the relay alongside the committed tokens. It is also compatible with prevalent DLM acceleration techniques, including block diffusion (Arriola et al., 2025) and KV caching (Wu et al., 2025b;a).

To summarize, our key contributions are as follows:

1. We propose RELAY, a general method for incorporating recurrent computation in MDMs by training the model—via truncated BPTT—to pass a learned latent relay forward across decoding steps. RELAY can train an MDM from scratch or adapt a pre-trained MDM through lightweight adaptation.
2. We validate RELAY at LLM scale through full-parameter adaptation of Fast-dLLM v2 1.5B (Wu et al., 2025a), outperforming standard supervised finetuning on coding tasks while reducing inference latency by up to 32%.
3. We perform extensive ablations that map out the design space of RELAY and validate our choices.

2. Background: Masked Diffusion Models

We tackle the hard reset problem in masked diffusion models by training them to pass along a learned relay state. Before presenting our approach, RELAY, we review the training and inference procedure for Masked Diffusion Models (MDMs) (Shi et al., 2024; Sahoo et al., 2024) that RELAY builds upon.

Notation. We denote the vocabulary as $\mathcal{V}$, including the `[M]` token. The space of sequences of length L over the vocabulary is $\mathcal{V}^L$. Superscripts denote the position in the sequence, e.g., x^i is the i-th token in the sequence $\boldsymbol{x} \in \mathcal{V}^L$. $\mathcal{M}(\boldsymbol{x}) \subseteq [L]$ denotes the set of masked positions in the sequence $\boldsymbol{x}$.

Training. The noising process proceeds by sampling a time $t \in [0, 1]$ and masking each position in a clean sequence $\boldsymbol{x}_0 \in (\mathcal{V} \setminus \{\texttt{[M]}\})^L$ independently with probability α_t, to obtain the noised (partially masked) sequence $\boldsymbol{x}_t$. The coordinate-wise posterior distribution $\mathbb{P}(X_0^i = x_0^i \,|\, \boldsymbol{X}_t = \boldsymbol{x}_t)$ is denoted as $p(x_0^i \mid \boldsymbol{x}_t)$. As noted in Zheng et al. (2024), this posterior depends on $\boldsymbol{x}_t$ only through its masked pattern and revealed tokens, not on the time t itself. The coordinate-wise posterior is parameterized by a neural network denoted as $p_\theta^i(\cdot \mid \boldsymbol{x}_t) \in \Delta$ for $i \in \mathcal{M}(\boldsymbol{x}_t)$ and is trained by minimizing the weighted sum of cross-entropy losses for each masked position[1]

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\boldsymbol{x}_0, t, \boldsymbol{x}_t} \left[\frac{1}{t} \sum\nolimits_{i:\boldsymbol{x}_t^i = \texttt{[M]}} -\log p_\theta^i(x_0^i \mid \boldsymbol{x}_t) \right]. \tag{1}$$

The coordinate-wise parametric posterior is implemented using embedding $\text{EMB}_\theta : \mathcal{V} \to \mathbb{R}^d$, unembedding $\text{UNEMB}_\theta : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{V}|}$, and a transformer backbone $f_\theta : \mathcal{V}^L \to \mathbb{R}^{L \times d}$ that produce the posterior distribution:

$$p_\theta^i(w \mid \boldsymbol{x}_t) = \frac{e^{\ell^i(w)}}{\sum_{w' \in \mathcal{V}} e^{\ell^i(w')}}, \quad \text{where} \quad \ell^i(w) = \text{UNEMB}_\theta(f_\theta(\text{EMB}_\theta(\boldsymbol{x}_t)))_w^i.$$

Inference. Generation proceeds along a decreasing time grid $1 = t_0 > t_1 > \cdots > t_K = 0$, iteratively unmasking positions from the all-masked sequence $\boldsymbol{x}_{t_0} = (\texttt{[M]}, \ldots, \texttt{[M]})$ to a fully unmasked sequence $\boldsymbol{x}_{t_K} \in (\mathcal{V} \setminus \{\texttt{[M]}\})^L$. At each step k, given the current partially masked sequence $\boldsymbol{x}_{t_k}$, the model computes logits $\boldsymbol{\ell}_k$ for the per-position posterior distribution for each masked position $i \in \mathcal{M}(\boldsymbol{x}_{t_k})$ and token $w \in \mathcal{V}$. An unmasking policy $u(\cdot \mid \boldsymbol{\ell}_k, \boldsymbol{x}_{t_k})$, which may be stochastic, then selects a set of positions $\mathcal{U}_k \subseteq \mathcal{M}(\boldsymbol{x}_{t_k})$ to reveal, producing the next partially masked sequence $\boldsymbol{x}_{t_{k+1}}$. Common choices for $u(\cdot \mid \boldsymbol{\ell}_k, \boldsymbol{x}_{t_k})$ include unmasking a fixed fraction of the remaining masks at each step (Nie et al., 2025) and confidence-based rules (Ben-Hamu et al., 2025; Kim et al., 2025; Patel et al., 2025).

The Hard Reset Problem. After each inference step, MDMs discard the entire computational state used to choose the newly revealed tokens. The next step starts again from $\boldsymbol{x}_{t_{k+1}}$ alone. Thus, standard MDM inference treats each partially masked sequence as a fresh prediction problem—a *hard reset*—rather than as a continuation of an ongoing computation. Because models can only perform a constant number of FLOPs in each forward pass, hard reset prevents the model

[1] We have assumed a linear noise schedule.

from amortizing reasoning across steps effectively. In the next section, we propose our solution to this problem: we learn a continuous latent state that is passed across the steps of MDM inference.

3. Learned Relay Representations

To address the *hard reset* problem, we introduce a continuous differentiable state that is carried across MDM inference steps and can circumvent the hard reset.

3.1. Augmented State Trajectories

The training of MDMs proceeds by sampling a data point $\boldsymbol{x}_0 \sim p_{\text{data}}$, a time $t \sim \mathcal{U}(0,1)$, and a partially masked sequence $\boldsymbol{x}_t$ under the noise schedule given the time t and the data point $\boldsymbol{x}_0$. During inference, we have a discretized time grid $1 = t_0 > \cdots > t_n = 0$, and the corresponding inference trajectory $\boldsymbol{x}_{t_0}, \ldots, \boldsymbol{x}_{t_n}$ obtained by using some unmasking policy u, where $\boldsymbol{x}_{t_0} = \{\texttt{[M]}\}^L$. We wish to pass a continuous state forward across decoding steps, that can carry intermediate computations from the previous step which have not yet been realized as a decoded token. We can break down this behavior into two primitives: a model must produce a relay state $\boldsymbol{h}_k$ at inference step k, and learn to consume that relay state at step $k{+}1$. Figure 1 shows a schematic of the augmented state trajectory produced by the model, where $\boldsymbol{s}_k = (\boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$ is the augmented state at step k.

3.2. Training

Architecture. We parameterize the augmented dynamics with a backbone f_θ, relay module R_θ, token embedding $\textsc{Emb}_\theta$, and unembedding head $\textsc{UnEmb}_\theta$ (see Figure 1). At step k, the model maps the current pair $(\boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$ to the next relay state and per-position logits via

$$\boldsymbol{h}_{k+1} = f_\theta(\textsc{Emb}_\theta(\boldsymbol{x}_{t_k}) + R_\theta(\boldsymbol{h}_k)), \qquad \boldsymbol{\ell}_k = \textsc{UnEmb}_\theta(\boldsymbol{h}_{k+1}), \tag{2}$$

initialized with $\boldsymbol{h}_0 = \mathbf{0}$. The per-position posterior $p^i_\theta(\cdot \mid \boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$ is read off from $\boldsymbol{\ell}_k$ by a softmax, exactly as in standard MDMs.

