MetaState: Persistent Working Memory Enhances Reasoning in Discrete Diffusion Language Models

Kejing Xia[1], **Mingzhe Li**[2], **Lixuan Wei**[3], **Zhenbang Du**[1], **Xiangchi Yuan**[1], **Dachuan Shi**[1], **Qirui Jin**[1], **Wenke Lee**[1]
[1]Georgia Institute of Technology [2]University of Massachusetts Amherst
[3]Harvard University

Abstract

Discrete diffusion language models (dLLMs) generate text by iteratively denoising a masked sequence. However, standard dLLMs condition each denoising step solely on the current hard-masked sequence, while intermediate continuous representations are discarded after sampling and remasking. We term this bottleneck the **Information Island** issue: continuous information remains isolated within individual denoising steps and fails to propagate across the trajectory. This bottleneck is especially harmful for reasoning, which requires intermediate reasoning state to be preserved and updated across many denoising steps. To address this limitation, we introduce **MetaState**, a lightweight recurrent augmentation that equips a frozen dLLM backbone with persistent, fixed-size working memory. MetaState comprises three modules with a shared time conditioner: a cross-attention **Mixer** that reads backbone activations into memory slots, a GRU-style **Updater** that integrates information across steps, and a cross-attention **Injector** that writes the updated memory back into the backbone. We train these modules with a dedicated K-step unrolling pipeline to learn multi-step dynamics. MetaState adds only $\sim$0.6% trainable parameters while keeping the backbone frozen, and consistently improves reasoning performance over frozen baselines on mathematical reasoning and code generation benchmarks, with an average gain of 4.5 percentage points across all evaluations. Our code is available at `https://github.com/Les1a/MetaState`.

1 Introduction

Autoregressive (AR) language models factorize the joint distribution over sequences into a product of conditional probabilities, producing one token per forward pass (Radford et al., 2018; 2019; Brown et al., 2020). Although this paradigm underlies many recent foundation models, the left-to-right causal structure prevents parallel decoding and limits the use of bidirectional context. Discrete diffusion language models (dLLMs) have recently emerged as a non-autoregressive alternative (Li et al., 2025; Sahoo et al., 2024; Gong et al., 2024). Starting from a fully corrupted sequence, dLLMs iteratively denoise to recover clean text and update arbitrary positions using bidirectional attention (Austin et al., 2021a). When scaled to billions of parameters, dLLMs achieve quality comparable to that of autoregressive models while retaining the advan-

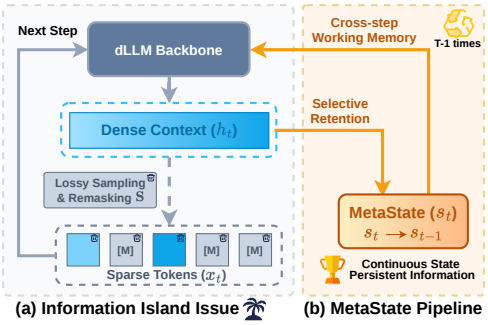

Figure 1: The Information Island issue in discrete diffusion: sampling and remasking compress continuous hidden activations into discrete tokens, imposing a lossy bottleneck between denoising steps. MetaState addresses this issue by maintaining a persistent state across steps.

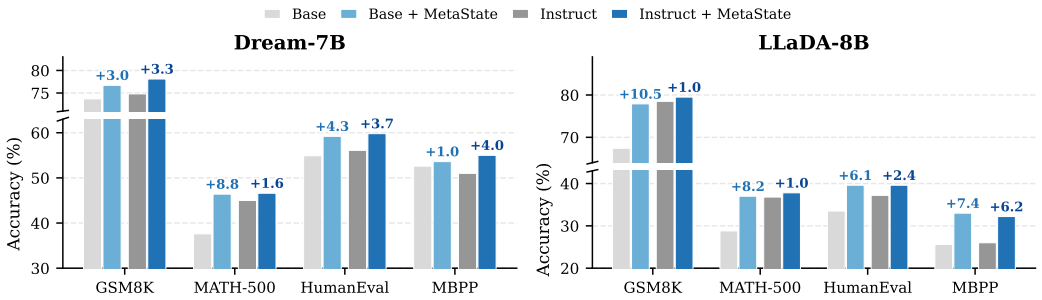

Figure 2: Performance comparison between MetaState and frozen baselines on reasoning benchmarks for LLaDA-8B and Dream-7B in both Instruct and Base versions.

tages of decoding parallelism and generation flexibility, as demonstrated by the LLaDA series (Nie et al., 2025; Zhu et al., 2025a; Bie et al., 2025) and Dream (Ye et al., 2025).

Current standard dLLMs nevertheless share a limitation that we term the Information Island issue. In the diffusion formulation, the inter-step process is Markovian in the discrete sequence state: each transition conditions on the current masked tokens $\mathbf{x}_t$, while the continuous hidden representation $\mathbf{h}_t$ computed at that step is not carried forward explicitly,

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \quad \mathbf{x}_{t-1} = \mathcal{S}(\mathbf{h}_t)\,. \tag{1}$$

At each denoising step t, the model computes a high-dimensional hidden representation $\mathbf{h}_t$ that encodes substantially richer information than the discrete tokens passed to the next step. Beyond token-level predictive semantics, $\mathbf{h}_t$ also captures long-range dependencies and global sequence structure information. However, the sampling-and-remasking interface $\mathcal{S}$ maps this rich continuous state to discrete token identities and sparse remasking indicators. This transition discards the continuous information in $\mathbf{h}_t$ and compresses each step's computation into a sparse discrete sequence. As a result, the next denoising step receives only a highly lossy representation of the information computed at the previous step. We refer to this repeated cross-step information loss as the Information Island issue.

This bottleneck arises at every transition along the denoising trajectory, as illustrated in Fig. 1. In the diffusion process, early high-noise steps often establish coarse global structure, while later low-noise steps refine local details and enforce fine-grained constraints. However, useful inferences made at one step must be reconstructed from the sparse token sequence at later steps. This repeated reconstruction can introduce cross-step drift: intermediate information may be weakened, overwritten, or inconsistently re-derived as denoising proceeds. Such drift is especially harmful for reasoning, where success depends on preserving intermediate computations in multi-step mathematical reasoning and maintaining global program constraints such as variable scope and control flow across long denoising trajectories.

To address this limitation, we propose MetaState, a lightweight recurrent augmentation that equips a frozen dLLM backbone with persistent working memory across denoising steps. Motivated by evidence that working memory capacity is an important factor in language model reasoning (Zhang et al., 2024), MetaState maintains a compact set of continuous memory slots that persist across denoising steps, thereby adding a parallel information path alongside the standard discrete denoising path. Concretely, three lightweight modules form a recurrent loop around the frozen backbone: a Mixer reads backbone activations into the memory slots, an Updater integrates newly extracted information through gated recurrence, and an Injector writes the updated state back into the backbone's input embeddings for the next step. A shared time conditioner coordinates all three modules. To train this recurrent memory to retain and update information across steps, we further introduce a dedicated K-step unrolling procedure that backpropagates through the denoising trajectory (§4.2). In summary, our contributions are as follows:

1. We identify the Information Island issue in discrete diffusion language models, a representational bottleneck in which rich hidden activations are compressed into

sparse and discrete tokens at every denoising step, and analyze why this issue is especially harmful for multi-step reasoning.

2. We propose **MetaState**, a backbone-agnostic recurrent augmentation that maintains constant-size persistent working memory throughout the denoising process, along with a K-step unrolling training procedure that enables gradient flow through the multi-step state trajectory.
3. We validate MetaState on two distinct dLLM backbones, LLaDA-8B (Nie et al., 2025) and Dream-7B (Ye et al., 2025), over standard mathematical reasoning and code generation benchmarks (GSM8K, MATH-500, HumanEval, and MBPP), achieving an average improvement of 4.5 percentage points at negligible parameter cost.

2 Related Work

2.1 Discrete Diffusion LLMs

Discrete diffusion LLMs formulate text generation as a non-autoregressive process by adapting diffusion dynamics to discrete token spaces. D3PM (Austin et al., 2021a) formalized this approach using discrete transition matrices. Subsequent models largely adopt the masked diffusion paradigm: MDLM (Sahoo et al., 2024) derives a variational lower bound and SEDD (Lou et al., 2023) introduces a score entropy objective, enabling masked dLLMs to reach likelihood and perplexity levels comparable to those of autoregressive (AR) models. At the billion-parameter scale, LLaDA series (Nie et al., 2025; Zhu et al., 2025a;b) and Dream (Ye et al., 2025) demonstrate that dLLMs can match AR quality while enabling parallel decoding and bidirectional attention. Semi-AR methods like BD3-LMs (Arriola et al., 2025) and SDAR (Cheng et al., 2025) combine inter-block autoregression with intra-block parallel diffusion. For cache integration, dLLM-Cache (Liu et al., 2025) targets dual redundancy in prompts and responses via adaptive caching, while Fast-dLLM (Wu et al., 2025) leverages activation similarity for block-wise KV-cache reuse.

2.2 Continuous Diffusion and Latent Reasoning

To fully use continuous context, prior work proposes modifying either the diffusion kernel or the decoding interface. CADD and CANDI (Zheng et al., 2025; Pynadath et al., 2025) couple discrete masking with continuous diffusion through hybrid kernels, changing the forward and reverse diffusion formulation and training the backbone to operate under that process. MetaState targets a different level: it keeps the original discrete diffusion backbone frozen and adds an external recurrent state at the sampling-and-remasking boundary. Other decoding-side methods such as LRD and RCD (Zhu et al., 2025c; Hu et al., 2026) replace hard token sampling with probability mixtures, whereas MetaState leaves the discrete path intact and adds a parallel cross-step memory path. DCoLT (Huang et al., 2025) designs a trajectory-level reasoning policy. In continuous image diffusion, Recurrent Interface Networks (Jabri et al., 2022) maintain persistent latent tokens across denoising steps, and Diffusion Forcing (Chen et al., 2024) couples an RNN with the diffusion process. In AR modeling, several latent reasoning methods propagate continuous latent representations to bypass discrete decoding bottlenecks, including Coconut, CODI, Soft Thinking and SwiReasoning (Hao et al., 2024; Shen et al., 2025; Zhang et al., 2025; Shi et al., 2025). Building on this line of work, LaDiR and STAR-LDM (Kang et al., 2025; Lovelace et al., 2026) use latent diffusion for trajectory planning. However, these techniques only apply to sequential generation and do not transfer to the dLLM paradigm. As a result, dLLMs still lack a frozen-backbone mechanism that maintains continuous memory across diffusion steps to address the Information Island issue.

3 Preliminaries

We consider the masked diffusion paradigm (Austin et al., 2021a; Ou et al., 2024), which defines a forward masking process that progressively replaces tokens with a special [MASK] token $\mathbf{M}$ and a reverse unmasking process that recovers the original sequence.

