Looped Diffusion Language Models

Sanghyun Lee
KAIST
lsh83210@kaist.ac.kr

Chunsan Hong
KAIST
hoarer@kaist.ac.kr

Seungryong Kim
KAIST
seungryong.kim@kaist.ac.kr

Jonghyun Lee
KRAFTON
jonghyunlee@krafton.com

Jongho Park
University of California, Berkeley
jjhpark@berkeley.edu

Dongmin Park[†]
KRAFTON
dongmin.park@krafton.com

Abstract

Masked diffusion models (MDMs) have emerged as a promising alternative to autoregressive models for language modeling, yet the effective design of transformer architectures for MDMs remains underexplored. In this paper, we show that selectively looping the early-middle transformer layers significantly improves both training efficiency and model performance in MDMs. We call this approach **LoopMDM** (Looped Masked Diffusion Model), which brings two key benefits: looping layers at training-time yields a depth-scaling effect without adding parameters, while varying the number of loops at inference-time enables flexible compute scaling. Despite the simplicity, the results are striking: across multiple pre-training corpora, LoopMDM matches the performance of same-size MDMs with up to $3.3\times$ fewer training FLOPs, while its final performance outperforms them on various reasoning benchmarks, including up to $+8.5$ points on GSM8K. It even surpasses deeper non-looped MDMs trained with comparable per-step compute, indicating that selective looping is more effective than naive depth scaling. Furthermore, LoopMDM can scale inference-time compute by increasing the number of loops. Adaptively adjusting the number of loops throughout the sampling process further yields additional gains in compute efficiency while maintaining performance. Lastly, with attention analysis, we provide evidence that looping is effective in MDMs by promoting interactions among masked positions.Our code is available at link.

1 Introduction

Masked diffusion models (MDMs) [2, 38, 59, 65] have emerged as a competitive alternative to autoregressive models (ARMs) for language modeling. Recent advances in training objectives [53], noise schedules [27], and architectures [7, 17] have steadily narrowed the gap with ARMs, establishing MDMs as an increasingly viable framework for text generation. Building on these improvements, recent work has further pushed MDMs toward larger-scale models, including the LLaDA family [47, 80, 4], Dream [76], and Seed Diffusion [66], to narrow the gap even further. Although these approaches demonstrate the promise of scaling, they also raise the question of how to improve performance along axes other than parameters and training tokens.

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

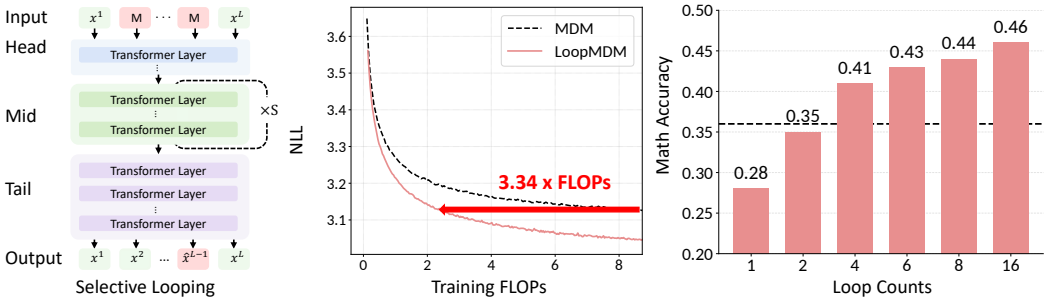


Figure 1: **Overview of LoopMDM.** (Left) LoopMDM selectively applies looping to a small early-middle layers of the denoising network. (Middle) Under matched training compute, LoopMDM (red) reaches the same test NLL as a non-looped MDM baseline with the same architecture (dashed black) using substantially fewer training FLOPs. (Right) Increasing the inference-time loop count consistently improves GSM8K accuracy; the dashed line denotes the same non-looped MDM baseline.

A natural candidate from the AR literature is the *looped transformer* [15, 20, 19, 81]. Looped architectures apply a shared block repeatedly to convert depth into loops at a fixed parameter cost. Looping has been explored in ARMs as a form of test-time compute scaling, but its role in MDMs has not been explored. It is unclear whether looping should help in this setting since MDMs already perform iterative computation across denoising steps. Moreover, previous works on looping have focused on parameter efficiency [63] or inference-time refinement [74, 18], leaving much to be desired regarding training-compute efficiency, as performance gain over non-looped models at fixed compute remains non-trivial [64].

Motivated by this gap, we study how to integrate looping into MDMs under matched training compute. Through a systematic study of where and how much to loop, we find that selectively looping a few early-middle transformer layers captures the full benefit of repeated computation at substantially lower training cost than uniform looping. We refer to this configuration as **LoopMDM**, the first application of looped transformers to masked diffusion language models. Figure 1 provides an overview of the architecture and its scaling behavior. Three observations summarize our findings:

Training compute efficiency. Across the three pretraining corpora, including FineWeb-Edu [51], OpenWebText [21], and LM1B [8], LoopMDM matches the test negative log-likelihood (NLL) of an equal-size MDM with up to $3.3\times$ fewer training floating-point operations (FLOPs). When trained at matched total training FLOPs, it consistently outperforms the baseline over various benchmarks in zero-shot perplexity, generative perplexity, and downstream task accuracy. Crucially, the observed efficiency gains depend strongly on *how* looping is applied, with a small number of early-middle layers already capturing most of the benefit, motivating the selective design.