Since we only care about the terminal state $\boldsymbol{x}_{t_n}$, we continue to provide supervision using the same cross-entropy loss as in standard MDMs, and train the model to produce useful relay states $\boldsymbol{h}_k$ that help improve predictions K steps ahead using truncated BPTT. Specifically, instead of sampling $\boldsymbol{x}_t$ as in standard MDMs, we start from an all masked sequence $\boldsymbol{x}_{t_0} = \{\texttt{[M]}\}^L$ and roll out under Equation (2) together with an unmasking policy u (see below), producing the augmented trajectory $(\boldsymbol{x}_{t_0}, \boldsymbol{h}_0), \ldots, (\boldsymbol{x}_{t_n}, \boldsymbol{h}_n)$. The total training loss is the expected sum of per-step cross-entropies over the trajectory:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\boldsymbol{x}_0,\, \xi_{0:n-1}} \left[\sum_{k=0}^{n-1} \sum_{i \in \mathcal{M}(\boldsymbol{x}_{t_k})} -\log p^i_\theta\big(x^i_0 \mid \boldsymbol{x}_{t_k}, \boldsymbol{h}_k\big) \right], \tag{3}$$

where $\xi_{0:n-1}$ denotes the exogenous randomness used by the unmasking policy along the rollout. Unlike an externally observed conditioning variable, $\boldsymbol{h}_k$ is an internal artifact of the rollout, part of the computational trajectory rather than of the generated object. At inference time each step carries forward the realized pair $(\boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$, but only $\boldsymbol{x}_{t_k}$ is eventually decoded into text, while $\boldsymbol{h}_k$ serves as a differentiable memory channel for future predictions. The full procedure is summarized in Algorithm 1; we derive the gradient estimator below.

Constructing rollouts. In order to perform truncated BPTT, we need to construct rollouts of the augmented state trajectory under an unmasking policy u. Given the current augmented state $(\boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$, one step of rollout proceeds as follows:

- **Position selection:** Sample which positions to unmask, $\mathcal{U} \sim u(\cdot \mid \boldsymbol{\ell}_k, \boldsymbol{x}_{t_k})$. The policy may use the model's own logits $\boldsymbol{\ell}_k$.
- **Token forcing:** For each $i \in \mathcal{U}$, commit the token from ground truth: $x^i_{t_{k+1}} = x^i_0$.

We teacher-force the token *values* (rather than sampling from the model's posterior $p^i_\theta(\cdot \mid \boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$) because sampled values would inject errors that the rollout has no mechanism to correct. The *position* sampler, by contrast, may use the model's own posterior without affecting the ideal minimizer: in absence of the continuous channel this leaves the standard MDM training objective (Equation (1)) unchanged (Kim et al., 2026), and for the augmented-state trajectory the same argument applies but a formal proof requires additional assumptions and is more involved.

Algorithm 1: RELAY Training

Input: model f_θ, relay module R_θ, unroll horizon K, unmasking policy u, training steps N, learning rate η

for $t \in \{1, \dots, N\}$ **do**
  if $t = 1$ or $\mathcal{M}(\boldsymbol{z}) = \emptyset$ **then**
    $\boldsymbol{x}_0 \sim p_{\text{data}}$, $\boldsymbol{z} \leftarrow \{\texttt{[M]}\}^L$, $\boldsymbol{h} \leftarrow \mathbf{0}$
  end
  $L \leftarrow 0$
  for $k \in \{0, \dots, K-1\}$ **do**
    $\boldsymbol{h} \leftarrow f_\theta(\text{EMB}_\theta(\boldsymbol{z}) + R_\theta(\boldsymbol{h}))$
    $\boldsymbol{\ell} \leftarrow \text{UNEMB}_\theta(\boldsymbol{h})$
    $L \leftarrow L + \mathcal{L}(\boldsymbol{\ell}, \boldsymbol{x}_0)$ ▷ masked positions only
    $\mathcal{U} \sim u(\cdot \mid \boldsymbol{\ell}, \boldsymbol{z})$
    $z^i \leftarrow x_0^i \;\; \forall i \in \mathcal{U}$
  end
  $\theta \leftarrow \theta - \eta \nabla_\theta L$
end
return θ

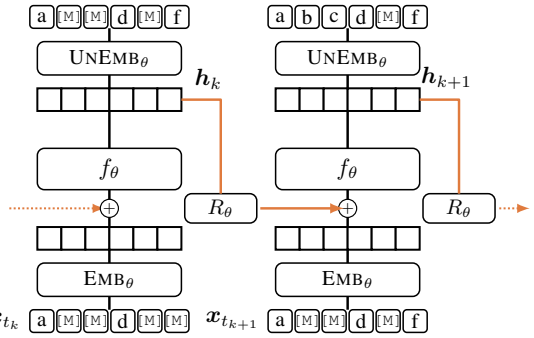


Figure 1. Schematic of RELAY over two consecutive inference steps. At each step k, the backbone f_θ consumes the sum of embedded tokens $\text{EMB}_\theta(\boldsymbol{x}_{t_k})$ and the projected relay state $R_\theta(\boldsymbol{h}_k)$, producing a hidden state $\boldsymbol{h}_{k+1}$ that is both unembedded into logits for the cross-entropy loss and forwarded through the relay module R_θ (orange path) into the next step. Tokens are progressively unmasked between steps (e.g. [M] →f at step k, [M] →b, c at step $k{+}1$), while $\boldsymbol{h}$ provides a continuous, differentiable channel for information that has not yet been committed to a discrete token.

Gradient estimation. We now derive the gradient estimator for one K-step window of the recurrence Equation (2). Let ξ_k denote the exogenous randomness used in the sampled unmasking step at k:

$$\mathcal{U}_k \sim u(\cdot \mid \boldsymbol{\ell}_k, \boldsymbol{x}_{t_k}), \quad \text{and} \quad x^i_{t_{k+1}} \leftarrow x^i_0 \quad \forall i \in \mathcal{U}_k. \tag{4}$$

Conditioning on the realized $\xi_{0:K-1}$, the per-window loss is

$$\mathcal{L}_K(\theta; \boldsymbol{x}_0, \xi_{0:K-1}) = \sum_{k=0}^{K-1} L_k(\boldsymbol{\ell}_k, \boldsymbol{x}_0), \tag{5}$$

where L_k is the per-step cross-entropy at step k (summed over the masked positions of $\boldsymbol{x}_{t_k}$), and $\boldsymbol{\ell}_k$, $\boldsymbol{h}_{k+1}$ are computed from $(\boldsymbol{x}_{t_k}, \boldsymbol{h}_k)$ via Equation (2). The discrete update $\boldsymbol{x}_{t_k} \to \boldsymbol{x}_{t_{k+1}}$ is treated as fixed after the rollout is sampled. Equivalently, this estimator sets $\partial \boldsymbol{x}_{t_{k+1}} / \partial \boldsymbol{\ell}_k = 0$ and does not differentiate through the sampled unmasking decisions. The BPTT adjoints over the differentiable relay state are then defined by

$$\begin{aligned} &\lambda_K = 0, \\ &\lambda_k = (\partial_{\boldsymbol{h}_k} \boldsymbol{\ell}_k)^\top \nabla_{\boldsymbol{\ell}_k} L_k(\boldsymbol{\ell}_k, \boldsymbol{x}_0) + (\partial_{\boldsymbol{h}_k} \boldsymbol{h}_{k+1})^\top \lambda_{k+1}, \qquad k = K-1, \dots, 0. \end{aligned} \tag{6}$$