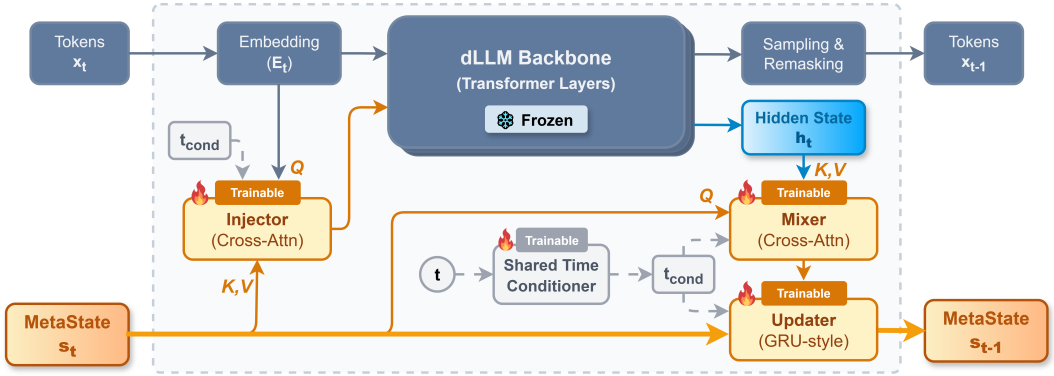

Figure 3: Overview of the MetaState architecture. The three modules (Injector, Mixer, Updater) and the shared time conditioner form a recurrent loop around the frozen backbone, propagating a persistent state across denoising steps.

Forward process. Given a clean sequence $\mathbf{x}_0 = (x_0^{(1)}, \ldots, x_0^{(N)})$ over vocabulary $\mathcal{V}$, the forward process produces a noisy sequence $\mathbf{x}_t$ by independently masking each token with probability $1 - \alpha_t$, where α_t denotes the token retention probability from the discrete diffusion convention (Shi et al., 2024; Nie et al., 2025). In continuous time $t \in (0, 1]$, the schedule decreases from $\alpha_0 = 1$ (fully clean) to $\alpha_1 \approx 0$ (fully masked). Each token is kept unchanged with probability α_t, and is replaced by the mask token $\mathbf{M}$ with probability $1 - \alpha_t$.

Reverse process. For any noise level $t \in (0, 1]$, the model p_θ predicts the clean token at each masked position i where $x_t^{(i)} = \mathbf{M}$, yielding the conditional distribution $p_\theta(x_0^{(i)} \mid \mathbf{x}_t)$.

Training objective. The training loss is the expected cross-entropy over masked positions:

$$\mathcal{L}_{\text{MDLM}} = \mathbb{E}_{t \sim \mathcal{U}(0,1],\, \mathbf{x}_0,\, \mathbf{x}_t \sim q(\mathbf{x}_t \mid \mathbf{x}_0)} \left[\frac{1}{t} \sum_{i:\, x_t^{(i)} = \mathbf{M}} -\log p_\theta(x_0^{(i)} \mid \mathbf{x}_t) \right].$$

Inference. Starting from a fully masked sequence, the time is discretized into T steps. At each discrete step t, the model samples clean tokens and selectively remasks a subset of positions according to prediction confidence, yielding a progressively cleaner sequence $\mathbf{x}_{t-1}$.

Information Island issue. As discussed in §1, this formulation gives rise to the Information Island issue: the sampling and remasking operator discards continuous hidden activations at each step (see Appendix A.1 for a detailed analysis).

4 Method

4.1 MetaState Overview

To resolve the Information Island issue, MetaState introduces a continuous memory that persists across the discrete interface steps. This memory is maintained by three lightweight modules, the *Injector*, *Mixer*, and *Updater*, coordinated by a shared time conditioner (Figure 3). All modules operate in bottleneck dimensions with a fixed slot count independent of sequence length. The complete pipeline is given in Algorithm 1 (Appendix).

Because the discrete interface between successive denoising steps discards all intermediate representations, each step has no access to the processing history accumulated by prior steps. To bridge this gap, we augment the denoising process with a persistent state $\mathbf{s}_t \in \mathbb{R}^{M \times D_s}$, organized as M fixed memory slots of dimension D_s. This fixed-size design is critical: it ensures that the memory overhead does not grow with the number of tokens, and it encourages the network to learn a compact representation of the generation trajectory rather than simply caching raw activations. The augmented transition becomes:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{s}_t), \qquad \mathbf{s}_{t-1} = g_\theta(\mathbf{s}_t, \mathbf{h}_t, t),$$

where g_θ denotes the state-update function realized by the Mixer and Updater. The state requires $\mathcal{O}(MD_s)$ storage and does not scale with the sequence length N. In addition, each cross-attention operation is performed with a fixed number of memory slots in a bottleneck dimension. Thus the overall overhead is dominated by the cost of the frozen backbone.

At each denoising step, the Injector first writes the current state (when the state exists) into the backbone's input embeddings using the conditioning feature from the end of the previous step. After the backbone forward pass, the Mixer reads the final-layer activations into the memory slots and simultaneously computes a content summary $\bar{\mathbf{h}}_t$, which is combined with the current timestep t to form the updated conditioning feature $\mathbf{t}_{\text{cond}}$. The Updater then integrates this new context with the existing state via gated recurrence, completing the recurrent loop.

4.1.1 Shared Time Conditioner and AdaRMSNorm

All three MetaState modules require a shared conditioning signal that captures both the diffusion timestep and the current content of the sequence. A pure timestep embedding is insufficient because the optimal modulation depends on which tokens have already been revealed at each step. We therefore construct a shared time conditioner from a sinusoidal embedding (Vaswani et al., 2017) followed by an MLP, with a zero-gated content residual:

$$\mathbf{t}_{\text{cond}} = \text{MLP}(\text{sinusoidal}(t)) \; + \; \boldsymbol{\alpha}_g \odot W_c\big(\text{RMSNorm}(\bar{\mathbf{h}}_t)\big) \in \mathbb{R}^{d_c},$$

where d_c is the output dimension of the time conditioner, $\bar{\mathbf{h}}_t \in \mathbb{R}^{d_m}$ is the mean-pooled content summary derived from down-projected backbone hidden states (computed by the Mixer before cross-attention, §4.1.2), W_c projects to time dimension d_c, and $\boldsymbol{\alpha}_g \in \mathbb{R}^{d_c}$ is a learnable per-channel gate. The gate is zero-initialized to ensure that the conditioner begins as a pure timestep function and only gradually incorporates content-aware modulation as training proceeds, avoiding unstable early-stage interactions.

The conditioning feature $\mathbf{t}_{\text{cond}}$ is consumed by the normalization layer through AdaRMSNorm, an adaptive variant of RMSNorm (Zhang & Sennrich, 2019). Each AdaRMSNorm layer includes a modulation projection W_{mod} that produces per-channel scale and shift parameters from the conditioning:

$$[\boldsymbol{\gamma}, \boldsymbol{\beta}] = W_{\text{mod}}(\mathbf{t}_{\text{cond}}), \quad \mathcal{N}(\mathbf{x}, t) = (1 + \boldsymbol{\gamma}) \odot \text{RMSNorm}(\mathbf{x}) + \boldsymbol{\beta}.$$

At initialization, AdaRMSNorm reduces to standard RMSNorm, preserving the pretrained behavior of any layer it wraps. We also define a zero-bridge variant $\mathcal{N}_0(\mathbf{x}, t) = \boldsymbol{\gamma} \odot \text{RMSNorm}(\mathbf{x}) + \boldsymbol{\beta}$, which outputs nearly $\mathbf{0}$ at initialization. This serves as a zero bridge in the Injector (§4.1.4), ensuring that the augmented model begins as the unmodified backbone.

4.1.2 MetaState Mixer

The Mixer is designed to convert the variable-length backbone activations $\mathbf{h}_t \in \mathbb{R}^{N \times D}$ into a fixed-size representation of M slots, where each slot should capture distinct aspects of the computation rather than collapse into a single pooled summary. We therefore use cross-attention with the state slots as queries, letting each slot selectively read the most informative tokens. To keep the module lightweight, the cross-attention operates in a d_m-dimensional bottleneck, and an up-projection recovers the full state dimension D_s. Before entering the bottleneck, a slot self-attention layer Attn_{GQA} with plain RMSNorm enables inter-slot coordination in the full D_s space, encouraging different slots to specialize and avoid redundant reads. Both the state and the hidden representation are then down-projected and time-conditioned within the bottleneck:

$$\mathbf{s}_t^b = \mathcal{N}(W_{\downarrow}^s \, \mathbf{s}_t, \; t) \in \mathbb{R}^{M \times d_m}, \quad \mathbf{h}_t^b = \mathcal{N}(W_{\downarrow}^h \, \mathbf{h}_t, \; t) \in \mathbb{R}^{N \times d_m}.$$

Before cross-attention, the Mixer computes a content summary $\bar{\mathbf{h}}_t = \text{MeanPool}(W_{\downarrow}^h \mathbf{h}_t)$ and passes it to the time conditioner (§4.1.1), so that subsequent normalization layers can adapt to the current sequence content. Cross-attention is then computed with $\mathbf{s}_t^b$ as queries and $\mathbf{h}_t^b$ as keys/values, yielding $\mathbf{a}_t^b = \text{CrossAttn}_{\text{GQA}}(\mathbf{s}_t^b, \mathbf{h}_t^b) \in \mathbb{R}^{M \times d_m}$. An FFN with AdaRMSNorm is then applied, followed by an up-projection to yield the Mixer output $\mathbf{c}_t \in \mathbb{R}^{M \times D_s}$.

4.1.3 *MetaState Updater*

The Updater must retain information accumulated over earlier denoising steps while incorporating new context from the current step. A time-conditioned GRU (Dey & Salem, 2017) addresses this trade-off directly: its learned update gate provides a per-dimension interpolation between the existing state and a candidate update. Given the current state $\mathbf{s}_t \in \mathbb{R}^{M \times D_s}$ and the Mixer output $\mathbf{c}_t \in \mathbb{R}^{M \times D_s}$, both inputs are first normalized with time conditioning. A learnable slot identity embedding $\mathbf{e}_{\text{slot}} \in \mathbb{R}^{M \times D_s}$ is added to the state before normalization so that each slot can learn distinct retention and update behaviors. The complete update rule is:

$$\begin{aligned}
&\bar{\mathbf{s}}_t = \mathcal{N}(\mathbf{s}_t + \mathbf{e}_{\text{slot}},\ t), \quad \bar{\mathbf{c}}_t = \mathcal{N}(\mathbf{c}_t,\ t), \\
&\mathbf{z}_t, \mathbf{r}_t = \sigma\big(W_g([\bar{\mathbf{s}}_t \,\|\, \bar{\mathbf{c}}_t])\big)\,, \quad \tilde{\mathbf{s}}_t = \tanh(W_{\tilde{s}}([\mathbf{r}_t \odot \bar{\mathbf{s}}_t \,\|\, \bar{\mathbf{c}}_t]))\,, \\
&\mathbf{s}_{t-1} = (1 - \mathbf{z}_t) \odot \mathbf{s}_t + \mathbf{z}_t \odot \tilde{\mathbf{s}}_t.
\end{aligned}$$

The update gate $\mathbf{z}_t$ controls how much of each dimension is overwritten, while the reset gate $\mathbf{r}_t$ determines how much of the previous state influences the candidate $\tilde{\mathbf{s}}_t$. The final interpolation ensures a smooth transition between retaining old information and integrating new context. Time modulation enters the GRU exclusively through the AdaRMSNorm layers on $\bar{\mathbf{s}}_t$ and $\bar{\mathbf{c}}_t$, which effectively provides timestep-dependent information.