Gains beyond depth scaling. On GSM8K, LoopMDM improves its same-size non-looping MDM baseline by $+8.5\%$, and also outperforms deeper MDMs whose per-step training FLOPs match those of LoopMDM. This suggests that the gains from looping are not merely a proxy for additional depth, but arise from the repeated application of a shared computation, particularly in math tasks.

Insights into looping and adaptive inference. We analyze how looping interacts with masked positions through a sequence of controlled studies. A Sudoku experiment with a restricted generation order shows that looping enables the model to solve a globally constrained task that a shallow baseline cannot solve. Attention analysis further reveals that mask-to-mask interactions increase with loop counts, and a timestep-level study shows that these gains are concentrated at intermediate noise levels. These observations motivate a simple adaptive strategy that allocates more loop counts where they are most effective, retaining performance while reducing compute.

Together, our findings suggest that masked diffusion language models have significant room to improve their training efficiency and performance through architectural or modeling choices, evidenced by our selective looping approach, and we hope this work sparks further exploration in this direction.

2 Background

We briefly review masked diffusion language models and the training objective used throughout this work. Related work is discussed in Appendix A.

2.1 Masked Diffusion Models

Notation. We consider sequences of length L, denoted $x = (x^1, x^2, \ldots, x^L)$, where each token x^i takes values in a vocabulary $\mathcal{V} = \{1, \ldots, V\}$ and is represented as a one-hot vector. We write $\bar{\mathcal{V}} = \mathcal{V} \cup \{\mathrm{m}\}$ for the vocabulary augmented with a mask token m, and use $\mathrm{Cat}(\cdot \mid p)$ to denote a categorical distribution parameterized by probabilities p over the vocabulary.

Forward process. Let $\alpha_t : [0,1] \to [0,1]$ be a strictly decreasing schedule with $\alpha_0 \simeq 1$ and $\alpha_1 \simeq 0$. We use $\delta_a(b)$ to denote the point mass at a. The forward process independently corrupts each token of a clean sequence $x_0 \sim p_{\text{data}}$ via

$$q(x_t^i \mid x_0^i) = \alpha_t \, \delta_{x_0^i}(x_t^i) + (1 - \alpha_t) \, \delta_{\mathrm{m}}(x_t^i). \tag{1}$$

Reverse process. The reverse process assumes conditional independence across positions. Starting from $x_1 = (\mathrm{m}, \ldots, \mathrm{m})$, a denoising network $p_\theta(x_0^i \mid x_t)$ predicts the clean-token distribution at every masked position. Letting $\alpha_{s|t} = (1 - \alpha_s)/(1 - \alpha_t)$, the reverse transition for $0 \le s < t$ keeps the unmasked tokens intact and updates each masked position as

$$p_\theta(x_s^i \mid x_t) = \alpha_{s|t} \, \delta_m(x_t^i) + (1 - \alpha_{s|t}) \, \mathrm{Cat}(x_s^i \mid p_\theta(x_0^i \mid x_t)), \quad x_t^i = \mathrm{m}. \tag{2}$$

The model is trained by minimizing the negative evidence lower bound (NELBO):

$$\mathcal{L}_{\text{NELBO}} = \mathbb{E}_{t \sim \mathcal{U}[0,1],\, x_t \sim q(x_t \mid x_0)} \left[\sum_{i=1}^{L} \mathbb{I}[x_t^i = \mathrm{m}] \, \frac{\alpha_t'}{1 - \alpha_t} \log \langle p_\theta^i(x_t, t), \, x_0^i \rangle \right], \tag{3}$$

where $\mathbb{I}[\cdot]$ denotes the indicator function and $\langle \cdot, \cdot \rangle$ the standard inner product.

3 Looped Masked Diffusion Models

We introduce *Looped Masked Diffusion Model* (LoopMDM), an architecture that integrates looping into the denoising network of a masked diffusion model. The design is guided by two principles: (i) loop counts should be applied selectively to layers where iterative refinement yields the largest benefit per unit of training FLOPs, rather than uniformly across the network; and (ii) the model should support effective scaling with the number of loop counts, producing monotonic gains as loop counts S increases at inference time. We describe each component below.

Architecture. For exposition, we divide the denoising transformer into three sequential groups of layers, a `head` of n_h layers, a `mid-block` of n_m layers, and a `tail` of n_o layers, where the tail includes an LM head that maps the final hidden states to per-position token distributions. The head and tail are applied once per forward pass, while the mid-block is a *looped* stack whose weights are shared across S successive applications. We keep the head and tail unshared to preserve their specialization for input embedding and output projection.

Given an input $x_t \in \bar{\mathcal{V}}^L$, the head produces an initial hidden state $H_t^{(0)} \in \mathbb{R}^{L \times d}$, the mid-block is applied S times reusing the same parameters, and the tail produces $p_\theta^S(x_0 \mid x_t)$:

$$H_t^{(0)} = \texttt{head}(x_t), \quad H_t^{(k)} = \texttt{mid}\Big(H_t^{(k-1)}\Big), \quad p_\theta^S(x_0 \mid x_t) = \texttt{tail}(H_t^{(S)}). \tag{4}$$

The forward pass unrolls to an effective depth of $n_h + S \cdot n_m + n_o$ layers, while the number of parameters matches a non-looped network with $n_h + n_m + n_o$ layers.