Throughout, $\partial_{\boldsymbol{h}_k} \boldsymbol{\ell}_k$ and $\partial_\theta \boldsymbol{\ell}_k$ denote the *total* derivatives along the single-step chain $\boldsymbol{h}_k \to \boldsymbol{h}_{k+1} \to \boldsymbol{\ell}_k$, i.e., $\partial_{\boldsymbol{h}_k} \boldsymbol{\ell}_k = (\partial_{\boldsymbol{h}_{k+1}} \text{UNEMB}_\theta)(\partial_{\boldsymbol{h}_k} \boldsymbol{h}_{k+1})$, and analogously for θ; the companion factor $(\partial_\theta \boldsymbol{h}_{k+1})^\top \lambda_{k+1}$ below uses the *direct* partial of step k's transition only ($\boldsymbol{h}_k$ held fixed). The boundary $\lambda_K = 0$ therefore reads as "no downstream losses past step $K{-}1$." The corresponding sampled gradient estimator is

$$\nabla_\theta \mathcal{L}_K = \sum_{k=0}^{K-1} \Bigg[\underbrace{(\partial_\theta \boldsymbol{\ell}_k)^\top \nabla_{\boldsymbol{\ell}_k} L_k(\boldsymbol{\ell}_k, \boldsymbol{x}_0)}_{\text{direct gradient from immediate cross-entropy}} + \underbrace{(\partial_\theta \boldsymbol{h}_{k+1})^\top \lambda_{k+1}}_{\text{BPTT through relay state}} \Bigg]. \tag{7}$$

For a two-step truncation beginning at step k, we have $\lambda_{k+2} = 0$, so the only downstream adjoint is

$$\lambda_{k+1} = \left(\partial_{\boldsymbol{h}_{k+1}} \boldsymbol{\ell}_{k+1}\right)^\top \nabla_{\boldsymbol{\ell}_{k+1}} L_{k+1}(\boldsymbol{\ell}_{k+1}, \boldsymbol{x}_0). \tag{8}$$

The two-step gradient is therefore

$$\nabla_\theta \left(L_k + L_{k+1}\right) = \underbrace{\sum_{j=k}^{k+1} (\partial_\theta \boldsymbol{\ell}_j)^\top \nabla_{\boldsymbol{\ell}_j} L_j(\boldsymbol{\ell}_j, \boldsymbol{x}_0)}_{\text{direct gradient from immediate cross-entropy}} + \underbrace{(\partial_\theta \boldsymbol{h}_{k+1})^\top \lambda_{k+1}}_{\text{BPTT through relay state}}.$$

Thus, each step receives the local cross-entropy gradient through its logits $\boldsymbol{\ell}_k$, and the additional recurrent gradient is back-propagated through the differentiable relay path $\boldsymbol{h}_k \to \boldsymbol{h}_{k+1}$.

4. Experiments

Through our experiments we seek to address the following research questions:

- **RQ1** Does training to be forward-thinking with BPTT improve performance and latency?
- **RQ2** Does weight-tying EMB_θ and UNEMB_θ have an impact on RELAY, since f_θ at the first layer must learn to consume the UNEMB_θ-aligned relay $\boldsymbol{h}$ from the last layer?
- **RQ3** Can we efficiently adapt state-of-the-art DLMs to use relay representations and improve their performance-latency frontiers with negligible additional training FLOPs?

We first motivate the design choices for RELAY with a thorough ablation on Sudoku. Subsequently, we post-train Fast-dLLM v2 (Wu et al., 2025a), a state-of-the-art DLM, demonstrating the effectiveness of RELAY on model adaptation for DLMs.

4.1. Sudoku

Dataset. The objective of a Sudoku puzzle is to fill in a 9x9 board (of nine 3x3 sub-squares) with digits 1-9 such that each row, column, and 3x3 square contains all the nine unique digits. A puzzle has a minimum of 17 clues, which is a mathematical prerequisite for it to have a unique solution (McGuire et al., 2012).

Setup. We choose the Sudoku-Extreme dataset (Wang et al., 2025) as a challenging benchmark that allows us to focus on modeling choices without the risk of overfitting. We release a derived version[2] that augments each puzzle with a step-by-step solver trajectory, step count, and the set of deduction strategies invoked, obtained by running the Sudoku solver of Vink (2024)[3] over every example; the `strategies` field underpins the *deduction-only* evaluation slice in Table 1. For the experiments in Figure 2 we evaluate on the first 50k examples of the test split, which are representative in difficulty (Appendix A.1.1). All methods use the same small Transformer backbone ($\sim$ 7M parameters; full architecture in Appendix A) with rotary position embeddings and are trained to convergence, in line with our experimental protocol of comparing methods by their test-time performance versus latency frontiers. [4] Our predictor uses top-probabilities as confidence values c_i, sorts by increasing $1 - c_i$, and unmasks all positions whose cumulative confidence falls below a threshold τ, falling back on the argmax if no such position exists. For RELAY's on-policy training rollout we use a stochastic threshold $\tau \sim \mathcal{N}(\mu = 0.15, \sigma = 0.1)$ for robustness (the threshold is a hyperparameter of the sampling decision $\mathcal{U} \sim u(\cdot \mid \boldsymbol{\ell}, \boldsymbol{z})$ in Algorithm 1 line 10).

Baselines and ablations. We compare four training objectives that progressively turn on the components of Algorithm 1, each instantiated with both *tied* and *untied* embeddings (whether EMB_θ and UNEMB_θ share weights). **MLM** (Sahoo et al., 2024; Shi et al., 2024) is standard uniform masked diffusion: a single forward pass per training step ($K{=}1$), no relay ($R_\theta \equiv 0$, so $\boldsymbol{h} \leftarrow f_\theta(\text{EMB}_\theta(\boldsymbol{z}))$), and no inner rollout. Instead, the masked input $\boldsymbol{z}$ is drawn fresh each step by sampling $t \sim \mathcal{U}(0, 1)$ and masking each token of $\boldsymbol{x}_0$ independently with probability t. The remaining three objectives all share RELAY's on-policy *position* sampler $u(\cdot \mid \boldsymbol{\ell}, \boldsymbol{z})$ (Algorithm 1 line 10) and teacher-force the committed positions to the values in $\boldsymbol{x}_0$ between passes (line 11), differing only in whether and how the relay channel is used (a related rollout training procedure is studied by Kim et al., 2026). **Rollout** unrolls $K{=}2$ inner steps but keeps $R_\theta \equiv 0$ so each step recomputes $\boldsymbol{h}$ from $\text{EMB}_\theta(\boldsymbol{z})$ alone; this isolates the contribution of *which* positions get committed between forward passes. **RELAY (sg)** additionally enables the relay path $R_\theta(\boldsymbol{h})$ inside the inner loop but stop-gradients $\boldsymbol{h}$ before feeding it back, so the backbone receives no temporal credit across the K steps. **RELAY** is the full method: $K{=}2$ BPTT through the relay (Algorithm 1). At inference we sweep deterministic thresholds $\tau \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$ and trace each method's accuracy-NFE frontier; lower τ commits fewer cells per forward pass and so spends more NFEs.

