MemDLM: Memory-Enhanced DLM Training

Zehua Pei[1], **Hui-Ling Zhen**[2], **Weizhe Lin**[2], **Sinno Jialin Pan**[1],
Yunhe Wang[2], **Mingxuan Yuan**[2], **Bei Yu**[1]
[1]The Chinese University of Hong Kong [2]Huawei Technologies Co., Ltd

Abstract

Diffusion Language Models (DLMs) offer attractive advantages over Auto-Regressive (AR) models, such as full-attention parallel decoding and flexible generation. However, standard DLM training uses a static, single-step masked prediction objective, while inference progressively processes a sequence of denoising states. Long contexts additionally challenge the retention of task-relevant information through token-space attention. We propose **MemDLM** (Memory-Enhanced DLM), which introduces a second memory channel by writing local denoising experience into fast weights via Bi-level Optimization. An inner loop updates a set of fast weights across correlated denoising states, forming a *Parametric Memory* of local denoising experience, while an outer loop updates the base model conditioned on this memory. This memory-aware training complements token-space attention and yields stronger long-context performance, even when the fast weights are discarded at inference time. Re-enabling the inner loop at inference provides an additional prompt-specific adaptation effect, where the Parametric Memory acts as an emergent *in-weight retrieval* mechanism on challenging Needle-in-a-Haystack tasks.

1 Introduction

Diffusion Language Models (DLMs) are a promising alternative to Auto-Regressive (AR) models [24], offering parallel generation, bidirectional context awareness, and flexible text manipulation [1, 34, 19, 36, 26, 60, 5, 39, 20]. However, standard DLM training uses a static Masked Diffusion Language Modeling (MDLM) objective that optimizes one masked state at a time, whereas inference processes states sequentially [10, 50, 15, 32]. Retaining sample-specific information through token-space attention also becomes challenging as context length grows [23, 56, 17, 52].

We propose **MemDLM** (Memory-Enhanced DLM), which introduces a second memory channel by writing local denoising experience into parameter space during training.[1] An inner optimization loop updates parameter-efficient fast weights across correlated noise levels, forming a *Parametric Memory* of the local adaptation path [44, 2, 12, 38]. The base model is then optimized conditioned on this memory. This design provides a parameter-space channel alongside token-space attention and exposes training to sequential, stateful adaptation.

By internalizing local denoising experience into transient fast weights, MemDLM enriches the training signal and yields stronger long-context performance even when the fast weights are discarded at inference time. When the inner loop is optionally re-enabled at inference, it provides an additional prompt-specific adaptation pathway. As shown in Figure 1, on LLaDA-MoE [63], MemDLM improves RULER Variable Tracking [13] at 8K from 78.8% to 95.8%, and on LLaDA2.1 [4], it improves BABILong [16] at 8K from 47.4% to 57.0%. Figure 2 summarizes the overall design.

In summary, our contributions are:

[1]Code: https://github.com/JarvisPei/MemDLM.

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

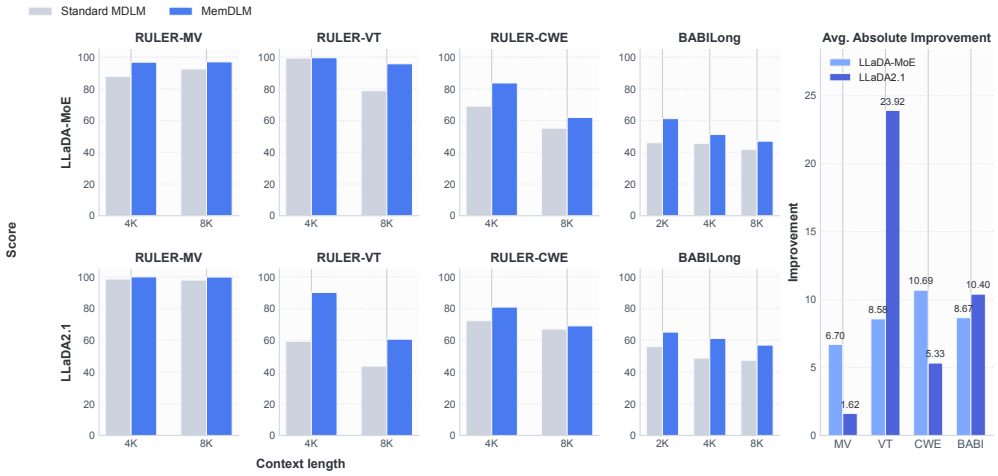

Figure 1: **Needle-in-a-Haystack results overview.** Gray bars denote Standard MDLM and blue bars denote MemDLM. Left: detailed results on RULER-MV, RULER-VT, RULER-CWE, and BABILong for the LLaDA-MoE-7B-A1B-Base and LLaDA2.1-mini backbones. Right: mean absolute improvement of MemDLM over Standard MDLM for each task, averaged across the evaluated context lengths within each backbone.

- We study two challenges of standard DLM training: learning from isolated masked states and retaining task-relevant information through token-space attention in long contexts.
- We introduce **MemDLM**, a Bi-level Optimization framework that trains through an anchor-consistent multi-noise adaptation path and induces Parametric Memory, creating a second memory channel that complements attention.
- We show that this memory-aware training improves long-context performance even when the fast weights are discarded at inference time, while re-enabling the inner loop at inference provides an additional prompt-specific adaptation effect that we interpret as *in-weight retrieval*.

2 Preliminaries and Motivation

Before formalizing our method, we review the standard training and inference paradigms of Masked Diffusion Language Models (MDLMs) [34, 36] and identify two limitations of this paradigm.

2.1 Preliminaries: Masked Diffusion Language Models

Consider a sequence of clean text comprising L tokens, denoted as $x_0 = (x_0^1, \ldots, x_0^L)$, where each token belongs to a discrete vocabulary $\mathcal{V}$. Discrete diffusion models operate by defining a forward corruption process that gradually introduces noise over a continuous time variable $t \in [0, 1]$. At $t = 0$, the sequence is completely clean (x_0), and at $t = 1$, the sequence reaches a state of pure noise (x_1). The model is then trained to approximate the reverse generative process, learning to map a noisy state x_t back to the original text x_0.

Absorbing-State Masking. In the specific framework of MDLMs, the forward corruption $q(x_t|x_0)$ is instantiated as an absorbing-state process. Rather than transitioning tokens to random vocabulary items, tokens are replaced by a dedicated absorbing token, $m \notin \mathcal{V}$ (often denoted as `[MASK]`). Under a linear noise schedule, the probability that the i-th token is masked at time t is simply t:

$$q(x_t^i|x_0^i) = (1 - t)\mathbb{I}(x_t^i = x_0^i) + t\mathbb{I}(x_t^i = m), \tag{1}$$

where $\mathbb{I}(\cdot)$ denotes the indicator function.

Training via Static Masking. The objective of the neural network $p_\theta(x_0|x_t)$, parameterized by θ, is to reconstruct the clean tokens x_0 given the corrupted sequence x_t. Because unmasked tokens are perfectly preserved in the absorbing-state formulation, the model only needs to predict the identities of the tokens at the currently masked indices, $\mathcal{M}_t = \{i \mid x_t^i = m\}$.