Configuring the looped block. Three factors govern the compute–accuracy trade-off: the number of looped layers n_m, the position of the looped block, and the maximum loop S_{max} during training. We find that effective performance does not require looping the entire network. Instead, selectively looping a small number of layers is sufficient to capture most of the gains, while looping more layers increases training cost without further benefit.

The placement of the loop is equally important. Looping early-middle layers consistently yields the strongest results, suggesting that repeated application is most effective after low-level token representations have formed, but before later layers become specialized for final prediction. Similar observations have been reported in recent studies of looped transformers, where intermediate layers provide a favorable balance between representation stability and iterative computation [3, 63]. A detailed ablation of these factors is presented in Section 4.1.

Training with stochastic loop count. We train LoopMDM by minimizing the standard MDM NELBO applied to the final loop output:

$$\mathcal{L}_{\text{NELBO}} = \mathbb{E}_{t,\, S,\, x_t}\left[\sum_{i=1}^{L} \mathbb{I}[x_t^i = \text{m}]\, \frac{\alpha_t'}{1-\alpha_t} \log \langle p_\theta^{S,i}(x_t, t),\, x_0^i \rangle\right], \tag{5}$$

where $t \sim \mathcal{U}[0,1]$ is sampled uniformly over timesteps, $x_t \sim q(x_t \mid x_0)$, and the loop count $S \sim \mathcal{U}\{1, \ldots, S_{\text{max}}\}$ is sampled uniformly from the discrete set $\{1, \ldots, S_{\text{max}}\}$ at each training step. This stochastic schedule exposes the shared mid-block to a range of effective depths, yielding representations that remain stable across iteration counts and enabling deployment with any S at inference, including values beyond S_{max}.

Training cost. Following standard convention [30], the training FLOPs of a transformer can be approximated as $C = 6ND$, where N is the number of parameters and D is the number of training tokens. Looped architectures differ in that, while the parameter count remains unchanged, looped layers are executed multiple times per forward pass, increasing the per-step compute. For a model with n_m mid layers and maximum loop count S_{max}, the per-step FLOPs of LoopMDM are

$$F_{\text{loop}} = F_{\text{base}} + (\mathbb{E}[S] - 1) \cdot n_m \cdot F_{\text{layer}}, \tag{6}$$

where $\mathbb{E}[S] = (S_{\text{max}} + 1)/2$ under stochastic sampling, F_{base} is the per-step FLOPs of the baseline, and F_{layer} is the FLOPs of a single transformer layer.

To ensure a fair comparison, we match *total training FLOPs* by training LoopMDM for proportionally fewer steps, so that all reported gains arise from architectural differences, not additional compute.

4 Experiments

We organize this section around three findings: (i) selective looping significantly reduces training compute while improving multiple quality metrics (§ 4.1), (ii) the gains are most pronounced on math reasoning tasks, where LoopMDM outperforms deeper MDMs with comparable per-step compute (§ 4.2), and (iii) looping allows for increased interactions among masked positions and exhibits timestep-dependent behavior, which together motivate a simple adaptive inference strategy (§ 4.3).

4.1 Selective looping matches MDM quality with substantially fewer training FLOPs

Our experiments show that the gains of LoopMDM depend critically on selective loop design. By applying looping only to a small set of early-middle layers, LoopMDM achieves performance comparable to or better than the MDM baseline while using only a fraction of the original training compute. Under this selective configuration, LoopMDM matches the baseline test NLL with up to $3.3\times$ fewer training FLOPs (Figure 2), and consistently improves zero-shot perplexity, generative perplexity, and downstream performance across all evaluation settings (Tables 1–3).

Experimental setup. We train two models, LoopMDM and a non-looped MDM baseline, matched in parameter count, training data, and total training FLOPs, differing only in per-step training and inference compute, which are controlled by the loop count S. We train both models on three

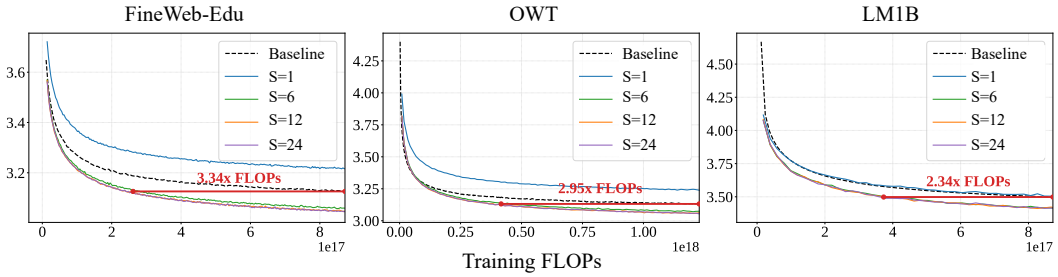


Figure 2: **Test NLL across language pre-training datasets.** Test NLL as a function of training FLOPs on FineWeb-Edu, OpenWebText (OWT), and LM1B. All models are iso-parameter (170M) and trained under matched training FLOPs. Looping is applied to two mid-layers 1-2 in zero-based indexing. Solid curves show LoopMDM with varying inference-time loop counts ($S = 1, 6, 12, 24$), where $S = 24$ exceeds the maximum loop count used during training ($S_{\max} = 12$); the dashed curve shows the MDM baseline. Performance improves consistently with increasing S.

Table 1: **Generative perplexity (↓) across sampling steps.** S denotes the inference-time loop count. Generative perplexity decreases monotonically with S across all denoising step budgets. Results are averaged over five random seeds.