4.1.4 *MetaState Injector*

The Injector should write the persistent state back into the backbone without disrupting its pretrained capabilities. We therefore realize it as an additive modulation of the input embeddings. Given embeddings $\mathbf{e}_t \in \mathbb{R}^{N \times D}$, we first down-project to a d_b-dimensional bottleneck: $\mathbf{x}_t^b = W_{\downarrow}^e \mathbf{e}_t \in \mathbb{R}^{N \times d_b}$. A self-attention layer Attn_{GQA} with sinusoidal positional encoding and plain RMSNorm then enriches these representations with explicit positional context, enabling the subsequent cross-attention to route slot information to the appropriate sequence positions. The state is down-projected and normalized via $\mathbf{s}_t^b = \mathcal{N}(W_{\downarrow}^s \mathbf{s}_t, t)$, and cross-attention is computed with $\mathbf{x}_t^b$ as queries and $\mathbf{s}_t^b$ as keys/values. The output is added as a residual to $\mathbf{x}_t^b$. A subsequent FFN refines the fused representation, and a zero-bridge layer ($\mathcal{N}_0$) up-projects the result back to the full embedding dimension:

$$\mathbf{x}_t^b \leftarrow \mathbf{x}_t^b + \text{FFN}\Big(\mathcal{N}(\mathbf{x}_t^b, t)\Big)\,, \quad \boldsymbol{\delta}_t = W_{\uparrow}\,\mathcal{N}_0(\mathbf{x}_t^b, t), \quad \tilde{\mathbf{e}}_t = \mathbf{e}_t + \boldsymbol{\delta}_t.$$

Because $\mathcal{N}_0$ outputs near-zero at initialization, the modulation $\boldsymbol{\delta}_t$ vanishes at the start of training, ensuring that the augmented model begins as the unmodified backbone while still allowing all Injector parameters to receive gradients. The final modified embeddings $\tilde{\mathbf{e}}_t$ are fed back into the frozen backbone for the current denoising step.

4.2 Training: K-Step Iterative Unrolling

Standard masked diffusion training samples a single random timestep t per example and optimizes a single-step denoising objective. This approach is inadequate for MetaState, whose persistent state forms a recurrent chain across the full denoising trajectory (§4.1): the modules must learn what information to write into the state, what to retain across steps, and how to adapt the gating behavior across the denoising trajectory. We therefore adopt a multi-step unrolling pipeline with backpropagation through time (BPTT) (Werbos, 2002; Gers et al., 2002) along the state trajectory.

Training Pipeline. Starting from a fully masked input, a warmup forward pass initializes the recurrent state without computing loss. A complete reveal trajectory is then pre-sampled to partition all N_m maskable positions into K batches. At each unrolling step, the model first predicts on the current masked input, and a batch of positions is then revealed via teacher forcing. Let $\mathcal{M} \subseteq \{1, \ldots, N\}$ denote the set of maskable positions, and let $N_m = |\mathcal{M}|$ denote the total number of such positions. At unrolling step k, let $\mathcal{M}_k$ denote the set of positions that remain masked, with $\mathcal{M}_1 = \mathcal{M}$ and $\mathcal{M}_k \subset \mathcal{M}_{k-1}$. Let $\mathcal{R}_k \subset \mathcal{M}_k$ denote the set of n_k

Table 1: Main results on 4 different benchmarks (generation length 256, block size 32, dual cache). MATH = MATH-500, HE = HumanEval. Δ denotes improvement over the corresponding baseline. **Bold** marks the best result per column within each backbone group.

	Dream Backbone (7B)				*LLaDA Backbone (8B)*				
Model	GSM8K	MATH	HE	MBPP	GSM8K	MATH	HE	MBPP	**Avg.**
Base	73.7	37.6	54.9	52.6	67.4	28.8	33.5	25.6	46.8
+ **MetaState**	**76.7**	**46.4**	**59.2**	**53.6**	**77.9**	**37.0**	**39.6**	**33.0**	**52.9**
Δ vs. Base	+3.0	+8.8	+4.3	+1.0	+10.5	+8.2	+6.1	+7.4	+6.2
Instruct	74.8	45.0	56.1	51.0	78.5	36.8	37.2	26.0	50.7
+ **MetaState**	**78.1**	**46.6**	**59.8**	**55.0**	**79.5**	**37.8**	**39.6**	**32.2**	**53.6**
Δ vs. Instruct	+3.3	+1.6	+3.7	+4.0	+1.0	+1.0	+2.4	+6.2	+2.9

positions revealed at step k. The sequence $\mathbf{x}^{(k+1)}$ is then obtained by revealing the ground-truth tokens at the positions in $\mathcal{R}_k$, and the masked set is updated as $\mathcal{M}_{k+1} = \mathcal{M}_k \setminus \mathcal{R}_k$. The complete training pipeline is summarized in Algorithm 2 in the Appendix.

Dirichlet trajectory. We pre-sample reveal counts $\mathbf{n} = (n_1, \ldots, n_K)$ from a symmetric Dirichlet–Multinomial distribution over the K denoising steps, which partitions the N_m maskable positions into step-wise reveal budgets. A random permutation of maskable positions determines the reveal order, and step k reveals the next n_k positions in that order. The timestep at step k is defined as a normalized masked ratio $t^{(k)} = |\mathcal{M}_k|/N_m \in [0, 1]$.

4.2.1 Loss Function

At each unrolling step k, the objective interpolates between dense supervision over all positions that remain masked and focused supervision over the subset scheduled for revelation:

$$\mathcal{L}_k = w_k \Big[\lambda_d \, \ell_k^{\text{dense}} + (1 - \lambda_d) \, \ell_k^{\text{reveal}}\Big], \qquad w_k = \frac{n_k}{N_m},$$

where $n_k = |\mathcal{R}_k|$ is the number of positions revealed at step k, and N_m is the total number of maskable positions. The dense and reveal losses are defined as

$$\ell_k^{\text{dense}} = \frac{1}{|\mathcal{M}_k|} \sum_{i \in \mathcal{M}_k} -\log p_\theta\Big(x_0^{(i)} \mid \mathbf{x}^{(k)}\Big), \qquad \ell_k^{\text{reveal}} = \frac{1}{n_k} \sum_{i \in \mathcal{R}_k} -\log p_\theta\Big(x_0^{(i)} \mid \mathbf{x}^{(k)}\Big),$$

where $\mathcal{M}_k$ denotes the set of positions still masked at step k, and $\mathcal{R}_k \subseteq \mathcal{M}_k$ denotes the subset selected for revelation at that step.

We also include a regularization term to penalize per-slot norms that exceed a threshold τ:

$$\mathcal{R}eg_s = \frac{\lambda_s}{KM} \sum_{k=1}^{K} \sum_{m=1}^{M} [\max(0, \|s_m^{(k)}\|_2 - \tau)]^2,$$

where τ leaves the state free below the threshold, and λ_s is the regularization weight. The total loss is the sum of per-step losses $\sum_{k=1}^{K} \mathcal{L}_k$ and the regularization term $\mathcal{R}eg_s$.

5 Experiments

5.1 Experimental Settings

Models and Datasets. We apply MetaState to two discrete diffusion LLM families, each in Base and Instruct variants: LLaDA-Instruct-8B / LLaDA-Base-8B (Nie et al., 2025) and Dream-v0-Instruct-7B / Dream-v0-Base-7B (Ye et al., 2025). To isolate the effect of the recurrent design, all backbone parameters are frozen throughout training. Only the MetaState components (Mixer, Updater, Injector, and the time-conditioning module) are trained, amounting to approximately 0.6% of each backbone. We train on 50,000 sequences sampled

Table 2: Compatibility with Soft Diffusion (Zhu et al., 2025c). MetaState and Soft Diffusion target different levels of the pipeline and can be combined. † denotes '+ Soft Diffusion'. **Bold** marks the best result per column. Hyperparameter details are provided in Appendix A.7.

	Dream-Instruct (7B)				*LLaDA-Instruct (8B)*				
Method	GSM8K	MATH	HE	MBPP	GSM8K	MATH	HE	MBPP	**Avg.**
Instruct baseline	74.8	45.0	56.1	51.0	78.5	36.8	37.2	26.0	50.7
+ Soft Diffusion	79.1	45.8	**60.4**	52.0	78.7	**37.8**	37.8	29.0	52.6
+ MetaState	78.1	**46.6**	59.8	55.0	79.5	**37.8**	39.6	32.2	53.6
+ MetaState†	**79.4**	46.4	59.2	**55.6**	**80.3**	37.6	**40.9**	**32.4**	**54.0**

from the Tülu-3 SFT mixture (allenai/tulu-3-sft-mixture) (Lambert et al., 2024), using each model's native tokenizer and chat template with a maximum sequence length of 1024.

Evaluation Benchmarks. We evaluate on four reasoning benchmarks: GSM8K (Cobbe et al., 2021) (5-shot) and MATH-500 (Lewkowycz et al., 2022) (4-shot) for mathematical reasoning, and HumanEval (Chen et al., 2021) (0-shot) and MBPP (Austin et al., 2021b) (3-shot) for code generation. For code benchmarks, accuracy refers to Pass@1 measured by functional correctness on unit tests. Following standard dLLM practice, the generation length is 256 with a block size of 32. Additional decoding-configuration results are shown in Appendix §A.9. All evaluations use the KV-cache and parallel decoding of Fast-dLLM (Wu et al., 2025).

5.2 Main Results

Table 1 compares MetaState against both Base and Instruct backbones on four reasoning benchmarks. MetaState consistently improves accuracy over both Base and Instruct variants across all benchmarks and both dLLM families. On Dream, MetaState outperforms Dream-Base on every benchmark, with gains of +3.0 on GSM8K, +8.8 on MATH-500, +4.3 on HumanEval, and +1.0 on MBPP. The same trend holds against the stronger Dream-Instruct baseline, where MetaState improves GSM8K by +3.3, MATH-500 by +1.6, HumanEval by +3.7, and MBPP by +4.0. LLaDA exhibits the same pattern at a larger scale. Relative to LLaDA-Base, MetaState yields gains of +10.5 on GSM8K, +8.2 on MATH-500, +6.1 on HumanEval, and +7.4 on MBPP. Relative to LLaDA-Instruct, the improvements are smaller but remain consistent, with gains of +1.0 on GSM8K, +1.0 on MATH-500, +2.4 on HumanEval, and +6.2 on MBPP. The smaller margins over Instruct backbones are consistent with lower recoverable cross-step loss after instruction tuning; MetaState does not update backbone weights and instead changes the augmented system through an input-side recurrent memory interface.

These results suggest that persistent working memory is particularly beneficial for tasks that require information to remain stable across long denoising trajectories. In mathematical reasoning, the model must retain intermediate computations and partial conclusions until the final answer is formed. In code generation, it must maintain global structural constraints such as variable scope, control flow, and program-level consistency over many lines of code. Both settings are vulnerable to cross-step drift, and the consistent gains of MetaState across all four benchmarks and two architecturally distinct dLLM families support the view that its benefits arise from mitigating the Information Island issue. We further evaluate MetaState on the newer RL-optimized LLaDA 1.5 backbone in Appendix A.13; MetaState improves all four evaluated metrics under the same protocol, supporting transfer beyond the original checkpoints.