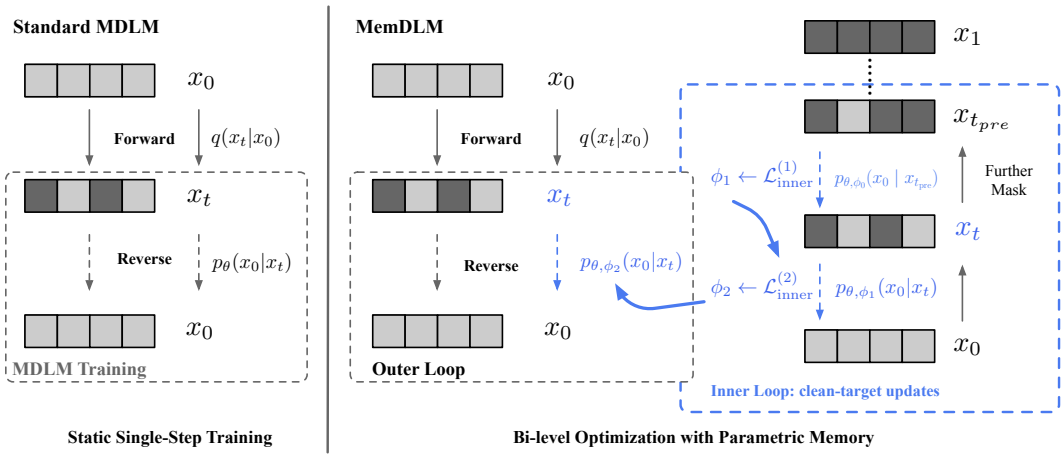

Figure 2: Overview of MemDLM. **Left:** standard MDLM training uses a static single-step denoising objective from x_t to x_0. **Right:** MemDLM uses Bi-level Optimization in which an inner loop reconstructs clean targets from nested pre-anchor and anchor states, carrying fast weights ϕ between the two updates. The outer loop updates the base model θ on the anchor state x_t conditioned on this parametric memory. **Legend:** dark tokens denote masks, light tokens denote observed tokens, gray arrows denote corruption or prediction, blue dashed arrows mark the inner-loop view sequence, and blue curved arrows denote fast-weight updates.

Standard MDLM training minimizes the expected negative log-likelihood of these masked tokens over uniformly sampled timesteps, yielding the following objective:

$$\mathcal{L}_{\text{MDLM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),x_0}\left[\omega(t)\sum_{i\in\mathcal{M}_t} -\log p_\theta(x_0^i|x_t)\right], \tag{2}$$

where $\omega(t)$ serves as a time-dependent weighting factor (e.g., $\omega(t) = 1/t$) to balance the loss across varying noise levels. Critically, Equation (2) represents a single-step, static masking objective: the model receives a masked text based purely on ground-truth data and is optimized to predict the clean sequence in one isolated step.

Inference via Iterative Denoising. In contrast, DLMs generate text during inference through a multi-step, progressive denoising trajectory. Starting from a fully masked sequence at $t = 1.0$, the model predicts the clean tokens. A subset of the highest-confidence predictions is then unmasked to form a partially noisy intermediate sequence $x_{t-\Delta t}$. This process repeats iteratively until $t = 0$, where all tokens are decoded. Crucially, at each step, the model's input is conditioned on its own noisy predictions from previous steps, rather than pristine ground-truth context.

2.2 Motivation

Standard MDLM training has two limitations.

Train-inference mismatch. The static single-step objective in Equation (2) optimizes an isolated corruption, while inference processes a sequence of partially revealed states. A further source of exposure bias is that inference states contain the model's own predictions [10, 50, 15, 32]. MemDLM introduces sequential fast-state adaptation across correlated noise levels during training. Its training states remain ground-truth corruptions; robustness to self-generated states is evaluated separately in Appendix C.

Attention dilution in long contexts. A second, complementary challenge arises as context length grows. Dense softmax attention must distribute probability mass across all tokens in the sequence. As the number of distractor tokens increases, maintaining attention on task-relevant information can become more difficult [23, 56, 17, 52]. This motivates exploring a complementary location for sample-specific information beyond token-space representations.

These observations motivate a second memory channel alongside token-space attention. Writing local denoising experience into fast weights exposes the base update to a stateful adaptation path and offers a parameter-space route for sample-specific information. We investigate this design through

long-context performance and component ablations; attention offloading provides an interpretation of its complementary role.

3 Methodology

Motivated by the limitations identified in Section 2, we introduce a second memory channel that complements token-space attention. **MemDLM** embeds an anchor-consistent multi-noise adaptation path into training via a Bi-level Optimization framework that naturally induces Parametric Memory.

3.1 Bi-level Optimization for Denoising Adaptation

To align the training objective with the iterative nature of inference, we partition the model parameters into the base weights θ and a set of parameter-efficient fast weights ϕ (e.g., low-rank adapters). We formulate the training process as a Bi-level Optimization problem:

$$\min_{\theta} \quad \mathbb{E}_{t\sim\mathcal{U}(0,1),x_0}\left[\omega(t)\sum_{i\in\mathcal{M}_t} -\log p_{\theta,\phi_K}(x_0^i \mid x_t)\right], \tag{3}$$

$$\text{subject to} \quad \phi_k = \phi_{k-1} - \eta\nabla_\phi \mathcal{L}_{\text{inner}}^{(k)}(\theta,\phi_{k-1}) \quad \text{for } k=1,\dots,K. \tag{4}$$

Here, Equation (4) represents the inner loop, which updates a transient fast state across K correlated denoising views of the current training batch. The effective LoRA contribution is reset to zero before adaptation, using random initialization for the first low-rank factor and zero initialization for the second. The resulting state ϕ_K acts as a *Parametric Memory* of local denoising experience. Equation (3) updates the base model θ conditioned on this memory.

3.2 The Inner Loop: Anchor-Consistent Trajectories

Rather than applying an arbitrary sequence of masks, we design an **Anchor-Consistent Local Trajectory** of denoising views and fast-weight updates. Because the outer objective is computed exactly at the noisy state x_t, the inner loop explicitly processes this same anchor state. This kind of masked inner-loop refinement is especially natural for DLMs: bidirectional denoising lets the model aggregate information from all visible tokens while updating multiple masked positions in a single step, whereas comparable hole-filling supervision is less direct under standard left-to-right auto-regressive factorization.

We use two inner updates ($K=2$). In the first stage (**Pre-Anchor Alignment**), we construct a noisier state $x_{t_{\text{pre}}}$ by further masking x_t, so that $\mathcal{M}_t \subseteq \mathcal{M}_{t_{\text{pre}}}$. The model reconstructs clean tokens from this pre-anchor view. In the second stage (**Anchor-to-Target**), it reconstructs clean tokens from the exact anchor state x_t using the adapted fast weights.

The fast weights accumulate the trajectory dynamics through the following sequence of updates:

$$\mathcal{L}_{\text{inner}}^{(1)} = \omega(t_{\text{pre}})\sum_{i\in\mathcal{M}_{t_{\text{pre}}}} -\log p_{\theta,\phi_0}(x_0^i \mid x_{t_{\text{pre}}}), \qquad \phi_1 = \phi_0 - \eta\nabla_\phi\mathcal{L}_{\text{inner}}^{(1)}, \tag{5}$$

$$\mathcal{L}_{\text{inner}}^{(2)} = \omega(t)\sum_{i\in\mathcal{M}_t} -\log p_{\theta,\phi_1}(x_0^i \mid x_t), \qquad \phi_2 = \phi_1 - \eta\nabla_\phi\mathcal{L}_{\text{inner}}^{(2)}, \tag{6}$$

where η is the inner learning rate and the losses use the same token normalization as standard MDLM. Both stages supervise clean targets at currently masked positions. The anchor-token-only ablation restricts the first loss to $\mathcal{M}_{t_{\text{pre}}} \setminus \mathcal{M}_t$. The local trajectory is carried by the evolving fast state ϕ_0, ϕ_1, ϕ_2 across nested ground-truth corruptions. Algorithm 1 summarizes the training procedure.

3.3 The Outer Loop: Conditioned Denoising

After the inner loop accumulates the adapted parameters ϕ_2 for a given batch, the outer objective is computed on the exact same anchor timestep t and masked state x_t. The full outer objective mirrors standard MDLM training, but conditions the prediction on the Parametric Memory ϕ_2:

$$\mathcal{L}_{\text{MemDLM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1),x_0}\left[\omega(t)\sum_{i\in\mathcal{M}_t} -\log p_{\theta,\phi_2}(x_0^i \mid x_t)\right]. \tag{7}$$

To update the base parameters θ, we employ a First-Order approximation. This avoids the computationally prohibitive calculation of second-order Hessian matrices by treating the inner gradients $\nabla_\phi \mathcal{L}_{\text{inner}}$ as independent of θ during the outer backward pass. For a given training batch, the update rule for the base model is computed using the per-sample loss:

$$\theta \leftarrow \theta - \beta \nabla_\theta \left(\omega(t) \sum_{i \in \mathcal{M}_t} -\log p_{\theta,\phi_2}(x_0^i \mid x_t) \right), \tag{8}$$

where β is the outer learning rate. The adapted fast state provides batch-specific context for the outer update. Training in this way changes the learned base weights, as demonstrated by the long-context gains retained in the *Train-Only* setting. At deployment, this setting uses the base model directly. *Train & Inference* instead adapts a fresh fast state by masking and reconstructing the known prompt before generation, providing an optional prompt-specific refinement.