Steps	64	128	256	512	1024
MDM	116.71± 2.72	84.71 ± 2.15	79.43± 2.60	55.50± 2.04	42.56± 1.09
$S = 1$	116.74± 2.66	82.71± 2.35	67.39 ± 1.86	53.13± 0.78	43.86± 1.39
$S = 6$	105.59± 3.83	76.76± 1.09	64.08± 1.39	49.16± 2.65	39.23± 1.31
$S = 12$	**99.06± 5.91**	**68.22± 1.58**	**56.45**± 0.50	**48.16± 3.17**	**38.53± 1.23**

Table 2: **Zero-shot perplexities (↓).** Performance tends to improve with increasing loop counts S. Results are averaged over five random seeds.

Method	**Evaluation benchmark**						
	PTB	WikiText	LM1B	Lambada	AG News	PubMed	Arxiv
MDM	108.5± 6.61	35.1 ± 1.11	68.1 ± 2.37	49.6 ± 1.67	69.3 ± 1.96	43.0 ± 0.42	37.8±1.01
$S = 1$	124.6 ± 12.02	41.8 ± 1.54	77.4 ± 2.69	53.3 ± 2.10	87.8 ± 2.17	47.9 ± 0.71	42.5 ± 1.49
$S = 6$	100.2 ± 9.44	34.2 ± 1.29	68.0 ± 2.72	**44.6** ± 1.91	59.2 ± 1.86	37.2 ± 0.71	33.1 ± 1.01
$S = 12$	**90.2** ± 8.59	**32.4** ± 1.31	**67.8** ± 2.78	45.8 ± 1.90	**56.6** ± 1.70	**36.5** ± 0.70	**32.7** ± 0.97

pretraining corpora, including OpenWebText [21], LM1B [8], FineWeb-Edu [51], using a 170M-parameter diffusion transformer with rotary positional encoding and adaptive layer normalization, following prior work [59, 65]. Sequence length is 128 for LM1B, and 1024 for OpenWebText and FineWeb-Edu. The MDM baseline is trained for 1M steps with a batch size of 512 and sequence packing, while 12-layer LoopMDM loops two (first and second) mid-layers 1-2 in zero-based indexing with $S_{\max} = 12$, without increasing the parameter count. For downstream evaluation, we use FineWeb-Edu checkpoints under matched training compute and follow the protocol of [59, 46], which are detailed in Appendix C.

Test NLLs. Figure 2 reports test negative log-likelihood (NLL) on LM1B, OpenWebText, and FineWeb-Edu. For $S \geq 6$, LoopMDM consistently outperforms the MDM baseline across all three corpora, reaching the same final NLL with $3.34\times$, $2.95\times$, and $2.34\times$ fewer training FLOPs, respectively. The behavior at extreme loop counts S further clarifies this trend. At $S = 1$, LoopMDM underperforms the baseline, consistent with prior observations that shared blocks behave as looping operators whose benefits emerge only through repeated application [15, 19, 81]. At $S = 24$, beyond the training maximum $S_{\max} = 12$, performance continues to improve, indicating that the learned operator generalizes beyond the training regime.

Sampling quality. Table 1 reports *generative perplexity* for unconditionally sampled sequences across varying numbers of denoising steps. LoopMDM consistently improves over the baseline, achieving the lowest perplexity at $S_{\max} = 12$, across all step budgets. The gains are more pronounced

Table 3: **Downstream task accuracy (%) across models.** Performance improves consistently with increasing S, with looped models outperforming the MDM baseline on most benchmarks. Results are averaged over five random seeds.

Method	ARC-c	ARC-e	BoolQ	HellaSwag	OBQA	PIQA	RACE	SIQA	WinoGrande	Avg.
MDM	25.5± 0.7	47.7± 0.9	46.6± 1.1	31.9± 0.3	30.8± 1.3	60.6± 0.2	28.5± 0.8	36.8± 0.4	51.3± 0.7	40.0± 0.3
$S = 1$	23.6± 0.8	45.2± 0.6	53.8± 0.8	31.1± 0.5	**28.2**± 0.6	58.5± 0.6	27.2± 0.7	35.9± 0.6	50.9± 0.5	39.4± 0.3
$S = 6$	**25.7**± 0.2	**47.7**± 1.2	53.8± 1.8	32.5± 0.4	28.0± 0.2	58.9± 0.6	31.3± 0.5	**37.4**± 0.3	51.4± 0.3	40.7± 0.9
$S = 12$	25.3± 0.9	**47.7**± 1.6	**57.4**± 1.3	**32.7**± 1.2	28.0± 0.6	**61.1**± 0.5	**31.7**± 0.4	37.3± 0.8	**51.5**± 0.8	**41.4**± 0.6

Table 4: **Ablation on loop design (test NLL ↓).** We vary loop position, number of looped layers n_m, and maximum loop count $S_{\max}$ under matched training FLOPs. Looping is most effective when applied to mid-layers 1-2 in zero-based indexing, with a small number of looped layers, and moderate loop counts. Performance degrades at both extremes, showing naive or excessive looping is inefficient. Green/red indicates improvements/degradations relative to the non-looped MDM baseline (3.798).