Results and analysis. Figure 2 plots validation metrics at the latest checkpoint for each seed. Replacing uniform masking (**MLM**) with the on-policy confidence-thresholded sampler under teacher forcing of the unmasked values (**Rollout**) yields the first improvement. Turning the relay channel on (**RELAY (sg)**) contributes the next big jump, highlighting the importance of a soft state carried between forward passes. Finally, replacing the stop-gradient with $K{=}2$ BPTT through the relay (**RELAY**, Algorithm 1) yields a further separation and the best accuracy-NFE frontier across thresholds.

We are able to trace this last separation of RELAY over RELAY (sg) to the fact that, at the same threshold τ, RELAY

[2] https://huggingface.co/datasets/brozonoyer/sapientinc-sudoku-extreme-timvink-sudoku-solver

[3] https://github.com/timvink/sudoku-solver

[4] We use the xLM package (https://github.com/dhruvdcoder/xlm-core) for all small-scale experiments on Sudoku.

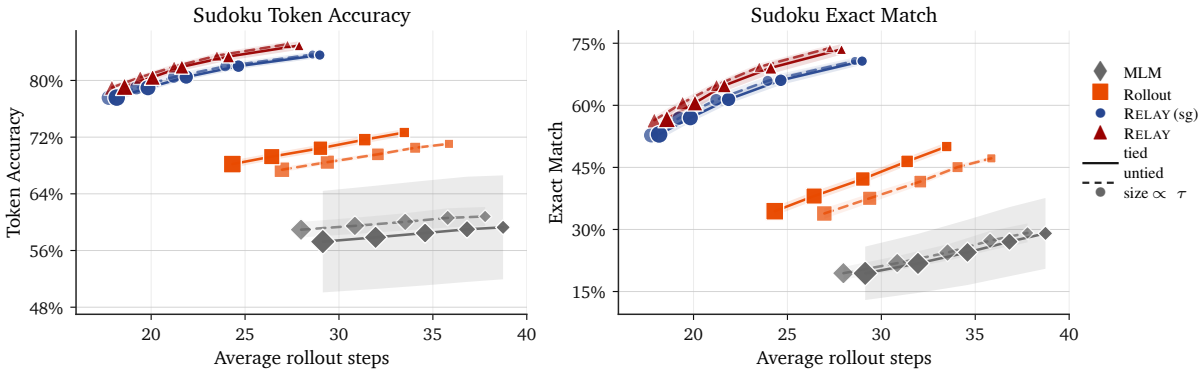


Figure 2. Accuracy-NFE frontier on Sudoku-Extreme validation. Each curve traces a single training method as we sweep the inference confidence threshold $\tau \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$. A lower τ commits fewer cells per forward pass and so spends more NFEs (rightward), and vice-versa. Shaded ribbons denote ± 1 sample standard deviation across three training seeds.

Table 1. Sudoku exact match and mean NFE at $\tau = 0.15$. *Unfiltered* reports performance on puzzles iterated from the test split in dataset order; *deduction-only* restricts to puzzles whose solver trace requires Advanced/Master heuristics (no recursive backtracking). Accuracies are % exact match, with sample s.d. across 3 training seeds. See Appendix A.1.1 for more details.

Objective	Tying	Acc (%) unfiltered	Mean NFE unfiltered	Acc (%) deduction-only	Mean NFE deduction-only
MLM	✓	20.27% ± 0.25%	13.76 ± 0.56	32.23% ± 1.87%	9.27 ± 0.17
	×	18.58% ± 2.17%	15.70 ± 0.32	35.67% ± 2.67%	10.23 ± 0.31
Rollout	✓	38.70% ± 2.09%	12.54 ± 0.31	52.93% ± 3.08%	10.45 ± 0.33
	×	35.55% ± 1.15%	13.94 ± 0.46	54.35% ± 4.32%	11.13 ± 0.23
RELAY (sg)	✓	58.42% ± 0.11%	7.62 ± 0.05	70.75% ± 0.92%	6.13 ± 0.09
	×	59.45% ± 3.06%	7.59 ± 0.11	70.65% ± 3.18%	6.01 ± 0.03
RELAY	✓	**62.67%** ± 2.40%	7.43 ± 0.18	**76.42%** ± 2.47%	5.86 ± 0.08
	×	62.07% ± 0.70%	**7.20** ± 0.18	73.10% ± 2.64%	**5.80** ± 0.08

commits more cells per forward pass while keeping the partial board legal — where a board is *legal* when no row, column, or 3×3 box yet contains a repeated digit. Legality is a necessary condition for correctness, and is well-defined at every intermediate denoising step, not only at the end. Since the studied architectures cannot perform recursive search, we restrict this qualitative analysis to a *deduction-only* cohort of 2,000 test puzzles for which the solver uses only human-like deduction strategies (Advanced or Master heuristics; cohort construction detailed in Appendix A.1.1).

At the matched threshold $\tau = 0.15$, RELAY produces a fully legal final board 74.8% of the time versus 70.7% for RELAY (sg) ($+4.1$ pp), and incurs 15% fewer row/column/box violations across the rollout (0.90 vs. 1.06 on average per puzzle); these legality gains are uniform across the Advanced ($+4.0$ pp) and Master ($+4.1$ pp) strata. In other words, **BPTT teaches the relay to keep the partial board self-consistent under more aggressive unmasking**: at the same confidence threshold τ, RELAY commits more cells per forward pass while still honoring the row/column/box constraints, so the rollout reaches the same accuracy in fewer total forward passes — producing the strict outward shift of the ($\tau \to$ accuracy-NFE) frontier in Figure 2. Table 1 reports the corresponding exact match and mean NFE at $\tau = 0.15$ on both the unfiltered test split and the deduction-only cohort: RELAY attains the highest exact match and the lowest mean NFE in every (slice, tying) cell, with a $+4$ to $+6$ pp gain over RELAY (sg) at uniformly lower NFE. Tying versus untying EMB_θ and UNEMB_θ has only a marginal effect on any objective (≤ 3 pp exact match across all rows), consistent with the small Sudoku vocabulary leaving the residual stream ample capacity to carry both predictive and relay-bearing information.

4.2. Pretrained Model Adaptation: Fast-dLLM v2

Next, we investigate whether state-of-the-art DLMs can be efficiently adapted into RELAY diffusion models with a limited amount of finetuning, and whether this adaptation can improve their accuracy-latency frontiers.

Table 2. Pretrained adaptation on Fast-dLLM-v2 (1.5B), evaluated at threshold 0.85. Average NFE is computed as the mean per-example count of active denoising forward calls during batched sample generation, excluding prompt prefill and final cache-update next-token forwards. Bold values are selected among adapted rows only, excluding the off-the-shelf baseline.

Method	HumanEval			MBPP		
	Base↑	Plus↑	NFE↓	Base↑	Plus↑	NFE↓
Fast-dLLM-v2 (1.5B)	38.4%	35.4%	178.1	46.8%	39.7%	133.0
Vanilla SFT	38.4%	34.1%	130.7	43.9%	38.1%	84.8
RELAY (sg)	38.4%	35.4%	104.4	43.1%	39.2%	80.1
RELAY	**42.1%**	**37.2%**	**88.3**	**46.6%**	**41.5%**	**78.8**

Base model. As our base model, we choose Fast-dLLM v2 (1.5B parameters) (Wu et al., 2025a), a state-of-the-art DLM adapted from Qwen2.5 (Yang et al., 2024) by finetuning on the LLaMA-Nemotron dataset (Bercovich et al., 2025).