Algorithm 1 MemDLM training step

Require: Clean prompt–response batch x_0, base parameters θ

1: Sample anchor timestep t and response mask $\mathcal{M}_t$; construct x_t
2: Set $t_{\text{pre}} = \min(1, 1.5t)$ and add response masks to form $x_{t_{\text{pre}}}$
3: Reset the transient LoRA factors so their effective contribution is zero
4: Compute the clean-token pre-anchor loss using Equation (5)
5: Update only the fast parameters, giving ϕ_1
6: Compute the clean-token anchor loss using Equation (6)
7: Update only the fast parameters, giving ϕ_2
8: Hold ϕ_2 fixed and compute the outer loss on the same anchor x_t
9: Update the persistent base LoRA using Equation (8)

4 Experiments

We study four questions: whether MemDLM improves long-context retrieval and generalization, which training-stage choices matter, how the inference-stage adaptation should be used, and which components are essential.

4.1 Experimental Setup

We implement MemDLM in PyTorch [28] on top of `dllm` [62] and evaluate with `lm-evaluation-harness` [8]. Main experiments use `LLaDA-MoE-7B-A1B-Base` [63] and `LLaDA2.1-mini` [4], abbreviated as **LLaDA-MoE** and **LLaDA2.1**; unless noted otherwise, targeted analyses and ablations use LLaDA-MoE. We instruction-tune on LongAlpaca [6] with asymmetric masking, use LoRA [14] for both outer- and inner-loop adaptation, and evaluate on RULER [13], BABILong [16], and LongBench [3]. All compared methods use one-token-per-step decoding within each backbone's generation schedule. Full setup details are in Appendix A; benchmark descriptions are in Appendix B.

4.2 Main Results: Long-Context Information Retrieval

Needle-in-a-Haystack retrieval is challenging for DLMs because the model must preserve a small amount of task-relevant information across long token-space contexts. We evaluate RULER (Multi-Value, Variable Tracking, and Common Words Extraction) and BABILong.

As shown in Table 1, MemDLM consistently improves over Standard MDLM across both backbones, especially at longer contexts. Crucially, the *Train-Only* variant already yields large gains, indicating that the main benefit is induced during training rather than coming solely from re-running the inner loop at inference. For example, on LLaDA-MoE, MemDLM improves RULER Variable Tracking at 8K from 78.8% to 95.8%, while on LLaDA2.1 it improves BABILong at 8K from 47.4% to 57.0%. Re-enabling the inner loop at inference then provides an additional prompt-specific adaptation effect, which we interpret as *in-weight retrieval*.

Length extrapolation via Parametric Memory. To probe robustness beyond the native 8K setting, we evaluate LLaDA-MoE at 16K and 32K. As shown in Table 2, performance drops for all methods as

Table 1: Needle-in-a-Haystack results on RULER and BABILong. We report Standard MDLM, MemDLM (Train-Only), and MemDLM (Train & Inference). Bold indicates the best result within each backbone block.

Backbone	Method	RULER-MV		RULER-VT		RULER-CWE		BABILong		
		4K	8K	4K	8K	4K	8K	2K	4K	8K
LLaDA-MoE	Standard MDLM	87.90	92.50	99.40	78.84	69.10	55.18	46.00	45.60	41.80
	MemDLM (Train-Only)	95.85	96.95	99.52	94.84	75.72	58.20	59.40	50.60	45.60
	MemDLM (Train & Inference)	**96.80**	**97.00**	**99.60**	**95.80**	**83.70**	**61.96**	**61.20**	**51.20**	**47.00**
LLaDA2.1	Standard MDLM	98.65	98.00	59.32	43.72	72.34	67.08	56.00	48.80	47.40
	MemDLM (Train-Only)	99.65	99.00	88.56	59.15	78.96	68.02	63.50	58.50	55.00
	MemDLM (Train & Inference)	**100.00**	**99.90**	**90.16**	**60.72**	**80.96**	**69.12**	**65.20**	**61.20**	**57.00**

Table 2: Length extrapolation on Needle-in-a-Haystack tasks with the LLaDA-MoE backbone. MemDLM continues to outperform Standard MDLM at 16K and 32K across RULER and BABILong.

Method	RULER-MV		RULER-VT		RULER-CWE		BABILong	
	16K	32K	16K	32K	16K	32K	16K	32K
Standard MDLM	22.35	14.30	52.56	9.48	44.20	17.82	19.20	6.80
MemDLM (Train-Only)	25.48	15.20	55.30	10.35	54.25	22.48	21.00	8.50
MemDLM (Train & Inference)	**29.40**	**15.35**	**56.84**	**11.44**	**57.24**	**24.48**	**22.20**	**9.00**

context grows, but MemDLM continues to improve over the baseline even in this extrapolation regime, suggesting that Parametric Memory helps preserve useful long-context representations beyond the lengths emphasized during training.

4.3 Long-Context Generalization

On LongBench, Parametric Memory improves zero-shot *Train-Only* performance, mirroring the NIAH results. Re-enabling the inner loop in *Train & Inference* yields consistent further gains, showing that prompt-specific adaptation complements the training-time benefit (Table 3).

Supplementary evaluations on Dream [55] extend the long-context gains to a third backbone. On GSM8K, MATH-500, HumanEval, and MBPP, the LLaDA-MoE Train-Only model maintains the performance range of Standard MDLM across three training seeds. Full results are reported in Appendix G.

4.4 Understanding MemDLM During Training

Figure 4 tracks the optimization objectives on training and evaluation splits. MemDLM's curves measure the memory-conditioned objective after response-label adaptation; downstream *Train-Only* evaluation uses the resulting base model without adaptation. Compared with the untuned `LLaDA-MoE-7B-A1B-Base` model, Standard MDLM fine-tuning lowers performance at 1K and 2K while improving at longer contexts. MemDLM consistently outperforms both models across 1K–32K (Figure 3).

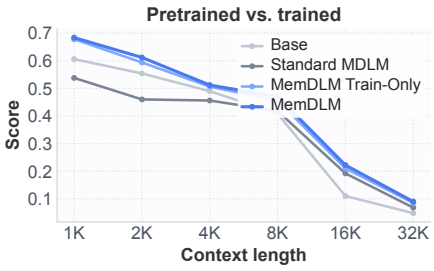

Figure 3: Comparison with the untuned pretrained LLaDA-MoE-7B-A1B-Base model across context lengths.

Inner-loop supervision. We compare the default cross-entropy objective with logit self-distillation using Kullback-Leibler (KL) [11] or reverse-KL divergence, and hidden-state self-distillation using cosine or MSE losses. Teacher and student share the model and current fast-weight state; the teacher runs under `no_grad` and, in the progressive setting, sees a later denoising state with more revealed context. All choices remain trainable (Figure 5), so memory writing is not tied to a single loss. Cross-entropy performs best on BABILong-1K (0.684), ahead of KL (0.660), reverse-KL (0.624), cosine (0.582), and MSE (0.572).

Table 3: LongBench results on LLaDA-MoE. **Train-Only** uses Parametric Memory during training but disables it at inference; **Train & Inference** reactivates the inner loop on the prompt.