(a) Loop position

Position	NLL	Position	NLL
0	3.744	6	3.735
1	**3.729**	7	3.739
2	**3.729**	8	3.747
3	3.733	9	3.756
4	3.734	10	3.777
5	3.734	–	–

(b) # Looped layers (n_m)

# Layers	NLL	# Layers	NLL
1	3.743	7	3.857
2	**3.729**	8	3.825
3	3.734	9	3.842
4	3.793	10	3.854
5	3.794	11	3.881
6	3.784	12	3.933

(c) Max loop ($S_{\max}$)

$S_{\max}$	NLL
4	3.741
8	3.733
12	**3.729**
16	3.740
20	3.763
24	3.800

at smaller step counts, indicating that looping is particularly effective when the denoising process is limited. Qualitative samples are provided in Appendix D.3.

Zero-shot perplexities. Following prior work [59], we measure *zero-shot perplexity* on seven external corpora unseen during training, including the validation splits of Penn Tree Bank (PTB) [40], WikiText [42], LM1B [8], Lambada [49], AG News [79], and the scientific papers from Scientific Papers (Pubmed and Arxiv; [12]). Table 2 reports zero-shot perplexity on seven external corpora. LoopMDM consistently improves over the baseline across all datasets, achieving gains of 0.3-18.3 points at $S = 12$, with the largest improvement on PTB ($108.5 \rightarrow 90.2$). Already at $S = 6$, it yields substantial improvements, indicating that most gains are realized with moderate loop counts.

Downstream tasks. We evaluate *zero-shot downstream accuracy* with 5 random seeds on nine widely used benchmarks spanning three categories: commonsense reasoning (HellaSwag [77], PIQA [6], Social IQA(SIQA) [62], WinoGrande [61]), reading comprehension (BoolQ [9], RACE [33]), and scientific question answering (OpenBookQA (OBQA) [44], ARC-Easy (ARC-e), ARC-Challenge (ARC-c) [10]).

Table 3 reports zero-shot accuracy on nine benchmarks. LoopMDM at $S = 12$ matches or improves over the baseline on all nine, with the largest gains on BoolQ (+10.8) and RACE (+3.2). Interestingly, despite LoopMDM seeing fewer training tokens under matched total training FLOPs, LoopMDM matches or improves the baseline on downstream tasks. One might have expected fewer training tokens to translate into reduced knowledge for tasks that primarily probe stored facts, yet this is not what we observe. Instead, the per-step compute spent on looping appears to compensate for the reduced token exposure, leaving downstream knowledge transfer intact.

Gains arise from selectively looping early-middle layers. The performance gains above depend critically on how looping is applied. Under a controlled ablation setting on LM1B using a 170M-parameter model trained for 0.1M steps under matched training FLOPs, Table 4 shows that selectively looping a small number of early-middle layers yields the strongest results, while broader or poorly placed looping can underperform the non-looped baseline.

Table 4(a) shows that loop position is highly consequential. Applying looping to early-middle layers (positions 1 or 2 in zero-based indexing) yields the best performance, with both positions achieving the same minimum NLL. In contrast, placing the loop near the input or output layers consistently

degrades performance. This suggests that repeated application of the shared block is most effective at early-middle stages of the network, rather than directly on embedding or final prediction layers.

Table 4(b) further shows that increasing the number of mid layers is inefficient. A small number of looped layers already captures most of the gains, while larger values of n_m consistently degrade performance under the same compute budget. In particular, looping more than six layers underperforms the baseline, indicating that simply applying looping to more layers does not lead to more effective use of training FLOPs.

Finally, Table 4(c) shows that the maximum loop count exhibits a similar trade-off. Small values of $S_{\max}$ limit the benefits of repeated application, while excessively large values degrade performance, with moderate loop counts performing best. Together, these results show that selective application is a critical component of effective training.

4.2 Looping outperforms deeper MDMs at matched per-step training FLOPs in math

Our experiments on GSM8K [11] show that LoopMDM improves over a same-size baseline by up to $+8.5$ points while matching its final accuracy with only 43% of the training FLOPs (Figure 3). It also outperforms deeper MDMs with comparable per-step training FLOPs, indicating that the gains cannot be explained by depth alone.

Experimental setup. LoopMDM is matched to a same-size MDM baseline in parameter count and total training FLOPs, and to the deepest MDM baseline (with 21 layers) in per-step training FLOPs, isolating the contribution of iterative application from raw depth. Following prior work [31], we use a 125M MDM with Qwen2-style attention [72] (hidden size 512, 14 layers, 8 heads, sequence length 512), pretrained on TinyGSM [37], an 11.8M-problem corpus of GSM8K questions paired with structured Python solutions. LoopMDM loops two mid-layers 1–2 in zero-based indexing with $S_{\max} = 8$ and is compared, under matched total training compute, against 14-, 18-, and 21-layer baselines. We evaluate accuracy with Top-2 and Top-3 unmasking under maximum token-wise probability, averaging results over three random seeds. Full details are in Appendix C.

Looping uses depth more effectively than stacking layers. Figure 3 shows that LoopMDM consistently outperforms the 14-layer baseline under matched training compute, with gains increasing monotonically with the loop count S and consistent across Top-2 and Top-3 unmasking. At $S = 16$ (beyond the training maximum $S_{\max} = 8$), the improvement reaches $+8.5$ and $+8.1$ points, respectively. LoopMDM also matches the 14-layer baseline's final accuracy using only 43% of its training FLOPs. More importantly, it outperforms a 21-layer baseline with comparable per-step FLOPs, achieving gains of $+2.7$ at $S = 8$ and $+3.4$ points at $S = 16$. These results show that repeated application of a shared block is more effective than increasing depth for the reasoning task.