5.3 Compatibility with Soft Diffusion

Both MetaState and Soft Diffusion (Zhu et al., 2025c) improve dLLM decoding but through orthogonal mechanisms. Soft Diffusion directly modifies the original discrete decoding path by replacing hard masked token positions with probability-weighted embedding mixtures. MetaState, in contrast, leaves the discrete path unchanged and instead introduces

Table 3: Ablation studies. Each row modifies one component while keeping the rest of MetaState intact. MLP variants are parameter-matched. MATH = MATH-500, HE = HumanEval. **Bold**: full model. Gray: backbone without MetaState (from Table 1).

	Dream-Instruct (7B)				*LLaDA-Instruct (8B)*				
Model Variants	GSM8K	MATH	HE	MBPP	GSM8K	MATH	HE	MBPP	**Avg.**
Backbone only	74.8	45.0	56.1	51.0	78.5	36.8	37.2	26.0	50.7
MetaState (full)	**78.1**	**46.6**	**59.8**	**55.0**	**79.5**	**37.8**	**39.6**	**32.2**	**53.6**
w/o recurrence	77.6	45.0	61.0	53.4	62.6	32.6	23.8	24.6	47.6
w/o BPTT	76.7	43.4	59.2	53.2	78.2	34.4	38.4	33.0	52.1
w/o time cond.	77.2	44.8	57.9	54.6	78.9	36.2	39.0	31.8	52.6
MLP Injector	77.2	41.2	54.3	52.2	77.7	36.2	38.4	30.8	51.0
MLP Mixer	76.9	42.4	56.7	54.2	74.4	33.8	33.5	32.2	50.5

a parallel persistent memory path that carries continuous information. In this sense, Soft Diffusion refines how token representations are formed within each step, whereas MetaState augments the decoding process with an additional cross-step information channel. Because one modifies the discrete path itself and the other adds a separate recurrent pathway alongside it, the two methods are naturally orthogonal and can be combined. Table 2 evaluates this combination on the Instruct variants of both backbones.

Comparing MetaState and Soft Diffusion individually (rows 2 and 3 of Table 2), MetaState outperforms Soft Diffusion on most benchmarks. Applying Soft Diffusion on top of MetaState brings further improvements, and the combination achieves the strongest overall results, including the best GSM8K accuracy of 80.3 and 79.4 on LLaDA and Dream. These results support the view that the two methods are complementary. The only exception is Dream-HumanEval, where MetaState + Soft Diffusion (59.2) underperforms both MetaState alone (59.8) and Soft Diffusion alone (60.4). We attribute this to two factors. First, HumanEval contains only 164 problems, so small differences in accuracy are inherently noisy. Second, the Injector is trained with pure mask embeddings as input, whereas Soft Diffusion replaces them with probability-weighted embedding mixtures, introducing an input distribution shift that may interfere with the Injector's additive modulation. Further details on Soft Diffusion hyperparameters are provided in Appendix A.7.

5.4 Ablation Studies

Table 3 presents ablation results for each MetaState component on both Dream-Instruct and LLaDA-Instruct under the same 50k-sample training setup.

Recurrence. Zeroing the previous state at every denoising step (w/o recurrence) has asymmetric effects across backbones: on LLaDA, performance drops substantially below the backbone-only baseline (-15.9 on GSM8K and -13.4 on HumanEval), indicating that the Injector without recurrent state becomes harmful. On Dream, in contrast, the effect is minimal, and the variant still remains above the Dream backbone on three of four benchmarks. This asymmetry suggests that Dream retains stronger per-step coherence during denoising, whereas LLaDA depends more heavily on the recurrent memory channel to preserve information across steps. Detaching the state between denoising steps (w/o BPTT), which preserves accumulation but blocks gradient flow through the unrolled trajectory, consistently degrades both backbones, with the largest drops on MATH-500 (-3.2 for Dream, -3.4 for LLaDA), showing that BPTT is important for learning useful state dynamics.

Architecture. Replacing the attention-based Injector with a parameter-matched MLP (MLP Injector) that flattens the slot state and broadcasts a uniform bias removes position-aware injection while preserving a path from memory to the token representations. This leads to clear degradation on both backbones, with particularly large drops on Dream (-5.5 on HumanEval and -5.4 on MATH-500). Replacing the attention-based Mixer with a parameter-matched MLP (MLP Mixer) that average-pools backbone hidden states before projection likewise hurts performance consistently, and is especially damaging on LLaDA (-5.1 on GSM8K and -6.1 on HumanEval). Together, these results show that both token-

selective reading in the Mixer and position-aware writing in the Injector are important. Removing explicit time conditioning by zeroing the SharedTimeConditioner (w/o time cond.) produces the smallest drops across benchmarks, suggesting that while timestep information is beneficial, the recurrent gating and attention mechanisms can partially infer denoising progress even without explicit conditioning.

6 Conclusion

We presented MetaState, a lightweight recurrent augmentation together with a dedicated training pipeline that equips frozen discrete diffusion LLM backbones with a persistent, fixed-size working memory across denoising steps, thereby addressing the Information Island issue. On frozen LLaDA-8B and Dream-7B backbones, MetaState only adds approximately 0.6% trainable parameters and yields consistent improvements on both mathematical reasoning and code generation benchmarks, demonstrating that persistent cross-step working memory is an effective mechanism for improving reasoning performance in dLLMs.

7 Acknowledgments

This material is based upon work supported by the National Science Foundation under grant No.2229876 and is supported in part by funds provided by the National Science Foundation, by the Department of Homeland Security, and by IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or its federal agency and industry partners.

References

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021a.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021b.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji,

Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, et al. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 1597–1600. IEEE, 2017.

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, et al. Scaling diffusion language models via adaptation from autoregressive models. *arXiv preprint arXiv:2410.17891*, 2024.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Yuezhou Hu, Harman Singh, Monishwaran Maheswaran, Haocheng Xi, Coleman Hooper, Jintao Zhang, Aditya Tomar, Michael W Mahoney, Sewon Min, Mehrdad Farajtabar, et al. Residual context diffusion language models. *arXiv preprint arXiv:2601.22954*, 2026.

Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. Reinforcing the diffusion chain of lateral thought with diffusion language models. *arXiv preprint arXiv:2505.10446*, 2025.

Allan Jabri, David Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. *arXiv preprint arXiv:2212.11972*, 2022.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. Ladir: Latent diffusion enhances llms for text reasoning. *arXiv preprint arXiv:2510.04573*, 2025.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. 2024.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

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

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

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

Justin Lovelace, Christian Belardi, Sofian Zalouk, Adhitya Polavaram, Srivatsa Kundurthy, and Kilian Q Weinberger. Stop-think-autoregress: Language modeling with latent diffusion planning. *arXiv preprint arXiv:2602.20528*, 2026.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

Patrick Pynadath, Jiaxin Shi, and Ruqi Zhang. Candi: Hybrid discrete-continuous diffusion models. *arXiv preprint arXiv:2510.22510*, 2025.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 2020.

Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025.

Dachuan Shi, Abedelkadir Asi, Keying Li, Xiangchi Yuan, Leyan Pan, Wenke Lee, and Wen Xiao. Swireasoning: Switch-thinking in latent and explicit for pareto-superior reasoning llms. *arXiv preprint arXiv:2510.05069*, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. Time is a feature: Exploiting temporal dynamics in diffusion language models. *arXiv preprint arXiv:2508.09138*, 2025.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 2002.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Chunhui Zhang, Yiren Jian, Zhongyu Ouyang, and Soroush Vosoughi. Working memory identifies reasoning limits in language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16896–16922, 2024.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *arXiv preprint arXiv:2505.15778*, 2025.

Huangjie Zheng, Shansan Gong, Ruixiang Zhang, Tianrong Chen, Jiatao Gu, Mingyuan Zhou, Navdeep Jaitly, and Yizhe Zhang. Continuously augmented discrete diffusion model for categorical generative modeling. *arXiv preprint arXiv:2510.01329*, 2025.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025a.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, et al. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*, 2025b.

Qinglin Zhu, Yizhen Yao, Runcong Zhao, Yanzheng Xiang, Amrutha Saseendran, Chen Jin, Philip Teare, Bin Liang, Yulan He, and Lin Gui. Latent refinement decoding: Enhancing diffusion-based language models by refining belief states. *arXiv preprint arXiv:2510.11052*, 2025c.

A Appendix

A.1 The Information Island Issue

The Information Island issue arises from the representational gap between the continuous hidden state $\mathbf{h}_t$ computed within each denoising step and the discrete sequence $\mathbf{x}_{t-1}$ passed to the next step. This gap is inherent to the Markovian formulation of dLLM decoding: each inter-step transition conditions solely on the current discrete state $\mathbf{x}_t$, while the continuous representation $\mathbf{h}_t$ is not carried forward explicitly. On one side, $\mathbf{h}_t$ encodes substantially richer information than what the discrete tokens can carry: beyond token-level predictive distributions, it captures long-range dependencies, partial reasoning information, and global structural constraints over the sequence. On the other side, the next-step input $\mathbf{x}_{t-1} = \mathcal{S}(\mathbf{h}_t)$ is produced by the sampling-and-remasking operator $\mathcal{S}$, which retains only discrete token identities at a sparse subset of positions and discards all remaining continuous context embedded in $\mathbf{h}_t$. This persistent gap at every transition along the denoising trajectory is what we term the Information Island issue.

This bottleneck degrades dLLMs in several related ways. First, useful information computed at one step cannot be directly reused at the next step, and the subsequent step can only re-derive it from a sparse, partially masked sequence that carries no trace of the earlier computation. Second, because this re-derivation occurs under changing noise levels and mask patterns, the trajectory can drift: information that is correct at one step may be weakened, overwritten, or inconsistently re-derived at later steps. Third, the problem is especially severe for reasoning. Tasks such as multi-step mathematics and code generation require the model to make use of intermediate computations. Without an explicit cross-step memory mechanism, these intermediate results are repeatedly exposed to the lossy discrete interface, making it difficult for them to make a useful contribution to later steps.

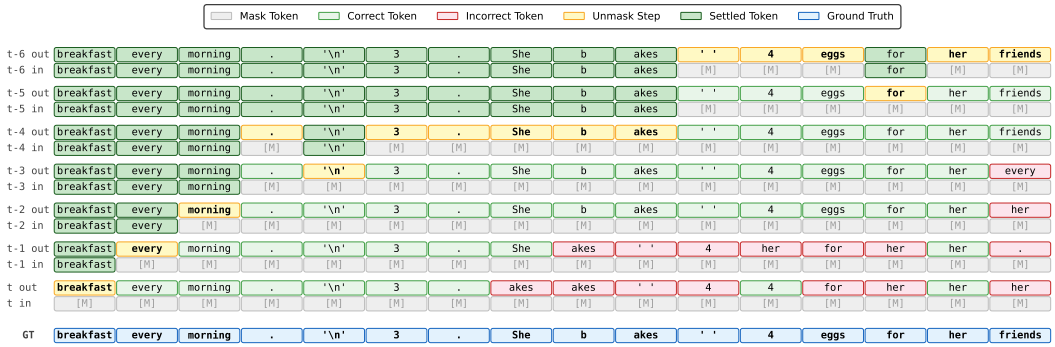

Figure 4: A step-by-step denoising trajectory. Each denoising step shows both the full argmax model prediction (`out`) and the remasked sequence (`in`) that is actually passed to the next step.