Method	TriviaQA	PR-en	PR-zh	MF-en	MF-zh	2WikiQA	DuReader	MultiNews	TREC	SAMSum	LCC	RB-P
Standard MDLM	55.29	50.32	74.50	29.22	42.47	21.85	22.13	23.52	70.00	13.59	62.55	54.65
MemDLM (Train-Only)	87.74	54.36	86.29	31.44	42.87	22.43	22.19	26.35	70.50	15.80	64.20	55.05
MemDLM (Train & Inference)	**87.77**	**54.69**	**87.38**	**31.97**	**43.28**	**22.61**	**22.34**	**26.70**	**71.00**	**16.14**	**64.25**	**55.23**

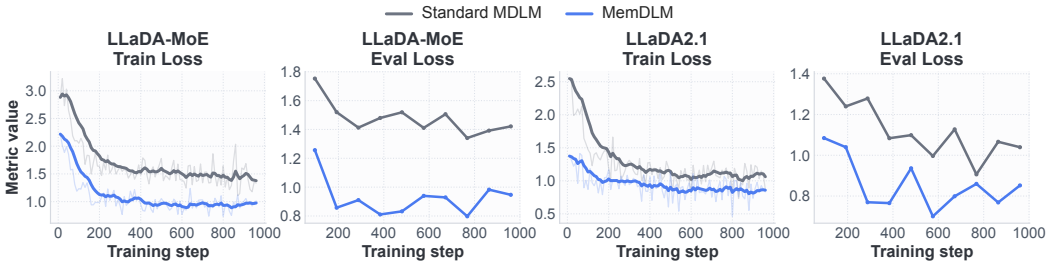

Figure 4: Training-objective dynamics on LLaDA-MoE and LLaDA2.1. Faint train-loss curves show raw values and bold curves show smoothed trends. MemDLM losses are measured after fast-weight adaptation, including response-label adaptation on the evaluation split.

Adaptation scope. Full-parameter updates achieve the lowest train loss but underperform restricted FFN-only updates downstream (Figure 6). Adapting FFN modules in the last 10% of layers yields the best BABILong-1K score (0.684), ahead of shallower adaptation (0.616 at 5%), broader adaptation (0.626 at 25%, 0.574 at 50%), FFN+attention updates at the same 10% scope (0.648), and full-parameter adaptation (0.602). Effective Parametric Memory depends not only on capacity, but also on where the update is written.

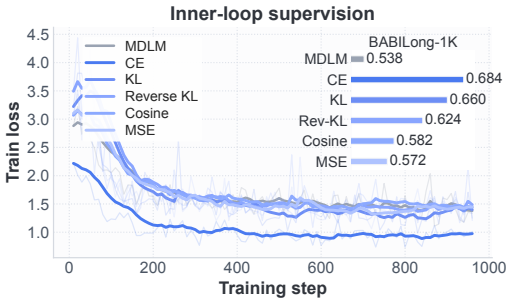

Figure 5: Inner-loop supervision analysis on the LLaDA-MoE, evaluated on BABILong-1K.

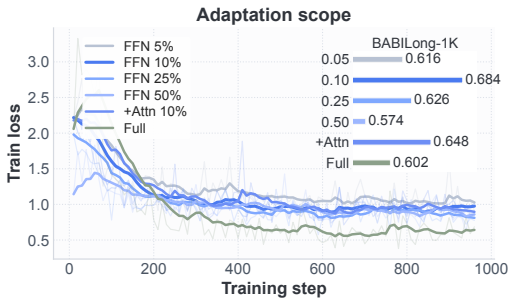

Figure 6: Adaptation scope analysis on the LLaDA-MoE, evaluated on BABILong-1K.

Robustness along denoising. We examine robustness to intermediate states produced by the model's own predictions. At each mask ratio, we compare clean-token losses on ground-truth-corrupted states ($\mathcal{L}_{\text{static}}$) and self-generated states reached by iterative unmasking ($\mathcal{L}_{\text{seq}}$). Figure 7 plots their ratio $\mathcal{R}_{\text{EB}} = \mathcal{L}_{\text{seq}}/\mathcal{L}_{\text{static}}$, normalized by each model's static loss. Standard MDLM reaches 27.54 at the final evaluation point, compared with 10.04 for MemDLM *Train-Only* and 9.22 for *Train & Inference*. The Train-Only improvement shows that the learned base model incurs less relative loss degradation along self-generated denoising paths. The full protocol is given in Appendix C.

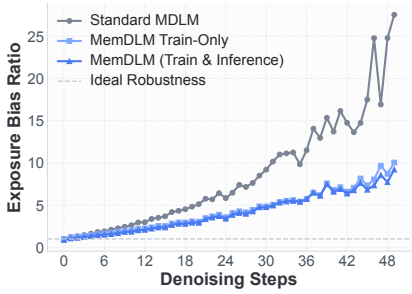

Figure 7: Sequential/static loss ratio across denoising steps. MemDLM shows lower relative loss degradation on self-generated states.

Further analyses of gradient normalization are reported in Appendix D.

4.5 Understanding MemDLM During Inference

Although the main effect of MemDLM appears during training, inference still introduces meaningful design choices. Here we study how the inner loop should be used at inference time and how sensitive it is to the synthetic anchor construction. Train-Only uses the standard decoder with no adaptation overhead. Train & Inference performs two prompt-side fast updates before generation, adding pre-generation latency while keeping the denoising schedule unchanged.

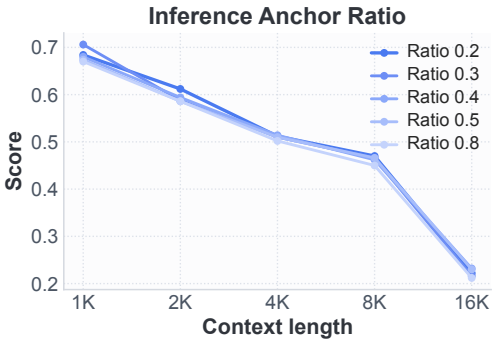

Figure 8: **Sensitivity to the inference anchor ratio.** We vary the target mask ratio of the adapted prompt state on LLaDA-MoE and evaluate from 1K to 16K. All settings follow a similar trend across context lengths.

At inference time, the anchor state is a design choice, which we parameterize by the target mask ratio of the adapted prompt state. Figure 8 shows that the method is relatively insensitive to this hyperparameter: ratios from 0.2 to 0.8 exhibit the same qualitative trend as context length increases, and their scores remain close throughout the full 1K–16K range. Even at 16K, results stay tightly grouped between 0.212 and 0.232. We therefore use 0.2 as the default not because it is uniquely optimal, but because it is a simple operating point in a fairly flat design space.

The bidirectional denoising objective supplies a prompt-reconstruction signal across this range of mask ratios. Changing the ratio changes the division between visible context and clean-token supervision, while preserving the prompt-specific nature of the update.

4.6 Ablation of Core Design Choices

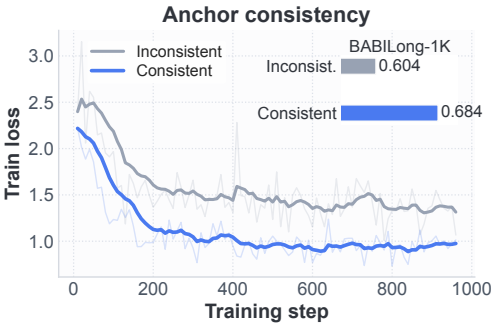

Figure 9: **Trajectory consistency.** Training loss for consistent and progressive-memory variants.

Consistency of the trajectory design. MemDLM's inner loop should remain consistent with the anchor-centered outer objective. Figure 9 compares our default design against an inconsistent progressive-memory variant and shows a clear optimization gap: the consistent trajectory converges to substantially lower training loss, and this carries over to downstream retrieval, improving BABILong-1K from 0.604 to 0.684. Trajectory consistency is therefore a core ingredient rather than an implementation detail.

Role of the two inner-loop stages. We ablate the two-stage inner loop by using only the pre-anchor stage or only the anchor-to-target stage. Figure 10 shows that combining the stages gives the strongest result: anchor-to-target only reaches 0.646, while pre-anchor only with anchor-token-only supervision reaches 0.620. With both stages active, restricting pre-anchor supervision to newly revealed anchor positions reaches 0.668, while the default clean-target supervision over all pre-anchor-masked positions reaches 0.684. Both variants use clean-token targets, with the broader loss providing a stronger first-stage update.