4.3 Where do the gains come from?

Sections 4.1 and 4.2 establish that selective looping helps; we now examine *how*. He *et al.*[25] show that MDMs rely on masked positions whose hidden representations participate in prediction. Now we build on this view to study how looping reshapes the computation over these masked positions.

First, we present an illustrative separation showing masked positions can expand the class of solvable tasks when used as a workspace. Second, we show that LoopMDM uses masked positions to solve a task that a shallow model cannot, by a controlled Sudoku experiment. Third, we analyze attention patterns and observe increased interactions among masked positions as the loop count grows. Finally, we show these gains vary across timesteps, which motivates a simple adaptive loop allocation strategy.

Masks as workspace: an illustrative separation. We present a simple example to illustrate the role of masked positions as a workspace. The goal is not to characterize typical behavior, but to provide a clear separation that makes this intuition concrete. We consider the k-Clique problem as a canonical globally coupled task, where all candidate solutions must be evaluated jointly.

Example. *There exists a class of globally coupled prediction tasks solvable by a single denoising step with a masked workspace and a constant number of internal loops, but not by an otherwise-identical predictor without masks under the same loop budget.*

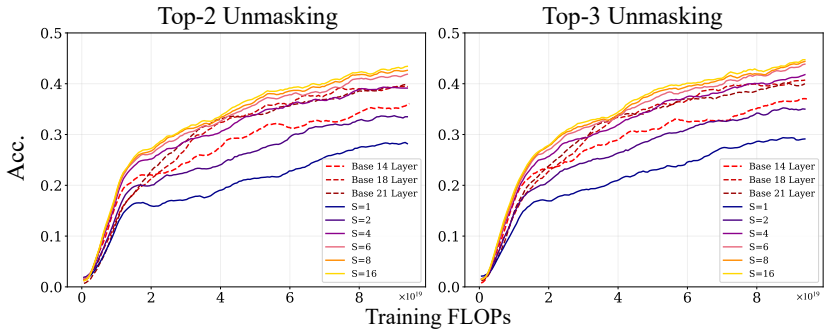


Figure 3: **GSM8K accuracy as a function of training FLOPs.** 14 layers LoopMDM (solid) is compared against MDM baselines with 14, 18, and 21 layers (dashed); the 21-layer baseline is sized so that its per-step training FLOPs approximately match those of LoopMDM. Inference-time loop counts $S \in \{1, 2, 4, 6, 8, 16\}$ are shown, with $S = 16$ exceeding the training maximum. Results are reported under both Top-2 (left) and Top-3 (right) unmasking rules.

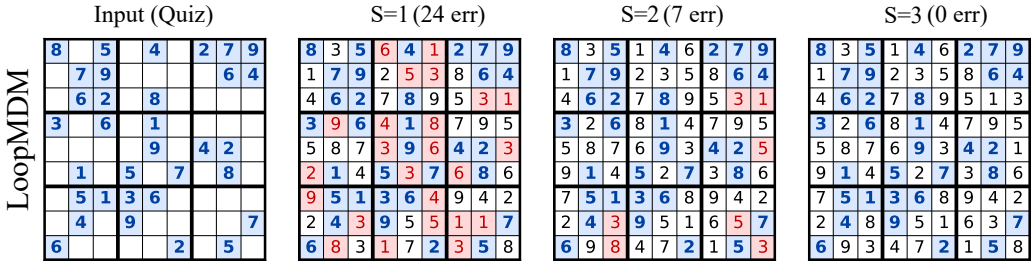


Figure 4: **Looping recovers global consistency under a restricted generation order.** While MDMs can solve Sudoku using adaptive unmasking, we remove this advantage by enforcing a fixed left-to-right (autoregressive-style) order, where early predictions are made with incomplete context. This isolates the role of within-step computation, since improvements can no longer come from selecting easier cells first. We show predictions at loop counts $S = 1, 2, 3$ on the same puzzle (red = errors). As S increases, errors sharply decrease ($24 \to 7 \to 0$), suggesting that looping allows masked positions to revise globally inconsistent early predictions before token commitment.

Proof sketch. A LoopMDM can allocate n^k masked positions, one per candidate k-tuple, and evaluate all candidates in parallel within a single denoising step. A predictor without masked positions lacks such parallel storage under the same compute budget. The full proof is provided in Appendix B.

This example serves to sharpen intuition: masked positions can act as a parallel workspace, enabling forms of computation that are not available without them under the same loop budget.

LoopMDM uses masked positions to solve Sudoku under a restricted generation order. The expressivity result above shows what masks *can* do; we ask whether a trained LoopMDM *does* use them this way in practice. Sudoku provides a natural testbed for this question: solving the puzzle requires assigning a correct value to every position under global consistency constraints, making interactions across unresolved positions essential. MDMs can solve Sudoku near-perfectly when using adaptive unmasking orders that resolve easier cells first [32], but this obscures the role of masked tokens during inference.

To isolate this effect, we enforce a fixed left-to-right generation order, identical to that used in autoregressive models, regardless of cell difficulty. This setting is known to be challenging even for large autoregressive models, including those at the 7B scale [75]. Under this restriction, any improvement must arise from computation within each denoising step rather than from the generation order. We jointly train a 1-layer MDM baseline and its looped counterpart under the same architecture ($d_{\text{model}} = 384$, 12 heads) on a 100k Sudoku dataset, and verify convergence within 100 epochs using $S_{\max} = 6$. We then vary the inference-time loop count S and measure the exact-solve rate on a 1,000-test set.