A step-by-step denoising example. Figure 4 makes this failure mode concrete by showing, at each denoising step, both (i) the full decoded prediction before remasking denoted by $\hat{\mathbf{x}}_t$, and (ii) the remasked sequence input $\mathbf{x}_{t-1}$ that is actually passed to the next step. The key observation is that correct information can already appear in $\hat{\mathbf{x}}_t$ several steps before generation is finalized, yet much of it is lost after remasking and never reaches subsequent steps. Each subsequent step receives only a partial discrete snapshot and must re-derive the missing relations from scratch. In the example, some correct tokens or reasoning fragments appear early, but because they are not preserved across the discrete interface, later steps fail to reuse them and may instead drift toward an inconsistent continuation.

Trajectory-level evidence from PASS@1 and EVERPASS@1. The Information Island issue not only discards useful intermediate representations, but can also degrade final generation quality, as correct tokens produced at earlier steps may be overwritten with incorrect ones after passing through the discrete sampling-and-remasking interface. Following recent analyses of temporal dynamics in dLLMs (Wang et al., 2025), Figure 5 compares the PASS@1 with EVERPASS@1, where EVERPASS@1 counts an example as successful if any intermediate full prediction along the denoising trajectory is correct.

Table 4: GSM8K trajectory diagnostic for LLaDA-Instruct. EVERPASS@1 counts an example as correct if any intermediate full prediction along the denoising trajectory is correct. The gap is EVERPASS@1 minus PASS@1.

Steps	Orig. PASS	Our PASS	Orig. EVER	Our EVER	gain	Orig. gap	Our gap
1–256	49.58	54.80	65.88	69.05	+5.23	16.30	14.25
256	63.00	66.00	79.00	79.00	+3.00	16.00	13.00
1–85	31.59	40.42	45.96	54.84	+8.84	14.38	14.41
86–170	56.34	59.87	73.89	74.95	+3.53	17.55	15.08
171–256	60.67	64.01	77.65	77.28	+3.34	16.98	13.27

On our test, EVERPASS@1 remains significantly higher than the PASS@1, indicating that many correct results are already present at intermediate denoising steps but are lost through the subsequent discrete remasking operation. In other words, the model often "knows" the right answer somewhere along the trajectory, yet fails to preserve that information through subsequent sampling and remasking. Together with the case study in Figure 4, these statistics show that the Information Island issue is a common failure mode in discrete diffusion language models.

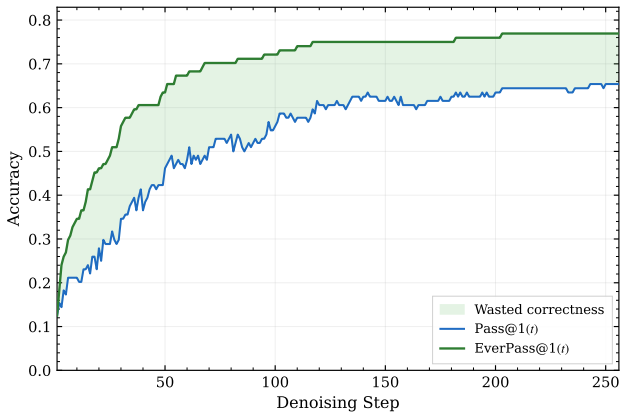

Figure 5: Comparison of PASS@1(t) and EVERPASS@1(t) across denoising steps on GSM8K with LLaDA-Instruct-8B.

These observations motivate MetaState, which introduces a persistent continuous working memory that carries useful intermediate information across steps and tackles the problems mentioned above.

A.2 Additional Trajectory Evidence

We further compare the trajectory-level PASS@1 and EVERPASS@1 diagnostics before and after adding MetaState on a subset of GSM8K. This diagnostic tests whether MetaState helps correct intermediate predictions survive the sampling-and-remasking interface. Table 4 shows that MetaState improves PASS@1 throughout the trajectory and reduces the wasted-correctness gap. At the final step, both systems reach the same EVERPASS@1 of 79.00, while final PASS@1 improves from 63.00 to 66.00. This pattern indicates that the augmented model does not merely make more answers reachable somewhere along the trajectory; it preserves more of the correct information that is already reachable so that it survives into the final output.

A.3 State and Memory Diagnostics

We next inspect the recurrent state itself. Table 5 shows that consecutive-step hidden-state cosine in the original backbone is nearly identical for Dream and LLaDA, so this raw coherence statistic does not explain the backbone asymmetry. In contrast, MetaState's state norm, slot norm, and state variance are consistently larger on LLaDA. This indicates that the recurrent interface performs more active cross-step work on LLaDA, which is consistent with the stronger effect of recurrence-related ablations on that backbone.

We also run a numeric semantic retention probe on the incoming memory state received by the Injector before it writes into the backbone input embeddings. For each recorded state, a low-capacity linear scorer ranks candidate numbers against three targets: gold intermediate

Table 5: GSM8K state and memory diagnostics. Two 128-example subsets show the same pattern: the original consecutive-step hidden-state cosine is nearly identical across backbones, while MetaState memory activity is larger and more variable on LLaDA.

Subset	Metric	Dream	LLaDA
1	Original response-step cosine, mean	0.9749	0.9775
2	Original response-step cosine, mean	0.9742	0.9769
1	MetaState state Frobenius norm, mean	31.42	52.04
2	MetaState state Frobenius norm, mean	31.21	51.98
1	MetaState mean slot norm, mean	3.93	6.50
2	MetaState mean slot norm, mean	3.90	6.50
1	State Frobenius pooled variance	24.11	76.43
2	State Frobenius pooled variance	24.27	77.29

Table 6: GSM8K numeric semantic retention probe. The probe reads the incoming MetaState memory before injection and ranks candidate numbers. Controls include random ranking, constant-state features, question-number preference, and visible-text-number preference.

Backbone	Target	Slice	MetaState MRR	Best control MRR	Gap
LLaDA	gold_intermediate	all	0.663	0.434	+0.229
LLaDA	gold_intermediate	not_visible	0.588	0.363	+0.225
LLaDA	gold_intermediate	not_in_question	0.646	0.410	+0.235
LLaDA	model_final	all	0.308	0.220	+0.088
LLaDA	model_final	not_visible	0.318	0.220	+0.099
LLaDA	model_final	not_in_question	0.311	0.220	+0.092
LLaDA	gold_final	all	0.327	0.220	+0.107
LLaDA	gold_final	not_visible	0.326	0.219	+0.106
LLaDA	gold_final	not_in_question	0.327	0.219	+0.107
Dream	gold_intermediate	all	0.664	0.473	+0.191
Dream	gold_intermediate	not_visible	0.546	0.349	+0.197
Dream	gold_intermediate	not_in_question	0.649	0.433	+0.216
Dream	model_final	all	0.303	0.217	+0.086
Dream	model_final	not_visible	0.299	0.217	+0.082
Dream	model_final	not_in_question	0.305	0.217	+0.087
Dream	gold_final	all	0.315	0.217	+0.098
Dream	gold_final	not_visible	0.315	0.216	+0.098
Dream	gold_final	not_in_question	0.317	0.216	+0.101

values, the model's final-answer candidate, and the gold final answer. The candidate pool includes question numbers, numbers already visible in the decoded response, target values, and magnitude-matched distractors. Table 6 reports mean reciprocal rank against the strongest among four controls: random ranking, constant-state features, question-number preference, and visible-text-number preference. MetaState exceeds the strongest control across both backbones and all target types, including the not-visible and not-in-question slices, showing that the state contains recoverable numeric reasoning information beyond the prompt and currently decoded text.

A.4 Pseudocode for MetaState

Algorithm 1 details the single denoising step procedure of MetaState, and Algorithm 2 summarizes the full K-step iterative unrolling training procedure.

Algorithm 1 MetaState: Single Denoising Step

Require: Noisy sequence $\mathbf{x}_t$, persistent state $\mathbf{s}_t$ (or None), previous time conditioning $\mathbf{t}_{\text{cond}}$ (or None), timestep t
Ensure: Logits $\hat{\mathbf{x}}_t$, updated state $\mathbf{s}_{t-1}$, time conditioning $\mathbf{t}_{\text{cond}}$

$\mathbf{e}_t \leftarrow \text{Embed}(\mathbf{x}_t)$
if $\mathbf{s}_t = \text{None}$ **then** ▷ Warmup step
  $\mathbf{s}_t \leftarrow \mathbf{s}_0$ ▷ Learnable init
  $\tilde{\mathbf{e}}_t \leftarrow \mathbf{e}_t$ ▷ Skip injection
else
  $\tilde{\mathbf{e}}_t \leftarrow \text{Injector}(\mathbf{e}_t, \mathbf{s}_t, \mathbf{t}_{\text{cond}})$ ▷ Additive modulation
end if
$(\mathbf{h}_t, \hat{\mathbf{x}}_t) \leftarrow p_\theta(\tilde{\mathbf{e}}_t)$ ▷ Frozen backbone
$\bar{\mathbf{h}}_t \leftarrow \text{MeanPool}(W^h_{\downarrow}\, \mathbf{h}_t)$ ▷ Content summary
$\mathbf{t}_{\text{cond}} \leftarrow \text{TimeCond}(t, \bar{\mathbf{h}}_t)$ ▷ Content-aware
$\mathbf{c}_t \leftarrow \text{Mixer}(\mathbf{s}_t, \mathbf{h}_t, \mathbf{t}_{\text{cond}})$ ▷ Read from $\mathbf{h}_t$
$\mathbf{s}_{t-1} \leftarrow \text{Updater}(\mathbf{s}_t, \mathbf{c}_t, \mathbf{t}_{\text{cond}})$ ▷ Update state
return $(\hat{\mathbf{x}}_t, \mathbf{s}_{t-1}, \mathbf{t}_{\text{cond}})$ ▷ For next step

Algorithm 2 MetaState Training

Require: Ground truth $\mathbf{x}_0$, maskable positions $\mathcal{M}$, K steps

$\mathbf{x} \leftarrow$ mask all positions in $\mathcal{M}$ with `[MASK]`
$(\text{logits}, \mathbf{s}, \mathbf{t}_{\text{cond}}) \leftarrow \text{Forward}(\mathbf{x}, \text{state}{=}\text{None}, \mathbf{t}_{\text{cond}}{=}\text{None}, t{=}1.0)$ ▷ Warmup (Alg. 1)
Sample reveal counts $n_1, \ldots, n_K \sim$ Dir-Multi
Sample random reveal ranks for $\mathcal{M}$
$\mathcal{L} \leftarrow 0$
for $k = 1$ to K **do**
  $t \leftarrow |\text{still masked}|/N_m$ ▷ Continuous timestep
  $(\text{logits}, \mathbf{s}, \mathbf{t}_{\text{cond}}) \leftarrow \text{Forward}(\mathbf{x}, \mathbf{s}, \mathbf{t}_{\text{cond}}, t)$ ▷ Alg. 1
  $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_k(\text{logits}, \mathbf{x}_0, \mathcal{M}_k, \mathcal{R}_k)$
  Reveal n_k positions (teacher forcing), update $\mathbf{x}$
end for
return $\mathcal{L} + \mathcal{R}eg_s$

A.5 Experimental Details and Hyperparameters

A.5.1 Experimental Details.