Multiple pre-anchor steps. Finally, we explore whether multiple pre-anchor steps further improve performance. Figure 11 shows a clear divergence between inner-loop optimization and downstream utility: increasing the number of pre-anchor steps from the default 2-step design to 3-step and 4-step variants steadily lowers the training loss, but the final BABILong-1K score drops from 0.684 to 0.644 and then to 0.590. The current two-stage design therefore appears sufficient for capturing the local trajectory information that matters, and lower inner-loop loss alone is not a reliable proxy.

Table 4: Supplementary training controls on LLaDA-MoE. Scores are percentages, reported as mean $\pm$ standard deviation across three training seeds. All methods disable fast weights at inference. Wall times are from the representative runs on 8 H200 GPUs.

Method	Steps	Time (h)	BABILong-2K	RULER-VT-8K
Standard MDLM (time-matched)	1,960	0.97	50.8 ± 0.9	84.57 ± 0.44
Trajectory-Only	960	0.93	54.9 ± 0.4	89.69 ± 0.32
MemDLM (Train-Only)	960	0.97	$\mathbf{59.3 \pm 0.6}$	$\mathbf{94.57 \pm 0.33}$

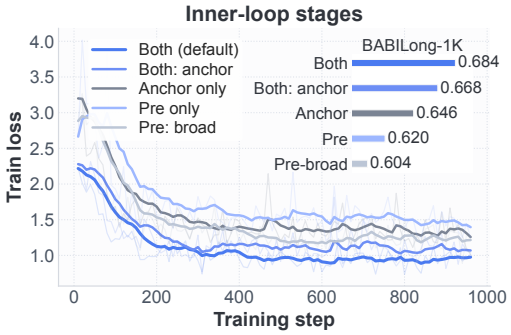

Figure 10: **Role of the two inner-loop stages.** Using both stages is consistently better than using either stage alone, and the default two-stage design with broader clean-target pre-anchor supervision outperforms the anchor-token-only alternative.

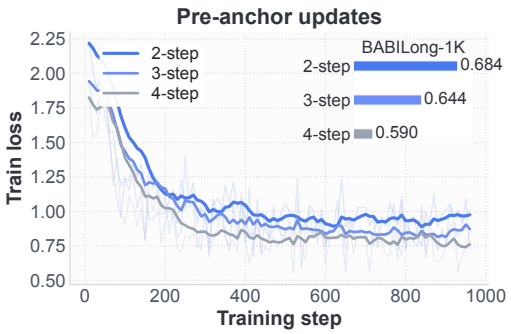

Figure 11: **Multiple pre-anchor steps.** Although additional pre-anchor steps reduce the inner-loop training loss, they consistently hurt downstream retrieval performance.

Pre-anchor design. We also investigate the pre-anchor state $x_{t_{\text{pre}}}$. In the anchor-consistent setting, its mask ratio is controlled by a pre-anchor scale hyperparameter s_{pre}, which sets the starting ratio as $\min(1, \max(s_{\text{pre}} \cdot t, t))$ for anchor mask ratio t. A scale of 1.5 performs best (0.684), while nearby values 1.75 and 2.0 remain competitive (0.674 and 0.678); a smaller scale 1.25 performs worse (0.624). The design is therefore meaningful but not highly fragile once the pre-anchor state is sufficiently noisier than the anchor.

Training budget and fast-weight conditioning. We complement these design ablations with a wall-clock-matched Standard MDLM baseline and a *Trajectory-Only* control. The latter uses the same nested pre-anchor and anchor states and optimizes the mean of their clean-token MDLM losses directly in the outer LoRA, with 960 outer updates and no fast state. Table 4 reports the results over three training seeds. MemDLM retains gains of $8.5/10.00$ percentage points over matched Standard MDLM and $4.4/4.88$ points over Trajectory-Only on BABILong-2K/RULER-VT-8K. These results support the contribution of fast-weight conditioning alongside multi-noise supervision. The training overhead and single-run controls are detailed in Appendices E and G.

5 Related Work

MemDLM lies at the intersection of diffusion language modeling, fast-weight memory, bi-level adaptation, and inference-time adaptation.

Diffusion language models and the training-inference gap. Recent diffusion language models support high-quality generation and flexible infilling [1, 34, 19, 36, 26, 55, 60, 5, 39, 20, 59, 48]. MDPO trains on model-generated denoising trajectories with policy gradients and intermediate rewards [10]; related RL methods also optimize denoising as sequential decision-making [50, 15]. Planner-alignment methods use confidence or self-planning signals [32]. EvoToken-DLM replaces hard masks with evolving soft token distributions and continuous trajectory supervision [61]. MemDLM retains hard-mask diffusion and learns through fast-weight adaptation across supervised denoising views.

Long-context attention and its limitations. Long-context attention has been studied from several angles [54]: Scalable-Softmax uses a length-aware normalization [23]; DySCO rescales attention during decoding [56]; attention sinks reveal disproportionate mass on early tokens [52]; and positional analyses identify retrieval failures in the middle of long contexts [17]. MemDLM complements these attention-level approaches with a parameter-space adaptation channel. Offloading contextual information into fast weights offers an interpretation of this complementary design.

Fast weights and parametric memory. The idea that neural networks can store short-lived information in parameters has a long history [44, 2, 12, 58]. Related adaptation methods show that local weight updates can enable rapid learning from context [38, 41, 21, 22, 47, 49, 27]. MetaState equips a frozen DLM with persistent recurrent activation slots across decoding steps [51]. MemDLM uses transient LoRA parameter states and updates the base model through a memory-conditioned outer objective; most gains remain when the transient state is absent at inference.

Meta-learning and Bi-level Optimization. MemDLM relates to meta-learning methods that combine inner-loop adaptation with an outer-loop objective [43, 7, 25, 45, 37, 35, 33, 9, 31]. Our inner loop internalizes local denoising experience within a training batch, and the outer loop learns base parameters under the resulting fast state. This structure connects rapid adaptation with base-model learning.

Test-time training. Finally, MemDLM is related to test-time training methods that update model behavior on the fly using unlabeled or self-supervised signals [40, 53, 30, 46, 57, 64, 42, 65]. TTT-E2E frames long-context modeling as continual test-time learning, using the same next-token objective at training and deployment time so that incoming context can be compressed into the weights during inference [42]. SEAL studies self-adapting language models that generate their own update directives or synthetic supervision and then perform persistent weight updates under a reward-driven loop [65]. This connection is most visible when we re-enable the inner loop at inference time, but our results show that the main gains already emerge from memory-aware training with inference-time adaptation providing only an additional prompt-specific refinement.

6 Limitations

MemDLM introduces extra training overhead from the inner loop ($\sim 2\times$ wall-clock under the default configuration; see Appendix E), which we mitigate with parameter-efficient adapters and restricted update scope. Our study covers three DLM backbones and one instruction-tuning dataset; extending memory-aware optimization to pretraining and longer training contexts remains a promising direction. The three-seed supplementary evaluations characterize run-to-run variation in the targeted comparisons. Inference-time adaptation also adds prompt-side latency, which may be undesirable in latency-critical settings [29]. Extending the inner loop beyond prompt adaptation and into decoding is left for future work.

7 Conclusion

We introduced **MemDLM**, a memory-aware training framework for diffusion language models based on Bi-level Optimization and fast weights as Parametric Memory. Our main finding is that training through a local fast-weight adaptation path changes what the base model learns: a parameter-space memory channel complements token-space attention and strengthens long-context performance even in the *Train-Only* setting. Re-enabling the inner loop at inference provides an additional prompt-specific adaptation effect that we interpret as *in-weight retrieval*. These results establish memory-aware training as a promising direction for long-context DLMs.

Acknowledgments and Disclosure of Funding

Funding. No external funding was received.

Competing Interests. The authors declare no competing interests.