Training. For RELAY adaptation we apply supervised fine-tuning to all parameters for 200 optimizer steps at effective batch size 32 on a 60,000-example mixture of filtered OpenCodeInstruct and OpenMathInstruct-2 examples with a $40/60$ code/math proportion (dataset and hardware details in Appendix A.2.3). To make Algorithm 1 compatible with state-of-the-art DLMs that combine block-autoregressive decoding with KV caching, we make two careful adaptations to the on-policy rollout. First, we run the $K=2$ relay rollout *only inside the active block* of Fast-dLLM v2's BD3-LM-style doubled (block-causal $\oplus$ block-bidirectional) attention (Arriola et al., 2025; Wu et al., 2025a), leaving previously decoded blocks frozen so their inter-block KV cache is reused unchanged across both passes. Second, within the active block we update the relay state $\boldsymbol{h}$ *only at positions that are still masked*: clean (already-committed) sub-block tokens contribute attention but their relay entries are not overwritten, which keeps within-block sub-block KV cache entries valid as the block fills in.

Evaluation. Inference follows Fast-dLLM v2's confidence-based parallel decoding (Wu et al., 2025a): within each block, the backbone applies the token-shift head so masked positions are read from the preceding token's logit row, samples are drawn with top-p filtering (p=0.95, temperature 0), and a position unmasks when the probability of its sampled token exceeds a confidence threshold τ, while the argmax masked position in each active sub-block is always unmasked so every forward makes progress. We use block length 32, sub-block length 8, and τ=0.85 for all HumanEval/MBPP numbers below (including NFE in Table 2). The *Plus* columns report HumanEval+ and MBPP+ from EvalPlus (Liu et al., 2023)—expanded unit-test suites released with the EvalPlus framework[5]—in the same *Base*/*Plus* layout used for code results in Wu et al. (2025a).

As in Sudoku, RELAY pushes the accuracy-NFE frontier here: it attains the best raw NFE among adapted methods on both HumanEval and MBPP, while also reaching the best accuracies. Notably, **on HumanEval, RELAY even surpasses the vanilla SFT accuracy at 32% less NFEs** (88.3 vs. 130.7), demonstrating that the RELAY improves both accuracy and the number of denoising steps required to reach it.

4.2.1. TRAINING MEMORY OVERHEAD

A natural concern is that BPTT through $K=2$ forward passes inflates training memory. Figure 3 profiles one micro-step on an A100 80GB. Each regime is shown with two curves: the solid trace samples live GPU memory at every transformer-layer hook, and the dashed trace is its running maximum, a high-water mark whose final value is the peak the run actually demanded. Thus a short-lived allocation can lift the dashed trace even if it is freed before the next solid-line sample. The largest such transient—and the binding peak of the whole micro-step in both regimes—is the cross-entropy backward through the vocabulary-projection head (`lm_head`), which materializes a $B \times T \times V$ fp32 grad-of-logits buffer at the start of `bwd`.

RELAY's second forward raises the live trace by ≈ 5 GiB through `fwd2`: the saved activations of forward 1 and the relay state $\boldsymbol{h}$ coexist with forward 2 to route credit through both passes (Algorithm 1). Most of that elevation is autograd intermediates rather than saved-for-backward state, and PyTorch releases it in a single step before the `lm_head` spike fires—live drops by ≈ 7 GiB for RELAY versus ≈ 2.7 GiB for vanilla, leaving the two regimes within ≈ 0.5 GiB of each other just before the spike. Adding the spike yields nearly identical peaks, 20.1 GiB for RELAY versus 21.2 GiB for vanilla SFT—in fact, RELAY's larger pre-spike drop edges its peak slightly below vanilla's. This near-tie is

[5] https://github.com/evalplus/evalplus

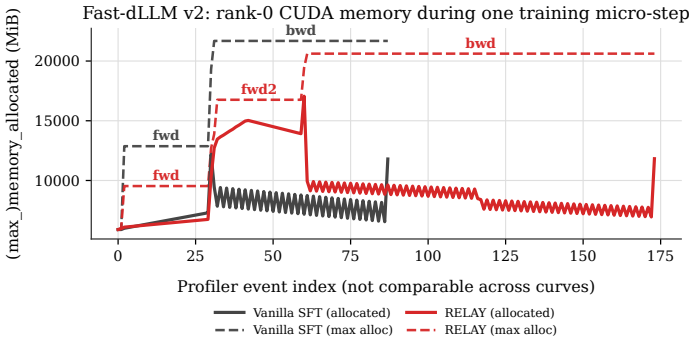


Figure 3. GPU memory during one training micro-step of Fast-dLLM v2 on an A100 80GB. Solid lines show the live GPU memory at every decoder-layer forward/backward hook. Dashed lines show the running maximum of live memory within the same micro-step (high-water mark). Phase labels (`fwd`, `fwd2`, `bwd`) mark each phase's plateau. RELAY carries higher live memory through `fwd2`, but its peak ($\approx$20.1 GiB) lands within $\approx$1 GiB of vanilla SFT's ($\approx$21.2 GiB); see main text and Appendix B for the mechanism.

structural rather than incidental: Fast-dLLM v2's vanilla SFT forward already doubles both the sequence and the batch (BD3-LM's $[\boldsymbol{x}_t \,\|\, \boldsymbol{x}_0]$ layout plus a complementary-mask copy along the batch), so each of RELAY's two forwards runs at half vanilla's per-pass batch and the two together demand memory comparable to vanilla's single doubled forward. BPTT through $K{=}2$ thus does not double peak memory in this setup (gradient checkpointing, ZeRO-3, non-fused CE)—RELAY trades vanilla's in-forward batch doubling for an explicit second pass—and we expect peak memory to stay comparable whenever the vanilla baseline already pays a doubled-batch forward and the `lm_head` backward dominates. Per-phase numbers and the profiling protocol are deferred to Appendix B.

5. Related Work

Discrete diffusion models (Austin et al., 2021), which apply the iterative denoising principles of continuous diffusion (Ho et al., 2020; Song et al., 2020) to categorical sequences, have emerged as a strong framework for language modeling. In particular, Masked Diffusion Models (MDMs) (Sahoo et al., 2024; Shi et al., 2024), which generate sequences by iterative unmasking have been shown to scaled well to larger models sizes (Nie et al., 2025; Ye et al., 2025; Wu et al., 2025b;a). The same diffusion style training that makes MDMs simple also limits what can be communicated between denoising steps: rich internal representations are collapsed into sampled tokens before the next step begins. Some recent works term the collapse of internal information as a "sampling wall" or "information island" (Jo et al., 2025; Xia et al., 2026).

To address this, several recent approaches use a continuous relaxation or augmenteded state trajectories. CADD (Zheng et al., 2025) pairs each position with a continuous diffusion process. Soft-Masked Diffusion (Hersche et al., 2025) pass output distributions or top-k predictions from the previous step back into the input to the model for the next step. CADD (Zheng et al., 2025) augments each discrete variable with a continuous variable that is trained using a continuous diffusion process in the embedding space. VADD (Xie et al., 2025), on the other hand, trains a VAE atop discrete diffusion. All these approaches rely on leveraging a continuous diffusion process to carry more information across steps even though we ultimately only care about the discrete variables. In contrast, our approach provides supervision through the discrete variables only.