We freeze the backbone and train only the MetaState recurrent components (Mixer, Updater, Injector, and SharedTimeConditioner) with AdamW (β_1=0.9, β_2=0.95), a peak learning rate of 2×10^{-5} with cosine decay and 5% linear warmup, and gradient clipping at max norm 1.0. All Metastate weights are initialized from a truncated normal distribution (σ=0.02), except from the zero-initialized ones. We train for one epoch on 50,000 sequences from the Tülu-3 SFT mixture (Lambert et al., 2024), with a maximum sequence length of 1,024 tokens, using bfloat16 mixed precision with DeepSpeed ZeRO-1 (Rajbhandari et al., 2020) on two NVIDIA H200 GPUs. Unless noted otherwise, all hyperparameters are shared across both backbones: M=64 memory slots, state dimension D_s=1024, bottleneck dimensions d_m=d_b=768, and unroll depth K=4. Loss is computed only over the response portion, and prompt tokens are excluded from both masking and loss computation. A hinge state-norm regularizer (λ_s=$1e{-}4$, threshold τ=1.0) penalizes per-slot norms that exceed the threshold.

During evaluation, Dream performs a single bootstrap forward pass at t=1.0 with $\mathbf{s}$=None before denoising begins, initializing the recurrent state. Chat-template application varies by backbone: LLaDA-Base evaluations omit the chat template entirely. LLaDA-Instruct applies

it for GSM8K and MATH-500 but omits it for HumanEval and MBPP. All Dream variants (Base and Instruct) apply the backbone's native chat template. Table 8 lists the remaining inference hyperparameters.

A.5.2 Architecture Hyperparameters.

Table 7 summarizes the architectural configuration of all MetaState modules.

Table 7: Architecture hyperparameters for the recurrent modules. All symbols correspond to notation introduced in §4.

Symbol	Description	Value
M	Number of memory slots	64
D_s	State dimension per slot	1024
d_c	Time conditioner output dimension	1024
d_m	Mixer bottleneck dimension	768
d_b	Injector bottleneck dimension	768
n_q	Query heads (Mixer & Injector)	8
n_{kv}	KV heads (GQA)	4
—	SwiGLU FFN expansion ratio	2.0
$\boldsymbol{\alpha}_g$	Content gate initialization	$\mathbf{0}$ (zero-init)
$\mathcal{N}_0$	Zero-bridge soft bias	10^{-3}

A.5.3 Training and Inference Hyperparameters.

Table 8 lists the optimization, unrolling, and inference settings shared across both LLaDA and Dream backbones.

Table 8: Training and inference hyperparameters (shared across LLaDA and Dream backbones).

Parameter	Value
Optimization	
Optimizer	AdamW
Learning rate	2×10^{-5}
LR schedule	cosine, 5% linear warmup
Gradient clipping	max norm 1.0
β_1, β_2	$0.9, 0.95$
Batch size (effective)	8 (4/GPU $\times$ 2 GPUs)
Epochs	1
Max sequence length	1024
Precision	bfloat16
Unrolling & Loss	
K (unroll steps)	4
λ_d (dense reveal mix)	0.75
λ_s (state norm weight)	10^{-4}
τ (state norm threshold)	1.0
Inference	
Max generation length	256
Block size	32
KV cache mode	dual
Confidence threshold	0.9

A.6 Model Parameter Analysis

All recurrent modules operate in bottleneck dimensions d_m=d_b=768 and memory slot dimension D_s=1024 rather than the backbone hidden size D, which keeps the parameter

count compact. Only three interface projections, the Mixer's backbone down-projection $\mathbf{W}^h_{\downarrow} \in \mathbb{R}^{d_m \times D}$, and the Injector's input/output projections $\mathbf{W}^e_{\downarrow} \in \mathbb{R}^{d_b \times D}$, $\mathbf{W}_{\uparrow} \in \mathbb{R}^{D \times d_b}$, depend on the backbone hidden size D. All other parameters are shared the same across backbones. Table 9 provides a per-module breakdown.

Table 9: Per-module parameter breakdown of the MetaState recurrent modules for each backbone (D_s=1024, d_m=d_b=768, M=64). Rows marked † are the only backbone-dependent components.

Module / Component	LLaDA	Dream
SharedTimeConditioner	2,102,016	2,102,016
Sinusoidal MLP (256→1024→1024)	1,312,768	1,312,768
Content path (norm + proj + gate)	789,248	789,248
Mixer	17,899,264	17,506,048
Slot self-attention (D_s space, GQA)	3,146,752	3,146,752
Down-projections† ($d_m \times D_s + d_m \times D$)	3,932,160	3,538,944
Cross-attention + AdaRMSNorms	6,494,976	6,494,976
SwiGLU FFN + up-projection	4,325,376	4,325,376
Updater	10,626,048	10,626,048
AdaRMSNorms (×2)	4,200,448	4,200,448
GRU projections (gate + candidate)	6,294,528	6,294,528
Learnable $\mathbf{s}_0$ + slot embeddings	131,072	131,072
Injector	20,457,217	19,670,785
Interface projections† ($d_b \times D + D \times d_b$)	6,291,456	5,505,024
Self-attention (RMSNorm, GQA)	1,770,241	1,770,241
State proj + cross-attn + AdaRMSNorms	8,856,576	8,856,576
SwiGLU FFN	3,538,944	3,538,944
Total trainable	**51,084,545**	**49,904,897**
Frozen backbone	8,056,602,369	7,655,458,049
Overhead	**0.63%**	**0.65%**

A.7 Soft Diffusion Hyperparameter Details

Soft Diffusion, introduced in Latent Refinement Decoding (LRD) (Zhu et al., 2025c), replaces the hard remasking at mask positions with a weighted mixture of token embeddings. For each mask position i with sampled token $\hat{x}_i$, the input embedding becomes:

$$\tilde{\mathbf{e}}_i = (1 - r_f) \cdot \mathbf{e}_{\hat{x}_i} + r_f \cdot \textstyle\sum_v p_v \cdot \mathbf{e}_v,$$

where p_v is the model's predicted probability for token v (after nucleus filtering with threshold p), $r_f \in [0, 1]$ is the mix ratio factor controlling the mixture weight, and $\mathbf{e}_v$ denotes the token embedding for vocabulary entry v. The two key hyperparameters are the nucleus probability threshold (top-p) and the mix ratio factor (r_f).

Following the practice of LRD (Zhu et al., 2025c), we consider top-$p \in \{0.2, 0.9\}$, since the paper reports that values ≥ 0.2 are effective, and $r_f \in \{0.1, 0.15, 0.2\}$, which spans the effective range $[0.1, 0.2]$ reported in the paper. For each method–backbone–benchmark combination, we evaluate all six (p, r_f) pairs listed above and report the best-performing configuration. Table 10 lists the selected hyperparameters.

Table 10: Best Soft Diffusion hyperparameters (p, r_f) for each method, backbone, and benchmark. All values selected by grid search over $p \in \{0.2, 0.9\}$ and $r_f \in \{0.1, 0.15, 0.2\}$.

Backbone	Method	GSM8K	MATH	HE	MBPP
Dream-Instruct	Soft Diffusion	(0.9, 0.15)	(0.9, 0.1)	(0.9, 0.2)	(0.9, 0.2)
Dream-Instruct	+ MetaState	(0.2, 0.1)	(0.2, 0.2)	(0.9, 0.1)	(0.2, 0.1)
LLaDA-Instruct	Soft Diffusion	(0.2, 0.1)	(0.2, 0.15)	(0.9, 0.2)	(0.2, 0.2)
LLaDA-Instruct	+ MetaState	(0.2, 0.1)	(0.9, 0.2)	(0.2, 0.15)	(0.2, 0.2)

To check that the compatibility conclusion is not driven only by the best selected hyperparameter in each cell, we also average over all six Soft Diffusion grid configurations. Table 11 shows that Soft Diffusion combined with MetaState improves six of eight backbone–benchmark averages, with only small drops on LLaDA-Instruct MATH-500 and Dream-Instruct HumanEval. This supports the same conclusion as the best-configuration table: Soft Diffusion and MetaState are largely complementary, while occasional small drops can occur from the embedding-mixture input distribution shift.

Table 11: Soft Diffusion average over all six grid configurations. Averaging removes dependence on a single selected (p, r_f) pair.

Backbone	Metric	Soft Diff. avg	Soft Diff. + Meta avg	Δ
LLaDA-Instruct	GSM8K flexible EM	78.14	79.48	+1.34
LLaDA-Instruct	HumanEval pass@1	36.89	39.12	+2.23
LLaDA-Instruct	MBPP pass@1	28.27	31.77	+3.50
LLaDA-Instruct	MATH-500 math verify	37.23	37.20	-0.03
Dream-Instruct	GSM8K flexible EM	76.25	78.27	+2.02
Dream-Instruct	HumanEval pass@1	57.52	56.91	-0.61
Dream-Instruct	MBPP pass@1	51.24	54.28	+3.04
Dream-Instruct	MATH-500 math verify	44.20	44.83	+0.63

A.8 Comparison with LoRA Fine-Tuning

MetaState and LoRA (Hu et al., 2022) both introduce a small number of trainable parameters on top of a frozen backbone, but they operate at fundamentally different levels. LoRA injects low-rank updates into the backbone's linear layers, directly modifying the model's internal representations and thereby its learned inner capabilities. MetaState, by contrast, leaves all backbone weights unchanged and operates entirely at the denoising interface: it reads from the backbone hidden states after each forward pass through the Mixer and writes a lightweight additive signal into the token embeddings before the next pass through the Injector. No gradient flows into the backbone during MetaState training, and the backbone's per-step predictive behavior is affected only through this external recurrent state channel. In this sense, MetaState introduces cross-step information passing without enhancing the model's internal capability, whereas LoRA improves single-step capability.

Because the two methods affect different aspects of the generation process, this comparison should not be interpreted as a head-to-head evaluation of fine-tuning methods. Nevertheless, a LoRA baseline under a matched training setup helps rule out an alternative explanation of MetaState's gains: namely, that the improvements arise primarily from exposure to the Tülu-3 training distribution rather than from the working-memory mechanism itself. To test this possibility, we fine-tune the same Instruct backbones with LoRA (r=32, α=64, `target_modules = all-linear`) on the identical 50k Tülu-3 sequences, using the same optimizer, learning-rate schedule, and training budget described in Appendix A.5.1.

Table 12: Comparison with LoRA fine-tuning on Instruct backbones. LoRA (r=32, α=64, all-linear) is trained on the same 50k Tülu-3 data with matched optimization hyperparameters. † denotes '+ Soft Diffusion'. **Bold** marks the best result per column.

	Dream-Instruct (7B)		*LLaDA-Instruct (8B)*		
Method	GSM8K	HumanEval	GSM8K	HumanEval	**Avg.**
Instruct	74.8	56.1	78.5	37.2	61.7
+ LoRA	76.0	55.5	79.2	39.0	62.4
+ MetaState	78.1	**59.8**	79.5	39.6	64.3
+ MetaState†	**79.4**	59.2	**80.3**	**40.9**	**65.0**

Table 12 shows that LoRA underperforms MetaState on every benchmark pair. On Dream-Instruct, LoRA even slightly reduces HumanEval accuracy relative to the original backbone (55.5 vs. 56.1), whereas MetaState improves it to 59.8. On LLaDA-Instruct, LoRA narrows the gap but still remains below MetaState on both GSM8K (79.2 vs. 79.5) and HumanEval

Table 13: GSM8K block-size sweep. Block size changes the per-block denoising trajectory length; MetaState is positive at all tested sizes.