References

[1] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.
[2] J. Ba, G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.
[3] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137, 2024.
[4] T. Bie, M. Cao, X. Cao, B. Chen, F. Chen, K. Chen, L. Du, D. Feng, H. Feng, M. Gong, et al. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.
[5] A. Campbell, J. Benton, V. De Bortoli, T. Rainforth, G. Deligiannidis, and A. Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.
[6] Y. Chen, S. Yu, S. Qian, H. Tang, X. Lai, Z. Liu, S. Han, and J. Jia. Long alpaca: Long-context instruction-following models. https://github.com/dvlab-research/LongLoRA, 2023.
[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.
[8] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. The language model evaluation harness, 07 2024.
[9] S. Garg, D. Tsipras, P. S. Liang, and G. Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in neural information processing systems*, 35:30583–30598, 2022.
[10] H. He, K. Renz, Y. Cao, and A. Geiger. MDPO: Overcoming the training-inference divide of masked diffusion language models. *arXiv preprint arXiv:2508.13148*, 2025.
[11] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
[12] G. E. Hinton and D. C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.
[13] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, Y. Zhang, and B. Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.
[14] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
[15] Z. Huang, Z. Chen, Z. Wang, T. Li, and G.-J. Qi. Reinforcing the diffusion chain of lateral thought with diffusion language models. *arXiv preprint arXiv:2505.10446*, 2025.
[16] Y. Kuratov, A. Bulatov, P. Anokhin, I. Rodkin, D. Sorokin, A. Sorokin, and M. Burtsev. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems*, 37:106519–106554, 2024.
[17] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.
[18] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
[19] A. Lou, C. Meng, and S. Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. *arXiv preprint arXiv:2310.16834*, 2023.
[20] C. Meng, K. Choi, J. Song, and S. Ermon. Concrete score matching: Generalized score matching for discrete data. *Advances in Neural Information Processing Systems*, 35:34532–34545, 2022.
[21] K. Meng, A. S. Sharma, A. Andonian, Y. Belinkov, and D. Bau. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*, 2022.
[22] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.
[23] K. M. Nakanishi. Scalable-softmax is superior for attention. *arXiv preprint arXiv:2501.19399*, 2025.
[24] NeoHorse Team, G. Cao, G. Dai, T. Guo, K. Han, H. Hu, Z. Jiang, X. Kuang, B. Li, Y. Li, et al. NeoHorse-1: Towards recursive self-improvement via agentic post-training with routing harness. *arXiv preprint arXiv:2609.08183*, 2026.
[25] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.
[26] J. Ou, S. Nie, K. Xue, F. Zhu, J. Sun, Z. Li, and C. Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

[27] S. Padmanabhan, Y. Onoe, M. Zhang, G. Durrett, and E. Choi. Propagating knowledge updates to lms through distillation. *Advances in Neural Information Processing Systems*, 36:47124–47142, 2023.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[29] Z. Pei, Y. Zhang, H.-L. Zhen, T. Yuan, X. Yu, Z. Dong, S. J. Pan, M. Yuan, and B. Yu. PreMoE: Proactive inference for efficient mixture-of-experts. *arXiv preprint arXiv:2505.17639*, 2025.

[30] Z. Pei, H.-L. Zhen, S. Kai, S. J. Pan, Y. Wang, M. Yuan, and B. Yu. Scope: Prompt evolution for enhancing agent effectiveness. *arXiv preprint arXiv:2512.15374*, 2025.

[31] Z. Pei, H.-L. Zhen, X. Yu, S. J. Pan, M. Yuan, and B. Yu. FocuSFT: Bilevel optimization for dilution-aware long-context fine-tuning. *arXiv preprint arXiv:2605.09932*, 2026.

[32] F. Z. Peng, Z. Bezemek, J. Rector-Brooks, S. Zhang, A. R. Zhang, M. Bronstein, A. J. Bose, and A. Tong. Planner aware path learning in diffusion language models training. *arXiv preprint arXiv:2509.23405*, 2025.

[33] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

[34] S. S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. T. Chiu, A. Rush, and V. Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

[35] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[36] J. Shi, K. Han, Z. Wang, A. Doucet, and M. Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

[37] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

[38] P. Sprechmann, S. M. Jayakumar, J. W. Rae, A. Pritzel, A. P. Badia, B. Uria, O. Vinyals, D. Hassabis, R. Pascanu, and C. Blundell. Memory-based parameter adaptation. *arXiv preprint arXiv:1802.10542*, 2018.

[39] H. Sun, L. Yu, B. Dai, D. Schuurmans, and H. Dai. Score-based continuous-time discrete diffusion models. *arXiv preprint arXiv:2211.16750*, 2022.

[40] Y. Sun, X. Wang, Z. Liu, J. Miller, A. Efros, and M. Hardt. Test-time training with self-supervision for generalization under distribution shifts. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9229–9248. PMLR, 13–18 Jul 2020.

[41] J. Tack, J. Kim, E. Mitchell, J. Shin, Y. W. Teh, and J. R. Schwarz. Online adaptation of language models with a memory of amortized contexts. *Advances in Neural Information Processing Systems*, 37:130109–130135, 2024.

[42] A. Tandon, K. Dalal, X. Li, D. Koceja, M. Rød, S. Buchanan, X. Wang, J. Leskovec, S. Koyejo, T. Hashimoto, et al. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025.

[43] S. Thrun and L. Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998.

[44] T. Tieleman and G. Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th annual international conference on machine learning*, pages 1033–1040, 2009.

[45] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[46] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021.

[47] Y. Wang, Y. Gao, X. Chen, H. Jiang, S. Li, J. Yang, Q. Yin, Z. Li, X. Li, B. Yin, J. Shang, and J. J. McAuley. MEMORYLLM: towards self-updatable large language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[48] Y. Wang, K. Han, H. Zhen, Y. Tian, H. Chen, Y. Huang, Y. Cui, Y. Shu, S. Gao, I. Elezi, et al. Top 10 open challenges steering the future of diffusion language model and its variants. *arXiv preprint arXiv:2601.14041*, 2026.

[49] Y. Wang, X. Liu, X. Chen, S. O'Brien, J. Wu, and J. McAuley. Self-updatable large language models by integrating context into model parameters. *arXiv preprint arXiv:2410.00487*, 2024.

[50] Y. Wang, L. Yang, B. Li, Y. Tian, K. Shen, and M. Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*, 2025.

[51] K. Xia, M. Li, L. Wei, Z. Du, X. Yuan, D. Shi, Q. Jin, and W. Lee. MetaState: Persistent working memory enhances reasoning in discrete diffusion language models. *arXiv preprint arXiv:2603.01331*, 2026.

[52] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[53] S. Xiong, O. Gungordu, J. C. Kerce, and F. Fekri. Adaptive information control for search-augmented LLM reasoning. *arXiv preprint arXiv:2602.01672*, 2026.

[54] S. Xiong, J. Zou, F. Fekri, and Y. J. Cho. Long-context modeling with dynamic hierarchical sparse attention for memory-constrained LLM inference. In *Forty-third International Conference on Machine Learning*, 2026.

[55] J. Ye, Z. Xie, L. Zheng, J. Gao, Z. Wu, X. Jiang, Z. Li, and L. Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.
[56] X. Ye, W. Zhang, F. Yin, H. Yen, and D. Chen. Dysco: Dynamic attention-scaling decoding for long-context lms. *arXiv preprint arXiv:2602.22175*, 2026.
[57] T. Zhang, S. Bi, Y. Hong, K. Zhang, F. Luan, S. Yang, K. Sunkavalli, W. T. Freeman, and H. Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025.
[58] T. Zhao and L. Jones. Fast-weight product key memory. *arXiv preprint arXiv:2601.00671*, 2026.
[59] H. Zhen, W. Lin, R. Liu, K. Han, Y. Li, Y. Tian, H. Chen, X. Li, X. Li, C. Chen, et al. Dllm agent: See farther, run faster. *arXiv preprint arXiv:2602.07451*, 2026.
[60] K. Zheng, Y. Chen, H. Mao, M.-Y. Liu, J. Zhu, and Q. Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.
[61] L. Zhong, L. Wu, B. Fang, T. Feng, C. Jing, W. Wang, J. Zhang, H. Chen, and C. Shen. Beyond hard masks: Progressive token evolution for diffusion language models. *arXiv preprint arXiv:2601.07351*, 2026.
[62] Z. Zhou, L. Chen, H. Tong, and D. Song. dllm: Simple diffusion language modeling, 2026.
[63] F. Zhu, Z. You, Y. Xing, Z. Huang, L. Liu, Y. Zhuang, G. Lu, K. Wang, X. Wang, L. Wei, et al. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*, 2025.
[64] Y. Zuo, K. Zhang, L. Sheng, S. Qu, G. Cui, X. Zhu, H. Li, Y. Zhang, X. Long, E. Hua, et al. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.
[65] A. Zweiger, J. Pari, H. Guo, E. Akyürek, Y. Kim, and P. Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025.