MetaState (Xia et al., 2026) and Loopholing (Jo et al., 2025) introduce a continuous pathway that carries hidden state across steps and train it without relying on continuous diffusion, which is quite similar to our approach. MetaState adds a fixed-size working memory to frozen dLLMs and trains it over multi-step denoising rollouts. Loopholing, on the other hand, simply injects the hidden state from the previous step into the input of the current step, like our RELAY (sg) setting in the ablations, and trains the entire model. This allows the model to *learn to use* the hidden state for future predictions. RELAY goes one step further by training the hidden state end-to-end via BPTT which allows the model to also *learn to save* the hidden state for future predictions.

6. Discussion

Summary of results. Masked diffusion models suffer from a *hard reset* between denoising steps: the Transformer

computes rich hidden states at every position—including those still masked—but discards them at the end of each forward pass, so the only information that persists is the discrete tokens just committed. RELAY addresses this by carrying the last-layer hidden states forward as a learned relay and training it end-to-end via truncated BPTT, so the model is explicitly rewarded for writing hidden states that will be useful to future denoising steps. Empirically, the three components that constitute RELAY—a rollout-based training procedure, passing the hidden state forward across denoising steps, and training the hidden state end-to-end via BPTT—each push the performance-latency frontier on their own, and combine constructively. On Sudoku-Extreme, the full method attained the best accuracy-per-NFE point on the Pareto frontier (Figure 2); on Fast-dLLM v2 it outperformed standard supervised fine-tuning on coding tasks while reducing inference latency by up to 32%.

Limitations. RELAY introduces two computational trade-offs. First, the relay mechanism adds a small per-step overhead for reading and writing the continuous relay state, though the reduced number of forward passes needed to reach a given accuracy can still yield a net inference-latency improvement. Second, two-step BPTT during training increases live activation memory and per-step compute. In our Fast-dLLM v2 profile, however, each RELAY forward runs at half the batch of vanilla SFT's doubled forward, so this added activation stays below the peak set by the vocabulary-projection (`lm_head`) backward (Section 4.2.1, Appendix B), leaving observed peak GPU memory nearly unchanged. Overall, while RELAY requires more training time than vanilla MLM training, this training-time gap can be amortized by improvements to the inference-time accuracy-latency frontier, and narrowed with more careful engineering.

Outlook and future work. RELAY is a meaningful step towards a non-greedy, forward-thinking approach to iterative non-autoregressive generation, and there are several natural follow-up directions. The relay state gives a diffusion model a continuous substrate on which to carry intermediate computation across denoising steps; understanding what this state encodes, and whether it can be probed or steered, is a promising direction for interpreting and improving latent reasoning in MDMs. Because the relay mechanism is largely architecture- and modality-agnostic, applying it beyond text—for example, to image or molecular discrete diffusion—is also a natural next step.

7. Conclusion

We introduced RELAY to address the hard reset problem in MDMs by passing a continuous, differentiable latent state across inference steps. By training a relay channel via truncated BPTT, we demonstrated that discrete diffusion models can explicitly optimize intermediate representations for future unmasking decisions, advancing the performance-latency Pareto frontier.

Acknowledgments

DP, BR, and AM thank Michael Boratko for helpful initial discussions. DP and BR acknowledge support from IBM under IBM Research Collaboration Agreement No. W1668553 and from the National Science Foundation under grant IIS-2106391. NB acknowledges support from an NSF Graduate Research Fellowship, Quad Fellowship, and Mercor Graduate Fellowship. TGJR acknowledges support provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, the Vector Institute for Artificial Intelligence, and by the Digital Research Alliance of Canada (alliancecan.ca).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Arriola, M., Gokaslan, A., Chiu, J. T., Yang, Z., Qi, Z.-H., Han, J., Sahoo, S., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models, 2025. URL http://arxiv.org/abs/2503.09573.

Austin, J., Johnson, D. D., Ho, J., Tarlow, D., and Berg, R. v. d. Structured denoising diffusion models in discrete state-spaces. In *Neural Information Processing Systems*, pp. 17981–17993, 2021.

Ben-Hamu, H., Gat, I., Severo, D., Nolte, N., and Karrer, B. Accelerated Sampling from Masked Diffusion Models via

Entropy Bounded Unmasking, 2025.

Bercovich, A., Levy, I., Golan, I., Dabbah, M., El-Yaniv, R., Puny, O., Galil, I., Moshe, Z., Ronen, T., Nabwani, N., et al. Llama-nemotron: Efficient reasoning models. *arXiv.org*, 2025. doi: 10.48550/arXiv.2505.00949.

Campbell, A., Benton, J., Bortoli, V. D., Rainforth, T., Deligiannidis, G., and Doucet, A. A continuous time framework for discrete denoising models. In *Neural Information Processing Systems*, pp. 28266–28279. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.48550/arXiv.2205.14987. URL https://openreview.net/forum?id=DmT862YAieY.

Gatmiry, K., Saunshi, N., Reddi, S. J., Jegelka, S., and Kumar, S. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15130–15152. PMLR, 21–27 Jul 2024. doi: 10.48550/arXiv.2410.08292. URL https://proceedings.mlr.press/v235/gatmiry24b.html.

Gong, S., Li, M., Feng, J., Wu, Z., and Kong, L. Generative recursive reasoning models. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/pdf?id=Vxu6kcIjwV.

Hersche, M., Moor-Smith, S., Hofmann, T., and Rahimi, A. Soft-masked diffusion language models, 2025. URL http://arxiv.org/abs/2510.17206.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models, 2020. URL http://arxiv.org/abs/2006.11239.

Jo, M., Yoon, J., Deschenaux, J., Gulcehre, C., and Ahn, S. Loopholing discrete diffusion: Deterministic bypass of the sampling wall, 2025. URL http://arxiv.org/abs/2510.19304.

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 *International Conference on Machine Learning*, 2025. doi: 10.48550/arXiv.2502.06768. URL https://openreview.net/forum?id=DjJmre5IkP.

Kim, J., Geuter, J., Alvarez-Melis, D., Kakade, S., and Chen, S. Stop training for the worst: Progressive unmasking accelerates masked diffusion training, 2026. URL http://arxiv.org/abs/2602.10314.

Li, Z., Liu, H., Zhou, D., and Ma, T. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2402.12875.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Neural Information Processing Systems*, pp. 21558–21572. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2023. doi: 10.52202/075280-0943. URL https://openreview.net/forum?id=1qvx610Cu7.

McGuire, G., Tugemann, B., and Civario, G. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics*, 23(2):190–217, 2012. doi: 10.1080/10586458.2013.870056.

Nie, S., Zhu, F., You, C., Zhang, X., Ou, J., and Zhu, J. LLaDA: Large language diffusion with autoregressive initialization, 2025. URL http://arxiv.org/abs/2502.09992.

Patel, D., Naseem, T., Pandey, G., Sultan, M., McCallum, A., and Fernandez, R. Improved sampling from masked diffusion models with position contrastive guidance. In *NeurIPS 2025 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2025. URL https://openreview.net/forum?id=e0WmOrWbtc.

Patel, D., Maram, D. P., Chintha, S. S., Rozonoyer, B., and McCallum, A. xLM: A python package for non-autoregressive language models. In Croce, D., Leidner, J., and Moosavi, N. S. (eds.), *Proceedings of the 19th Conference of the European Chapter of the ACL (Volume 3: System Demonstrations)*, pp. 445–456, Rabat, Morocco, March 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.eacl-demo.31. URL https://aclanthology.org/2026.eacl-demo.31/.