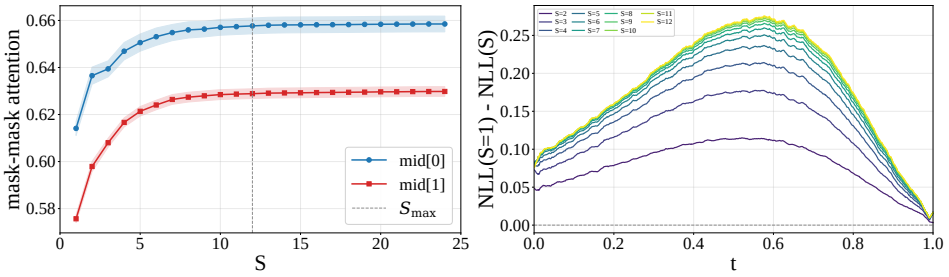


Figure 5: **Analysis of looping behavior.** (Left) Average mask-to-mask attention at timestep $t = 0.5$ for a model with $n_m = 2$, measured at the two mid-block layers (denoted $\mathrm{mid}[0]$ and $\mathrm{mid}[1]$) as a function of loop counts S. Each curve shows attention at the corresponding layer after S loop applications. Attention increases with S and saturates near the training maximum $S_{\max} = 12$ (dashed line). (Right) NLL improvement $\mathrm{NLL}(S{=}1) - \mathrm{NLL}(S)$ across timesteps for sequences of length 1024 on OpenWebText, with $S_{\max} = 12$. As S increases, improvement becomes more concentrated and converges faster, indicating that additional loop counts primarily refine intermediate timesteps.

Figure 4 shows a representative trajectory, where errors decrease from 24 to 7 to 0, as the loop count increases. Early iterations produce locally plausible but globally inconsistent assignments, while later iterations revise these through interactions among masked positions, yielding a consistent solution. Meanwhile, the 1-layer MDM baseline achieves only 10.9% accuracy, and the looped model at $S = 1$ similarly reaches 10.6%, indicating that shallow left-to-right generation alone is insufficient for solving the task.

Mask-to-mask attention increases with loop count in language modeling. The Sudoku result shows that LoopMDM can effectively utilize masked positions; we next examine whether similar patterns arise in language modeling. We analyze how interactions among masked positions evolve with the loop count. On a LoopMDM trained on OpenWebText, we measure *mask-to-mask attention* at two mid-block layers, defined as the average attention mass from masked-position queries to masked-position keys, while sweeping the inference loop count S up to and beyond $S_{\max}$.

Figure 5 (left) shows that mask-to-mask attention increases with S at both layers and plateaus near the training maximum $S_{\max} = 12$. The trend is consistent across layers and remains stable when S exceeds $S_{\max}$. While not causal, this pattern suggests that later loop counts place greater emphasis on interactions among masked positions, consistent with the behavior observed in the Sudoku experiment.

Looping helps unevenly across denoising timesteps. We next ask where along the denoising process looping is most useful. For each timestep t, we measure the per-position NLL gain from additional loop iterations, $\mathrm{NLL}_t(S = 1) - \mathrm{NLL}_t(S)$. Figure 5 (right) shows that the gains are not uniform: they peak at intermediate timesteps, where the model has both sufficient context and unresolved positions, and are smaller near the two ends of the denoising process. Increasing S amplifies this profile but gradually saturates, suggesting that additional loops are useful unevenly across timesteps.

This observation suggests a simple way to choose loop budgets at inference time. Rather than fixing the same loop count at every timestep, we monitor the relative hidden-state change between successive loop iterations, $\|H^{(k)} - H^{(k-1)}\|/\|H^{(k)}\|$, and stop once this ratio falls below a threshold ϵ or the maximum loop budget is reached. In all experiments, we use $\epsilon = 0.1$. This rule allocates fewer iterations when hidden states stabilize early and more when updates remain large. As summarized in Table 5, adaptive looping reduces the average loop count from $S = 12$ to roughly five iterations while preserving most of the performance gains across all evaluation settings.

Table 5: **Adaptive looping trade-off.** Adaptive looping preserves performance while substantially reducing the average loop count.

Setting	$S \to \bar{S}$	Metric
Zero-shot PPL (↓)	$12 \to 5.1$	$51.7 \to 52.9$
Generative PPL (↓)	$12 \to 5.2$	$38.53 \to 39.11$
Downstream Acc. (↑)	$12 \to 4.9$	$41.4 \to 41.3$

Appendix D.1 shows that this rule yields a smooth computation-performance trade-off and naturally allocates more loops to intermediate timesteps. These results provide a simple practical guide for selecting loop counts in LoopMDM.

5 Conclusion

We introduced LoopMDM, which selectively loops a small shared block inside masked diffusion language models. LoopMDM improves matched-compute performance across language modeling and reasoning tasks without increasing parameter count, and achieves comparable or better performance with fewer training FLOPs than non-looped baselines. These results suggest that the gains from looping are not solely explained by increased depth, but are consistent with benefits from reusing computation within the diffusion architecture. More broadly, our findings indicate that selective looping can serve as a simple approach for improving masked diffusion models under fixed parameter and compute budgets. We believe our work opens up broad opportunities for architectural advances in improving training efficiency and generalization. We provide limitations in Appendix E.