A Additional Experimental Details

Implementation and Baselines. We implement MemDLM in PyTorch [28] on top of the open-source `dllm` [62] training library and use `lm-evaluation-harness` [8] for downstream evaluation. We study two backbones in the main experiments: `LLaDA-MoE-7B-A1B-Base` [63] and `LLaDA2.1-mini` (16B) [4], abbreviated throughout the paper as **LLaDA-MoE** and **LLaDA2.1**. Unless otherwise noted, targeted training-stage analyses and core ablations use LLaDA-MoE. Supplementary experiments on Dream [55] are reported in Appendix G. Our baseline is **Standard MDLM** [34], which optimizes the standard denoising objective with time-dependent reweighting. Training and evaluation code are available at `https://github.com/JarvisPei/MemDLM`.

Training Data and Processing. We conduct instruction tuning on LongAlpaca [6], filtering the dataset to sequences of at most $4,096$ tokens for computational efficiency. During training, we apply an asymmetric masking strategy: prompt tokens remain strictly unmasked and are excluded from the loss, while the noise and masking processes are applied only to the response tokens.

Hyperparameters, optimization, and compute. For parameter efficiency, we load the base model in 4-bit quantization and apply Low-Rank Adaptation (LoRA) [14] for the outer loop with rank $r = 32$ and $\alpha = 64$. The outer loop is optimized using AdamW [18] with learning rate 2×10^{-5} and a cosine schedule with 0.1 warmup ratio. For the Parametric Memory mechanism, the inner loop uses a separate transient set of LoRA adapters with the same configuration ($r = 32, \alpha = 64$). To reduce overhead, the inner loop only updates Feed-Forward Network (FFN) modules in the final fraction of transformer layers. The inner-loop adaptation consists of a single epoch of SGD with learning rate 0.1 and gradient clipping set to 1.0.

All training and evaluation use 8 NVIDIA H200 GPUs. We train with batch size 2 per device for 5 epochs; under our default MemDLM setting, wall-clock training is approximately 1 hour for `LLaDA-MoE-7B-A1B-Base` and approximately 3 hours for `LLaDA2.1-mini`.

Evaluation benchmarks. We evaluate on **RULER** [13], **BABILong** [16], and **LongBench** [3]. Expanded descriptions, including the RULER sub-tasks we report, are given in Appendix B.

Generation and decoding settings. Decoding hyperparameters are chosen to match each backbone's reference practice while keeping denoising behavior consistent where noted. For **LLaDA-MoE**, we use *single-block* generation and set the output length according to each benchmark's requirements. For **LLaDA2.1**, we use *multi-block* generation. For *both* models, during iterative denoising we generate *one token per step*. Other shared evaluation options (e.g., via `lm-evaluation-harness`) are held fixed across compared methods for a given backbone.

B Details of evaluation benchmarks

RULER [13] is a *synthetic* benchmark for long-context models with configurable sequence length and task complexity. In the official benchmark, tasks are grouped into retrieval, multi-hop tracing, aggregation, and question-answering families. We report three representative sub-tasks used in our main table (see also the official RULER repository task table):

- **Multi-Value (MV).** A retrieval-style extension of needle-in-a-haystack in which multiple target values are embedded in long distractor text and the model must return the requested set of values, rather than only one needle.
- **Variable Tracking (VT).** The official RULER task `variable_tracking`, where the context contains chains of variable name-binding updates and the model must follow the chain to identify the final binding.
- **Common Words Extraction (CWE).** The official RULER task `common_words_extraction`, where the context is constructed with controlled word frequencies and the model must extract the words that occur commonly under the task specification.

BABILong [16] evaluates *reasoning-in-a-haystack*: it extends bAbI-style reasoning to long-context settings by scattering the relevant supporting facts through very long documents. The benchmark contains 20 reasoning tasks, including fact chaining, induction, deduction, counting, and list/set

manipulation, and is designed to test whether models can both retrieve and combine distant evidence as context length grows.

LongBench [3] is a *bilingual*, multi-task benchmark for realistic long-context understanding. Our main table (Table 3) follows the task names used in the LongBench suite and our evaluation harness: **TriviaQA**, **PR-en**, **PR-zh**, **MF-en**, **MF-zh**, **2WikiQA**, **DuReader**, **MultiNews**, **TREC**, **SAMSum**, **LCC**, and **RB-P** (RepoBench-P). Concretely, this subset covers question answering, passage retrieval, summarization, classification, and code-completion style long-context tasks in both English and Chinese.

C Empirical Analysis of Exposure Bias

We evaluate the trained models' robustness to self-generated intermediate states using a validation set of prompt-response pairs. For a given mask ratio corresponding to timestep t, we compare two conditions:

Static Condition: The model predicts masked tokens from a pristine context where the ground-truth response is artificially masked according to the true forward process:

$$\mathcal{L}_{\text{static}} = \mathbb{E}_{x_0, x_t \sim q(\cdot|x_0)} \left[-\log p_\theta(x_0|x_t) \right]. \tag{9}$$

Sequential Condition: Starting from a 100% masked response, the model iteratively predicts and unmasks tokens using its own predictions until reaching timestep t:

$$\mathcal{L}_{\text{seq}} = \mathbb{E}_{x_0, \hat{x}_t \sim p_\theta} \left[-\log p_\theta(x_0|\hat{x}_t) \right]. \tag{10}$$

We define the **Exposure Bias Ratio** as $\mathcal{R}_{\text{EB}} = \mathcal{L}_{\text{seq}} / \mathcal{L}_{\text{static}}$. This post-training diagnostic measures relative loss degradation on self-generated states, normalized by each model's static loss at the same mask ratio. Training uses the ground-truth-corrupted views described in Section 3.

Figure 7 shows a rising sequential/static loss ratio for Standard MDLM, reaching 27.54 at the final evaluation point. The ratio is 10.04 for MemDLM *Train-Only* and 9.22 for *Train & Inference*. The Train-Only result shows that memory-aware training improves the learned base model's relative robustness to self-generated states. Prompt adaptation provides a further refinement.

D Additional Training Ablations

Gradient normalization in the inner loop. Because the inner loop performs rapid task-local adaptation, its behavior can depend on gradient normalization. On LLaDA-MoE / BABILong-1K, local per-parameter normalization with clipping 1.0 achieves the best score (0.684), whereas global normalization degrades performance to 0.632. Varying the clipping threshold under local normalization has a weaker effect: clipping at 0.5 or 2.0 yields 0.630 and 0.640, while removing clipping remains competitive at 0.682. The key choice therefore appears to be *local* normalization rather than the exact clipping threshold.

Pre-anchor scaling results are reported in Section 4.6.

E Training Workload Analysis

Table 5 reports wall-clock training time and per-step throughput for Standard MDLM and MemDLM under the default configuration (2-step inner loop, FFN-only LoRA on the last 10% of layers), as well as selected inner-loop variants. The runs in this table use 8 NVIDIA H200 GPUs, batch size 2 per device, and 960 training steps (5 epochs). The time-matched controls use the budgets specified in Tables 4 and 6.