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, 2024. URL http://arxiv.org/abs/2406.07524.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2502.17416.

Shi, J., Han, K., Wang, Z., Doucet, A., and Titsias, M. K. Simplified and generalized masked diffusion for discrete data, 2024. URL http://arxiv.org/abs/2406.04329.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations, 2020. URL http://arxiv.org/abs/2011.13456.

Vink, T. sudoku-solver: a python Sudoku solver that traces the human-style strategies it uses. https://github.com/timvink/sudoku-solver, 2024.

Wang, G., Li, J., Sun, Y., Chen, X., Liu, C.-L., Wu, Y., Lu, M., Song, S., and Abbasi-Yadkori, Y. Hierarchical reasoning model. *arXiv.org*, 2025. doi: 10.48550/arXiv.2506.21734.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Wu, C., Zhang, H., Xue, S., Diao, S., Fu, Y., Liu, Z., Molchanov, P., Luo, P., Han, S., and Xie, E. Fast-dllm v2: Efficient block-diffusion llm, 2025a. URL https://arxiv.org/abs/2509.26328.

Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv.org*, 2025b. doi: 10.48550/arXiv.2505.22618.

Xia, K., Li, M., Wei, L., Du, Z., Yuan, X., Jin, Q., and Lee, W. MetaState: Persistent working memory for discrete diffusion language models, 2026. URL http://arxiv.org/abs/2603.01331.

Xie, T., Xue, S., Feng, Z., Hu, T., Sun, J., Li, Z., and Zhang, C. Variational autoencoding discrete diffusion with enhanced dimensional correlations modeling, 2025. URL http://arxiv.org/abs/2505.17384.

Yang, Q. A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Dong, G., et al. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models, 2025. URL https://arxiv.org/abs/2508.15487.

Zheng, H., Gong, S., Zhang, R., Chen, T., Gu, J., Zhou, M., Jaitly, N., and Zhang, Y. Continuously augmented discrete diffusion model for categorical generative modeling, 2025. URL http://arxiv.org/abs/2510.01329.

Zheng, K., Chen, Y., Mao, H., Liu, M.-Y., Zhu, J., and Zhang, Q. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. In *International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2409.02908. URL https://openreview.net/forum?id=CTC7CmirNr.

A. Experimental Details

A.1. Sudoku

A.1.1. Dataset

Sudoku Extreme. We train and evaluate on our derived dataset built on top of Sudoku-Extreme (Gong et al., 2025; Wang et al., 2025) by running the solver of Vink (2024) over every puzzle.[6] The base dataset consists of 9×9 Sudoku puzzles with 17 given clues, the minimum number compatible with a uniquely solvable puzzle (McGuire et al., 2012). Each puzzle is represented as a flat sequence of length L=81 over a vocabulary of $|\mathcal{V}|$=11 task tokens: digits $\{1, \ldots, 9\}$, a blank/zero token for unfilled cells, and a mask token. The clue positions are treated as fixed and are not modified during inference; the remaining 64 positions are mutable. Our derived version augments each puzzle with:

- `trajectory`: step-by-step board states from question to solution
- `num_steps`: number of solver calls to reach the solution
- `strategies_used`: set of human-like deduction strategies invoked (used by the deduction-only cohort below)

We use the training split (3,831,994 puzzles) and evaluate on the test split (422,786 puzzles), and validate on the first 100 batches at batch size 512 (51,200 puzzles) per checkpoint.

Deduction-only cohort. For the qualitative legality analysis of Section 4.1 we use the `strategies_used` field described above to filter puzzles. Since the studied architectures cannot perform recursive search, we keep only puzzles whose solver trace contains *Advanced* (Naked Pair, Hidden Pair, Naked Triple, Hidden Triple, Naked Quad, Hidden Quad) or *Master* (X-Wing, Swordfish, Jellyfish, Forcing Chain) strategies and never falls back on recursive backtracking. The resulting cohort contains 2,000 test puzzles (1,933 Advanced + 67 Master).

Evaluation protocol for Table 1. Each cell of Table 1 aggregates the first N=2000 puzzles from the Hugging Face test split in dataset order; for the deduction-only cohort, we keep the first 2,000 examples whose solver trace uses Advanced or Master strategies without recursive backtracking.

A.1.2. Model Architecture

The backbone for all Sudoku experiments (Table 1) is a shallow rotary Transformer:

- **Depth / width:** L=4 layers, hidden dimension d_{model}=384, feedforward width $4d_{\text{model}}$=1536
- **Attention:** H=6 heads (head dimension d_{model}/H=64), rotary positional embeddings (rotary width 64)
- **MLP:** ReLU nonlinearities, dropout 0.1
- **Vocabulary:** digits $\{0, \ldots, 9\}$ plus special tokens, $|\mathcal{V}|$=17

The Relay variant adds a differentiable carry channel following Jo et al. (2025). At each inference step the relay tensor h_t from the previous step is normalized by an affine LayerNorm (ε_{LN}=10^{-5}), yielding $\delta_t = \text{LN}_{\text{relay}}(h_t)$, and injected additively into the residual stream before layer zero: $x \leftarrow \text{Embed}(x_t) + \delta_t$. The outgoing relay state h_{t+1} is read from the final transformer block, while logits are always produced from the same terminal hidden states. We initialize LN_{relay} with PyTorch defaults ($\boldsymbol{\gamma}_{\text{relay}} \leftarrow \mathbf{1}$, $\boldsymbol{\beta}_{\text{relay}} \leftarrow \mathbf{0}$). We implement all the models using the xLM (Patel et al., 2026) package, which provides a unified interface for training and inference of non-autoregressive language models making the ablations and experiments easy to reproduce.

Parameter counts (with and without weight tying) are:

- **Baseline** (MLM / rollout-buffer only): 7,105,536 untied; 7,099,008 tied
- **Relay**: 7,106,304 untied; 7,099,776 tied

A.1.3. Training Hyperparameters

- **Batch size:** 512 (single GPU, bf16 mixed precision)

[6] Derived dataset: https://huggingface.co/datasets/brozonoyer/sapientinc-sudoku-extreme-timvink-sudoku-solver. Solver code: https://github.com/timvink/sudoku-solver.

- **Optimizer:** AdamW, learning rate 5×10^{-4}, weight decay 10^{-2}
- **LR schedule:** constant with 2,000-step linear warmup, no decay thereafter
- **Gradient clipping:** global Frobenius norm 0.5
- **BPTT unroll horizon:** $K{=}2$ steps (RELAY runs only)
- **Confidence threshold:** $\tau{=}0.15$ (maximum softmax probability), perturbed by $\mathcal{N}(0, 0.1^2)$ during training and fixed at inference
- **Validation:** every 5,000 steps on 100 batches; threshold sweep $\tau \in \{0.05, 0.10, \ldots, 0.25\}$
- **Total steps:** 300,000; results reported in Table 1

A.2. Fast-dLLM v2

A.2.1. DATASET

OpenCode/OpenMath c40m60 mixture for Fast-dLLM v2 adaption. For Fast-dLLM v2 adaptation, we use a 60k-example supervised fine-tuning mixture from `nvidia/OpenCodeInstruct`[7] and `nvidia/OpenMathInstruct-2`[8], with 24k code examples and 36k math examples. We filter for high-quality prompt–answer pairs, remove held-out evaluation contamination, format examples as one-turn conversations, and cap sequences at 2048 tokens.