Backbone	Block size	Original	MetaState	Gain
Dream-Instruct	16	78.24	78.85	+0.61
Dream-Instruct	32	74.75	78.09	+3.34
Dream-Instruct	64	65.43	77.18	+11.75
Dream-Instruct	128	52.01	74.83	+22.82
Dream-Instruct	256	50.34	53.07	+2.73
LLaDA-Instruct	16	77.94	78.62	+0.68
LLaDA-Instruct	32	78.47	79.53	+1.06
LLaDA-Instruct	64	76.88	78.01	+1.13
LLaDA-Instruct	128	71.95	75.66	+3.71
LLaDA-Instruct	256	52.46	53.45	+0.99

(39.0 vs. 39.6). When MetaState is further combined with Soft Diffusion (last row), the margin widens on all benchmarks. Since LoRA directly updates backbone weights and therefore has greater capacity to absorb distributional regularities from the training data than MetaState's frozen-backbone design, its inferior performance suggests that MetaState's gains cannot be explained simply by data-domain exposure. Instead, the improvements should be attributed to the persistent working memory introduced by MetaState: by carrying continuous information across denoising steps, MetaState alleviates the Information Island issue (§A.1) in a way that LoRA's single-step weight adaptation cannot replicate.

A.9 Robustness Across Decoding Configurations

The main results use generation length 256, block size 32, and confidence threshold 0.9. We add three sweeps to test whether MetaState's gains depend on this single decoding configuration.

First, Table 13 varies block size, which directly changes the amount of information that must be preserved within each per-block denoising trajectory. MetaState improves all tested block sizes on both Dream-Instruct and LLaDA-Instruct. The gains become especially large for Dream at block sizes 64 and 128, where the original backbone degrades sharply and MetaState preserves much more accuracy. We do not interpret block size 256 as part of a monotonic trend, because the reveal process becomes aggressive and both the original and augmented models degrade substantially.

Second, Table 14 varies the confidence threshold under the main generation length and block size. Lower thresholds reduce the effective number of denoising updates. MetaState remains positive in all 28 matched GSM8K cells across thresholds from 0.60 to 0.95, indicating that the gain is not restricted to a single slow or fast decoding budget.

Third, Table 15 varies the maximum generation length on Dream-Instruct. MetaState improves both GSM8K and MBPP at all tested lengths from 128 to 1024 tokens, showing that the effect is not specific to the default 256-token setting.

Finally, Table 16 reports a GSM8K multi-seed check. MetaState improves all four settings across seeds, with large gains on Base backbones and smaller positive improvements on Instruct backbones.

A.10 Hyperparameter Sensitivity

To evaluate the robustness of MetaState, we vary four hyperparameters individually while keeping all others fixed at their default values in Table 8. Specifically, we sweep the dense-reveal loss mixing ratio $\lambda_d \in \{0.60, 0.75, 0.90\}$ (default: 0.75), the number of memory slots $M \in \{32, 48, 64\}$ (default: 64), the training set size $\in \{30\text{k}, 40\text{k}, 50\text{k}\}$ (default: 50k), and the unroll depth $K \in \{3, 4, 5\}$ (default: 4). All experiments use the same optimizer, learning-rate schedule, and evaluation protocol described in Appendix A.5.1, and are conducted on two NVIDIA A100 GPUs. Figures 6–9 report per-task accuracy on both Dream-Instruct and LLaDA-Instruct backbones.

Table 14: GSM8K confidence-threshold sweep at generation length 256, block size 32, and dual cache. Lower thresholds reduce the effective number of denoising updates. MetaState improves all matched cells.

Model	Threshold	Original	MetaState	Gain	Orig. fwd/seq	Meta fwd/seq
Dream-Base	0.95	73.16	76.42	+3.26	156.34	78.50
Dream-Base	0.85	70.43	75.28	+4.85	139.10	67.14
Dream-Base	0.80	70.58	74.15	+3.57	131.05	63.10
Dream-Base	0.75	69.60	73.16	+3.56	123.63	58.82
Dream-Base	0.70	66.26	70.20	+3.94	117.02	54.75
Dream-Base	0.65	64.37	67.48	+3.11	110.39	51.78
Dream-Base	0.60	60.27	64.67	+4.40	103.88	48.74
Dream-Instruct	0.95	75.59	79.76	+4.17	69.60	74.47
Dream-Instruct	0.85	73.84	77.94	+4.10	59.93	65.41
Dream-Instruct	0.80	74.15	77.33	+3.18	56.37	61.84
Dream-Instruct	0.75	71.65	73.77	+2.12	52.46	55.64
Dream-Instruct	0.70	68.23	72.48	+4.25	49.12	52.67
Dream-Instruct	0.65	65.88	68.46	+2.58	45.91	50.83
Dream-Instruct	0.60	63.61	63.91	+0.30	43.27	47.11
LLaDA-Base	0.95	68.46	78.09	+9.63	153.58	100.40
LLaDA-Base	0.85	68.16	79.23	+11.07	128.34	74.47
LLaDA-Base	0.80	68.16	77.71	+9.55	119.36	66.72
LLaDA-Base	0.75	68.01	77.41	+9.40	111.92	59.64
LLaDA-Base	0.70	67.17	77.63	+10.46	105.04	54.04
LLaDA-Base	0.65	66.11	76.50	+10.39	98.16	48.93
LLaDA-Base	0.60	61.71	72.93	+11.22	91.93	44.35
LLaDA-Instruct	0.95	76.80	78.47	+1.67	101.01	105.67
LLaDA-Instruct	0.85	75.97	78.32	+2.35	75.03	79.42
LLaDA-Instruct	0.80	75.51	77.26	+1.75	66.72	71.34
LLaDA-Instruct	0.75	76.35	76.57	+0.22	59.68	63.86
LLaDA-Instruct	0.70	74.30	77.03	+2.73	54.09	57.62
LLaDA-Instruct	0.65	75.44	75.83	+0.39	48.87	52.36
LLaDA-Instruct	0.60	71.87	72.78	+0.91	44.56	47.64

Table 15: Dream-Instruct generation-length sweep. MetaState remains positive on GSM8K and MBPP from 128 to 1024 tokens.

Length	GSM8K Orig.	GSM8K Meta	Gain	MBPP Orig.	MBPP Meta	Gain
128	69.98	71.87	+1.89	49.80	52.60	+2.80
256	74.75	78.09	+3.34	51.80	55.00	+3.20
384	75.89	78.85	+2.96	50.20	55.00	+4.80
512	75.59	79.83	+4.24	50.40	54.20	+3.80
768	74.30	79.53	+5.23	51.40	55.20	+3.80
1024	75.51	78.70	+3.19	52.20	54.60	+2.40

Across all four sweeps, performance varies only within a relatively narrow range on both backbones and across all benchmarks. Overall, these results suggest that MetaState is not overly sensitive to a single hyperparameter, and that the default setting represents a robust operating choice rather than a narrowly tuned optimum.

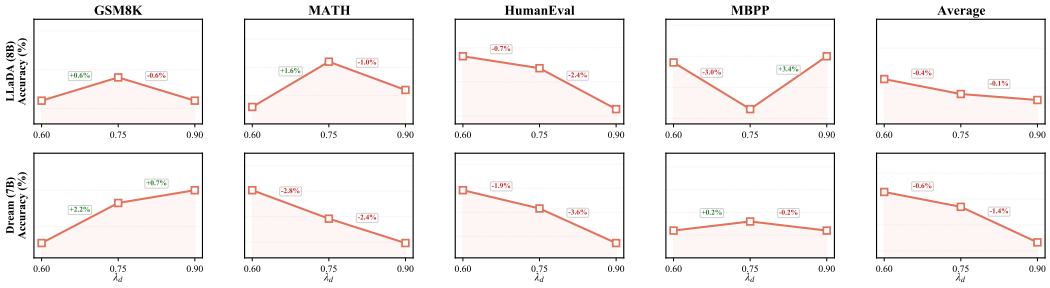

Figure 6: Sensitivity to the dense-reveal loss mixing ratio λ_d. Increasing λ_d from 0.6 to 0.9 yields modest drops in average scores.

Table 16: GSM8K multi-seed robustness. MetaState improves all four settings across seeds, with large gains on Base backbones and smaller improvements on Instruct backbones.

Model	Original	MetaState	Gain	Gain / pooled std
Dream-Base	73.92 ± 0.80	76.00 ± 0.88	+2.08	2.47
Dream-Instruct	75.19 ± 0.88	78.64 ± 0.67	+3.45	4.41
LLaDA-Base	67.87 ± 0.75	77.67 ± 0.67	+9.80	13.78
LLaDA-Instruct	78.16 ± 0.86	78.92 ± 0.98	+0.76	0.82

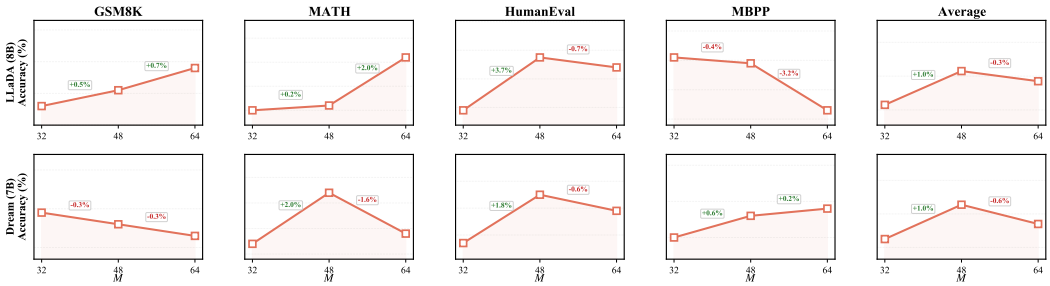

Figure 7: Sensitivity to the number of memory slots M. Increasing M from 32 to 48 yields modest gains on most benchmarks, while further increasing to 64 produces modest drops.

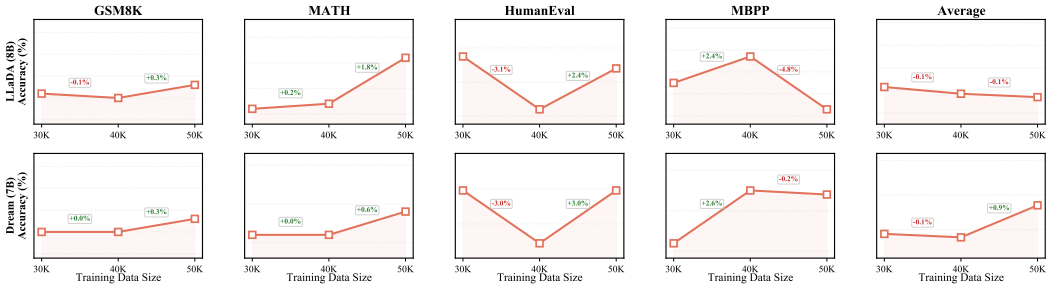

Figure 8: Sensitivity to training data size. MetaState achieves competitive performance even with 30k training examples, and accuracy remains stable as the dataset increases to 50k.