Under the default configuration, MemDLM adds approximately $2.1\times$ wall-clock overhead compared to Standard MDLM on both backbones. This overhead is consistent: training LLaDA-MoE takes $\sim$1 hour (vs. $\sim$0.5 hours for the baseline) and LLaDA2.1 takes $\sim$3.4 hours (vs. $\sim$1.6 hours). The per-step cost scales predictably with the number of parameters updated in the inner loop: restricting updates to FFN modules in the last 5–10% of layers keeps overhead near $2\times$, while full-parameter

Table 5: Training workload comparison. Wall-clock time and per-step cost are measured from TensorBoard logs. Overhead is relative to the Standard MDLM baseline of each backbone.

Backbone	Configuration	Wall Time (h)	s/step	Overhead	BABILong-1K
LLaDA-MoE	Standard MDLM	0.47	1.78	1.0×	0.538
	MemDLM (default)	0.97	3.67	2.1×	0.684
LLaDA2.1	Standard MDLM	1.60	6.08	1.0×	0.646
	MemDLM (default)	3.40	12.87	2.1×	0.706
Adaptation scope variants (LLaDA-MoE)					
	FFN, last 5% layers	0.92	3.51	2.0×	0.616
	FFN, last 10% layers (default)	0.97	3.67	2.1×	0.684
	FFN+Attention, last 10% layers	0.99	3.75	2.1×	0.648
	FFN, last 25% layers	1.09	4.14	2.3×	0.626
	FFN, last 50% layers	1.31	4.95	2.8×	0.574
	Full parameters	1.68	6.38	3.6×	0.602
Inner-loop stage variants (LLaDA-MoE)					
	Pre-anchor only (1-step)	0.81	3.08	1.7×	0.620
	Anchor-to-target only (1-step)	0.76	2.88	1.6×	0.646
	Both stages, 2-step (default)	0.97	3.67	2.1×	0.684
	Both stages, 4-step	1.46	5.55	3.1×	0.590

inner-loop updates raise it to $3.6\times$. Similarly, increasing the number of inner-loop steps from 2 to 4 increases overhead from $2.1\times$ to $3.1\times$. As shown in the main text, the default restricted configuration (FFN, last 10%, 2-step) achieves the best downstream performance, so the most effective setting is also the most efficient.

F Training and Deployment Procedure

Algorithm 1 gives the default training step. Prompt positions remain visible and are excluded from the response loss. The first update reconstructs clean labels from the pre-anchor view, and the second reconstructs the same clean labels from the anchor view. Inner gradients use per-parameter clipping at 1.0. The outer update treats the adapted fast state as fixed, implementing the first-order approximation.

At deployment, *Train-Only* removes the transient adapters and uses the standard decoder. *Train & Inference* resets a fresh fast state for each prompt, masks positions in that known prompt, and performs two clean-prompt reconstruction updates before generation. Both modes use the same denoising schedule. The latter adds prompt-side adaptation latency; it introduces no extra adaptation step inside decoding.

G Supplementary Validation

G.1 Training Controls and Three-Seed Replication

The Standard MDLM reference uses the training setup in Appendix A. Its wall-clock-matched variant extends training to the 0.97 h MemDLM budget while keeping the backbone, data, outer LoRA, and decoding configuration fixed. Trajectory-Only samples the same nested masks as MemDLM and minimizes $\frac{1}{2}(\mathcal{L}_{\text{pre}} + \mathcal{L}_{\text{anchor}})$ directly in the persistent outer LoRA. Both losses use clean targets, scheduler weights, and the Standard MDLM token normalization. There is no transient fast state in this control. It uses the same 960 outer updates as MemDLM.

Table 6 records the single-run comparisons. Table 4 reports their three-seed replication. All supplementary uncertainty values are standard deviations over three training seeds, measuring run-to-run variation; scores are reported as percentages. The main benchmark tables retain their reference-run results. MemDLM supplementary evaluations use *Train-Only*, with transient fast weights disabled at inference.

Table 6: Single-run training controls on LLaDA-MoE. All rows use the standard decoder without inference-time fast weights.

Method	Steps	Time (h)	BABILong-2K	RULER-VT-8K
Standard MDLM	960	0.47	46.00	78.84
Standard MDLM (time-matched)	1,960	0.97	51.00	85.00
Trajectory-Only	960	0.93	54.50	90.00
MemDLM (Train-Only)	960	0.97	**59.40**	**94.84**

G.2 Standard Reasoning and Coding Tasks

We evaluate the LLaDA-MoE models on GSM8K and MATH-500 using exact match, and HumanEval and MBPP using pass@1. Standard MDLM uses the original fine-tuning configuration in Appendix A; the extended-budget variant is explicitly labeled "time-matched" in the training-control tables. Table 7 shows that MemDLM remains within the baseline performance range across all four tasks, supporting preservation of general reasoning and coding capabilities after long-context instruction tuning.

Table 7: Standard-task performance on LLaDA-MoE, as mean $\pm$ standard deviation over three training seeds. Scores are percentages; HumanEval and MBPP use pass@1.

Method	GSM8K	MATH-500	HumanEval	MBPP
Standard MDLM	67.0 ± 0.7	36.7 ± 0.9	46.3 ± 1.2	53.0 ± 0.9
Trajectory-Only	67.2 ± 0.6	36.5 ± 0.7	$\mathbf{46.9 \pm 1.0}$	53.1 ± 0.9
MemDLM (Train-Only)	$\mathbf{67.5 \pm 0.5}$	$\mathbf{36.9 \pm 0.8}$	46.7 ± 0.9	$\mathbf{53.5 \pm 0.7}$

G.3 Additional Backbone: Dream

We further evaluate Dream [55] on BABILong-1K and RULER-CWE-1K. Table 8 reports matched training controls and three-seed results. MemDLM improves over the time-matched Standard MDLM by 6.5 and 6.2 percentage points and over Trajectory-Only by 4.0 and 3.9 points. The benefit therefore transfers beyond the two LLaDA backbones used in the main comparisons.

Table 8: Supplementary validation on Dream. Results are percentage scores, as mean $\pm$ standard deviation over three training seeds. Relative training time is measured against Standard MDLM on the same backbone.

Method	Relative time	BABILong-1K	RULER-CWE-1K
Standard MDLM	$1.0\times$	55.0 ± 1.4	82.3 ± 0.9
Standard MDLM (time-matched)	$\sim 2.1\times$	57.5 ± 1.1	84.3 ± 0.7
Trajectory-Only	$2.0\times$	60.0 ± 0.9	86.6 ± 0.7
MemDLM (Train-Only)	$2.1\times$	$\mathbf{64.0 \pm 1.2}$	$\mathbf{90.5 \pm 0.6}$

H Impact Statement

MemDLM improves long-context modeling through memory-aware training. Its potential benefits include more reliable use of long documents. Downstream applications inherit language-model risks associated with inaccurate generation, data bias, and sensitive information in prompts. Prompt-side adaptation should respect the access and retention policies of the deployment environment.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Abstract and intro state MemDLM, bi-level training, Parametric Memory, main empirical claims (e.g. NIAH / long-context), and scope; contributions align with Experiments and Method.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Discussion of limitations is present in the main text.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [N/A]

Justification: The paper is empirical; there are no formal theorems or proofs.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Method is described; training/eval pipeline is named (dllm, harness); hyperparameters and data are in main + Appendix.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: Training and evaluation code are public at https://github.com/JarvisPei/MemDLM; all evaluation datasets are public.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: Summarized in Experiments; full details in Appendix.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: The targeted training controls and supplementary reasoning, coding, and Dream evaluations report mean and standard deviation over three training seeds. The main benchmark tables report the reference runs. See Table 4 and Appendix G.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Summarized in Experiments; full details in Appendix.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Paper adheres to NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: The Impact Statement in Appendix H discusses long-document applications, generation errors, bias, and prompt-data handling.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper poses no such risks

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Code is open-source (dllm, harness); data are public (LongAlpaca, LongBench, RULER, BABILong).

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: The MemDLM repository provides training and evaluation code; the method and experimental configurations are described in the paper and appendix.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: No crowdsourcing or human subjects are involved.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: No human subjects are involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: The DLM backbones being studied are described in Section 4.1 and Appendix A. No auxiliary LLM annotator or judge is used in the method or evaluation.

    Guidelines:

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