A.2.2. MODEL ARCHITECTURE

Unlike standard MDMs, which denoise the entire token sequence globally, Fast-dLLM v2 models (Wu et al., 2025a) a block-wise Markov process. By partitioning the sequence into blocks of size D, it targets the local conditional distribution $p_\theta(x^b | x_t^b, x_0^{<b})$. This localizes the diffusion process while anchoring it to an autoregressive prefix, successfully bypassing the immense pretraining costs associated with full-attention MDMs.

The core architectural shift lies in its attention topology. Fast-dLLM v2 concatenates the noised x_t and clean x_0 sequences into a $2L$-length tensor, governed by a full attention mask $\mathcal{M}_{full} \in \{0,1\}^{2L \times 2L}$ (Arriola et al., 2025). This mask explicitly splits into three distinct functional roles:

- $\mathcal{M}_{BD}$: Enables intra-block bidirectional attention within each block.
- $\mathcal{M}_{OBC}$: Allows the noised block to attend to the completely denoised, clean prefix $x_0^{<b}$.
- $\mathcal{M}_{BC}$: Enforces standard left-to-right causality among the clean tokens.

The $2L$ concatenation lets the noised and clean views be processed in a single forward pass. On top of this, a complementary masking strategy trains on both a sampled mask m and its complement $\bar{m} = 1 - m$, so that every token in the input contributes supervision rather than only those masked under m.

At inference, this topology enables hierarchical Key-Value caching—a major advantage over standard MDMs, which typically require full-sequence recomputation at every denoising step. Completely denoised blocks $x_0^{<b}$ are saved as read-only context, while a DualCache handles prefix and suffix activations within the active, partially noised block x_t^b.

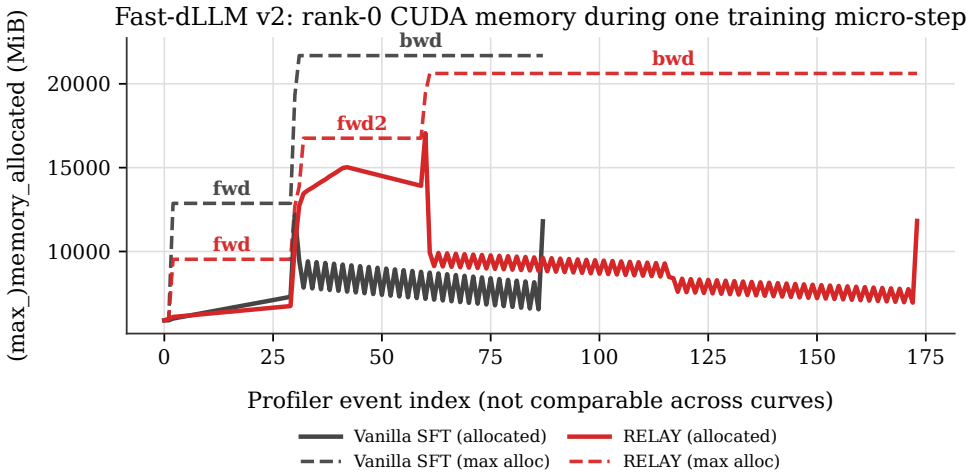
A.2.3. FAST-DLLM V2 TRAINING HARDWARE AND PARALLELISM

All adaptation runs use DeepSpeed ZeRO-3 with bf16 mixed precision and gradient checkpointing on two NVIDIA A100 80GB GPUs, with per-device batch size 2 and gradient accumulation 16 (effective batch size 32). For RELAY adaptation, $\mathrm{LN}_{\mathrm{relay}}$ uses zero-initialized $\boldsymbol{\gamma}_{\mathrm{relay}}$ (with $\boldsymbol{\beta}_{\mathrm{relay}}{=}0$), so early forward passes approximate an identity relay until training updates $\boldsymbol{\gamma}_{\mathrm{relay}}$ (Wu et al., 2025a).

[7] https://huggingface.co/datasets/nvidia/OpenCodeInstruct
[8] https://huggingface.co/datasets/nvidia/OpenMathInstruct-2

B. Fast-dLLM v2 memory profiling

This section gives the protocol and per-phase numbers behind Figure 3, repeated below.



Setup. We profile a single training micro-step of Fast-dLLM v2 on the OpenCode/OpenMath `c40m60` mixture under the same hardware and parallelism as the main runs (Appendix A.2.3): two A100 80GB GPUs, DeepSpeed ZeRO-3, bf16, and gradient checkpointing, with sequence length 2048 and per-device batch size 2. Production runs use gradient accumulation 16; profiling forces accumulation to 1 and replaces `optimizer.step` with a no-op so that the recorded peak is attributable to a single forward/backward pair rather than to optimizer-state allocation.

Instrumentation. On every decoder-layer forward and backward hook we log `memory_allocated` and `max_memory_allocated` from `torch.cuda`—the solid and dashed traces in Figure 3, respectively; we call `reset_peak_memory_stats()` once at the start of the profiled micro-step so the dashed series is a within-step high-water mark rather than a long-run accumulator. All measurements are taken in eager mode with `torch.compile` and FlashAttention 2 disabled, so steps in the dashed roof correspond directly to discrete kernel-level allocations. Phase labels (`fwd`, `fwd2`, `bwd`) are placed at each phase's plateau in the dashed series. Horizontal axes are profiler event indices (88 for vanilla SFT, 174 for RELAY) and are not directly comparable across the two curves.

Per-phase peaks. The dashed all-time peaks settle at 20,618 MiB ($\approx$ 20.1 GiB) for RELAY versus 21,683 MiB ($\approx$ 21.2 GiB) for vanilla SFT. In both regimes the peak is set at the start of `bwd`, when the cross-entropy backward through `lm_head` transiently allocates a $B \times T \times V$ gradient-of-logits buffer that HuggingFace materializes in fp32 for numerical stability, on top of the bf16 logits tensor it is differentiating. The effective batch B is not the same in the two regimes: Fast-dLLM v2's vanilla SFT forward applies BD3-LM's complementary-mask doubling along the batch dimension—in addition to the $[\boldsymbol{x}_t \,\|\, \boldsymbol{x}_0]$ doubling along the sequence dimension that both regimes share—whereas each of RELAY's two rollout forwards runs at the undoubled batch. At per-device batch size 2 (with $T{=}2048$, $V{=}151936$), the fp32 buffer is therefore $[4, 2048, 151936] \times 4\,\text{B} \approx 4.6$ GiB for vanilla but only half that, ≈ 2.3 GiB, for RELAY. RELAY offsets this smaller spike by retaining *two* forwards' activations: its second forward elevates the live trace by $\approx$5 GiB through `fwd2`—saved activations of forward 1 plus the relay state $\boldsymbol{h}$ must coexist with forward 2 to provide credit through both passes (Algorithm 1)—reaching 17,057 MiB at `relay_fwd2_end`, still below the 20,618 MiB peak that the CE backward sets one event later, so the live plateau through `fwd2` does not become the binding peak. These two asymmetries—vanilla's larger CE buffer versus RELAY's heavier retained activations—roughly cancel, leaving the peaks within $\approx$1 GiB. The residual gap in RELAY's favor is structural rather than allocator noise: more autograd intermediates from forward 1 are released by the time the CE backward fires than vanilla releases by the analogous event (Δlive $= -7{,}119$ MiB across this transition for RELAY vs. $-2{,}727$ MiB for vanilla). Rank-1 traces reproduce both peaks within <0.1 MiB, so Figure 3 shows only rank 0.