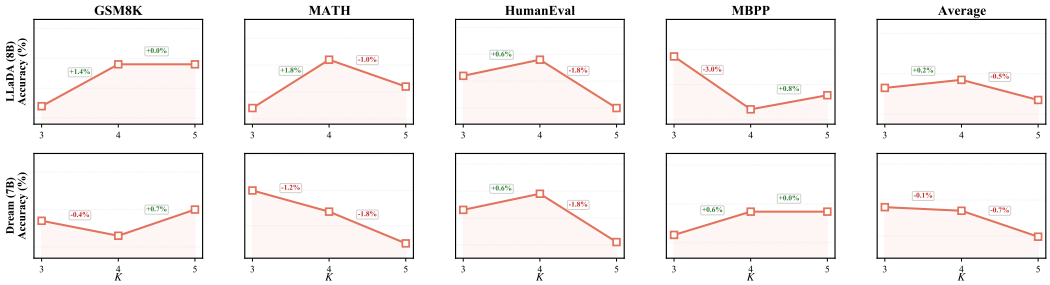

Figure 9: Sensitivity to the unroll depth K. Performance is broadly stable across $K \in \{3, 4, 5\}$, with average accuracy varying by less than one point.

A.11 Case Study

To provide a qualitative view of how MetaState influences the denoising process, we present case studies that compare the full denoising trajectories of the original LLaDA-Instruct and Dream-Instruct backbones with their counterparts augmented by MetaState on mathematical reasoning examples. All experiments in this section are conducted on NVIDIA A100 GPUs. The results show a clear qualitative pattern: with MetaState, the denoising trajectory is more likely to preserve correct intermediate computations, organize the correct solution structure earlier, and maintain a coherent reasoning path through later steps. These observations provide evidence that MetaState mitigates the Information Island issue by carrying forward useful intermediate information that would otherwise be lost across the discrete sampling-and-remasking interface.

Case Study 1: *LLaDA-Instruct (8B)* vs. *LLaDA-Instruct (8B) + MetaState*

Question:
Toula went to the bakery and bought various types of pastries. She bought 3 dozen donuts which cost $68 per dozen, 2 dozen mini cupcakes which cost $80 per dozen, and 6 dozen mini cheesecakes for $55 per dozen. How much was the total cost?
Please solve this step by step and end with #### <answer>.
Ground-truth answer: 694

Denoising Timestep $t = 0.80$

LLaDA-Instruct (8B)

```
Cost of donuts: [MASKx1] 3 [MASKx90]
```

LLaDA-Instruct (8B) + MetaState

```
The cost of the donuts is [MASKx90]
```

Denoising Timestep $t = 0.60$

```
Cost of donuts: 3 dozen [MASKx1]
$68 [MASKx1] dozen [MASKx8]
mini cupcakes: 2 dozen [MASKx1]
$80 [MASKx65]
```

```
The cost of the donuts is 3 [MASKx1]
$68 = $204 [MASKx25] mini [MASKx53]
```

Denoising Timestep $t = 0.40$

```
Cost of donuts: 3 dozen [MASKx1] $68
per dozen = $204
Cost of mini cupcakes: 2
dozen [MASKx1] $80 per dozen = $160
Cost of mini cheesecakes: 6 dozen *
$55 per dozen = $330
Total cost: [MASKx32]
```

```
The cost of the donuts is
3 [MASKx1] $68 = $204 [MASKx15]
$160. [MASKx1] The cost of the mini
cheesecakes is 6 [MASKx1] $55 = $330.
[MASKx36]
```

Denoising Timestep $t = 0.20$

```
Cost of donuts: 3 dozen * $68 per
dozen = $204
Cost of mini cupcakes: 2 dozen * $80
per dozen = $160
Cost of mini cheesecakes: 6 dozen *
$55 per dozen = $330
Total cost: $204 + $160 + $330 = $794
#### [MASKx6] 9 [MASKx1]
```

```
The cost of the donuts is 3 * $68 =
$204 [MASKx6] cupcakes is 2 * $80 =
$160.
The cost of the mini cheesecakes is 6
* $55 = $330.
[MASKx31] [MASKx1] 94
```

Denoising Timestep $t = 0.00$

```
Cost of donuts: 3 dozen * $68 per
dozen = $204
Cost of mini cupcakes: 2 dozen * $80
per dozen = $160
Cost of mini cheesecakes: 6 dozen *
$55 per dozen = $330
Total cost: $204 + $160 + $330 = $794
#### 794
```

```
The cost of the donuts is 3 * $68 =
$204 cupcakes is 2 * $80 = $160.
The cost of the mini cheesecakes is 6
* $55 = $330.
cost is $204 + $160 + $330 = $694 .
#### 694
```

Case Study 2: *Dream-Instruct (7B)* vs. *Dream-Instruct (7B) + MetaState*

Question:
Gloria is shoe shopping when she comes across a pair of boots that fit her shoe budget. However, she has to choose between the boots and two pairs of high heels that together cost five dollars less than the boots. If one pair of heels costs $33 and the other costs twice as much, how many dollars are the boots?
Please solve this step by step and end with #### <answer>.
Ground-truth answer: 104

Denoising Timestep $t = 0.80$

Dream-Instruct (7B)

```
[MASKx80] .
#### 13 [MASKx1]
The answer is: 1 [MASKx2]
```

Dream-Instruct (7B) + MetaState

```
If one pair of heels costs
$33 [MASKx1] the [MASKx83]
```

Denoising Timestep $t = 0.60$

```
[MASKx65] the boots [MASKx1]
$13 [MASKx1] + $5 = $13 [MASKx1] .
#### 13 [MASKx1]
The answer is: 13 [MASKx1]
```

```
If one pair of heels costs
$33 and the other costs twice
as much, then the [MASKx1]
pair of heels costs [MASKx6]
= [MASKx1] 66 [MASKx58] . [MASKx1]
```

Denoising Timestep $t = 0.40$

```
[MASKx3] of [MASKx52] cost
five dollars less than the
boots, [MASKx1] the boots cost $132 +
$5 = $137 .
#### 137
The answer is: 137
```

```
If one pair of heels costs $33 and
the other costs twice as much, then
the second pair of heels costs
$33 [MASKx1] 2 = $66.
[MASKx53] 104 . [MASKx1]
```

Denoising Timestep $t = 0.20$

If one pair of heels costs $33, then the other [MASKx41] heels together cost five dollars less than the boots, then the boots cost $132 + $5 = $137 .
137
The answer is: 137

If one pair of heels costs $33 and the other costs twice as much, then the second pair of heels costs $33 [MASKx1] 2 = $66.
[MASKx29] boots [MASKx1] $99 + $5 = $104 .
104
The answer is: 104 . [MASKx1]

Denoising Timestep $t = 0.00$

If one pair of heels costs $33, then the other pair costs $33 x 2 = $66.
The total cost of the two pairs of heels is $33 + $66 = $132.
If the two pairs of heels together cost five dollars less than the boots, then the boots cost$132 + $5 = $137 .
137
The answer is: 137

If one pair of heels costs $33 and the other costs twice as much, then the second pair of heels costs $33 x 2 = $66.
Together, the heels cost $33 + $66 = $99.
The heels cost five dollars less than the boots, so the boots cost $99 + $5 = $104 .
104
The answer is: 104 .

A.12 Practical Overhead

MetaState trains only the external recurrent interface while keeping the dLLM backbone frozen. Table 17 reports the resulting offline training wall-clock on the 50k Tulu-3 subset. Tables 18 and 19 report inference throughput, latency, and memory under the main GSM8K decoding setup. The current implementation increases per-sample latency by about 14–15% and reduces raw throughput by about 15–18%. The bf16 module footprint is about 90 MiB, and peak allocated memory rises by less than 1% on both backbones.

Table 17: Training wall-clock for the 50k Tulu-3 subset. The backbone remains frozen; the listed time is the offline cost of training the external MetaState interface.

Family	Data	Runtime
Dream-Instruct	50k	2.78 h (10012.8 s)
LLaDA-Instruct	50k	1.89 h (6787.9 s)
LLaDA-1.5	50k	1.87 h (6742.1 s)

Table 18: GSM8K inference throughput and latency. The current implementation adds about 14–15% latency and 15–18% raw throughput overhead under the main decoding setup.

Family	Raw tok/s	Drop	Sample latency	Overhead
Dream-Instruct	162.66 → 134.03	17.60%	3096.82 → 3664.79 ms	+15.49%
LLaDA-Instruct	150.20 → 127.47	15.13%	3815.96 → 4427.21 ms	+13.82%

Table 19: Memory overhead. The bf16 module footprint is about 90 MiB, and peak allocated memory rises by less than 1% for both backbones.

Family	Module size	State / seq	Allocated memory	Delta	Peak delta
Dream	89.18 MiB	0.125 MiB	14526.15 → 14635.34 MiB	+109.19 MiB	+181.50 MiB
LLaDA	91.43 MiB	0.125 MiB	15292.51 → 15402.32 MiB	+109.81 MiB	+117.84 MiB

Table 20: MetaState on LLaDA 1.5, a newer RL-optimized dLLM backbone. MetaState improves all four metrics under the same protocol, supporting transfer beyond the original LLaDA and Dream checkpoints.

Benchmark	Original	MetaState	Gain
GSM8K	80.74	82.56	+1.82
HumanEval	37.80	38.41	+0.61
MBPP	28.20	35.60	+7.40
MATH-500	37.40	38.00	+0.60

A.13 Result on an RL-Optimized Backbone

To test whether MetaState transfers beyond the original LLaDA and Dream checkpoints, we attach the same frozen-backbone recurrent interface to LLaDA 1.5, a newer RL-optimized dLLM backbone. Table 20 shows positive gains on all four metrics, with the largest gain on MBPP. This result is important because it shows that MetaState remains useful on a stronger post-trained dLLM, supporting the view that the method is an interface-level augmentation rather than a checkpoint-specific artifact.

A.14 Limitations

We discuss several limitations of the current approach. **Training overhead.** The K-step unrolling training pipeline requires $K+1$ sequential forward passes through the backbone per training iteration (one warmup pass plus K unrolled steps), compared to a single forward pass in standard dLLM training. This multiplicative increase in computation directly raises wall-clock training time and GPU memory consumption, as intermediate activations must be retained across steps for backpropagation through the unrolled computation graph, although the recurrent modules themselves are lightweight ($\sim$0.6% of backbone parameters). **Inference overhead.** At inference time, each denoising step requires executing the Mixer, Updater, and Injector in addition to the frozen backbone forward pass. Although these modules operate in compact bottleneck dimensions and add modest per-step latency, the overhead accumulates over the full denoising trajectory. Furthermore, maintaining the constant-size persistent state tensor increases peak memory usage during generation. **Potential solutions.** Several directions may mitigate the above limitations. Systems-level optimizations such as kernel fusion of the recurrent modules, hardware-aware scheduling to overlap recurrent and backbone computation, and selective recomputation strategies could reduce both training and inference overhead. Curriculum-based K scheduling or auxiliary objectives that explicitly encourage long-horizon state stability may also help close the training-to-inference extrapolation gap.