Acknowledgments and Disclosure of Funding

This research was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2019-II190075, RS-2024-00509279, RS-2025-II212068, RS2023-00227592, RS-2025-02214479, RS-2024-00457882, RS-2025-25441838, RS2025-02217259, RS-2026-25519202, RS-2024-00436680) and the Culture, Sports, and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism (RS-2024-00345025, RS-2024-00333068), the InnoCORE program of the Ministry of Science and ICT (N10250156) and National Research Foundation of Korea (RS-2024-00346597).

References

[1] Cem Anil, Ashwini Pokle, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, Shaojie Bai, J Zico Kolter, and Roger B Grosse. Path independent equilibrium models can better exploit test-time computation. *Advances in Neural Information Processing Systems*, 35:7796–7809, 2022.

[2] 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, 2021.

[3] Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025.

[4] 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.

[5] Ahsan Bilal, Muhammad Ahmed Mohsin, Muhammad Umer, Asad Aali, Muhammad Usman Khanzada, Muhammad Usman Rafique, Zihao He, Emily Fox, and Dean F Hougen. s^3: Stratified scaling search for test-time in diffusion language models. *arXiv preprint arXiv:2604.06260*, 2026.

[6] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[7] Chen-Hao Chao, Wei-Fang Sun, Hanwen Liang, Chun-Yi Lee, and Rahul G Krishnan. Beyond masked and unmasked: Discrete diffusion models via partial masking. *arXiv preprint arXiv:2505.18495*, 2025.

[8] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 615–621, 2014.

[9] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pages 2924–2936, 2019.

[10] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[11] 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.

[12] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, 2018.

[13] Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6292–6299, 2019.

[14] Meihua Dang, Jiaqi Han, Minkai Xu, Kai Xu, Akash Srivastava, and Stefano Ermon. Inference-time scaling of diffusion language models with particle gibbs sampling. *arXiv preprint arXiv:2507.08390*, 2025.

[15] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[16] Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

[17] Justin Deschenaux, Lan Tran, and Caglar Gulcehre. Partition generative modeling: Masked modeling without masks. *arXiv preprint arXiv:2505.18883*, 2025.

[18] Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. *arXiv preprint arXiv:2409.15647*, 2024.

[19] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

[20] Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pages 11398–11442. PMLR, 2023.

[21] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[22] Shansan Gong, Ruixiang ZHANG, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. In *The Fourteenth International Conference on Learning Representations*, 2026.

[23] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024.

[24] 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.

[25] Andre He, Sean Welleck, and Daniel Fried. Reasoning with latent tokens in diffusion language models. *arXiv preprint arXiv:2602.03769*, 2026.

[26] Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuan-Jing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. In *Proceedings of the 61st annual meeting of the association for computational linguistics (ACL)*, volume 1, pages 4521–4534, 2023.

[27] Chunsan Hong, Sanghyun Lee, and Jong Chul Ye. Unifying masked diffusion models with various generation orders and beyond. *arXiv preprint arXiv:2602.02112*, 2026.

[28] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. volume 34, pages 12454–12465, 2021.

[29] Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

[30] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[31] Jaeyeon Kim, Jonathan Geuter, David Alvarez-Melis, Sham Kakade, and Sitan Chen. Stop training for the worst: Progressive unmasking accelerates masked diffusion training. *arXiv preprint arXiv:2602.10314*, 2026.

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

[33] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 785–794, 2017.

[34] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[35] Sanghyun Lee, Seungryong Kim, Jongho Park, and Dongmin Park. Lookahead unmasking elicits accurate decoding in diffusion language models. *arXiv preprint arXiv:2511.05563*, 2025.

[36] Sanghyun Lee, Sunwoo Kim, Seungryong Kim, Jongho Park, and Dongmin Park. Effective test-time scaling of discrete diffusion through iterative refinement. *arXiv preprint arXiv:2511.05562*, 2025.

[37] Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving> 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*, 2023.

[38] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. 2024.

[39] Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed: Dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025.

[40] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[41] Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025.

[42] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[43] William Merrill and Ashish Sabharwal. Exact expressive power of transformers with padding. *arXiv preprint arXiv:2505.18948*, 2025.

[44] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2381–2391, 2018.

[45] Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. Cotformer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. *arXiv preprint arXiv:2310.10845*, 2023.

[46] Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

[47] 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. 2025.

[48] 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. 2025.

[49] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534, 2016.

[50] William Peebles and Saining Xie. Scalable diffusion models with transformers. pages 4195–4205, 2023.

[51] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

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

[53] Fred Zhangzhi Peng, Zachary Bezemek, Jarrid Rector-Brooks, Shuibai Zhang, Anru R Zhang, Michael Bronstein, Alexander Tong, and Avishek Joey Bose. Planner aware path learning in diffusion language models training. *arXiv preprint arXiv:2509.23405*, 2025.

[54] Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *First Conference on Language Modeling*, 2024.

[55] Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

[56] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[57] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[58] Benjamin Rossman. On the constant-depth complexity of k-clique. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 721–730, 2008.

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

[60] Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. The diffusion duality. 2025.

[61] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[62] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 4463–4473, 2019.

[63] Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

[64] Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026.

[65] Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. volume 37, pages 103131–103167, 2024.

[66] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

[67] Anej Svete and Ashish Sabharwal. On the reasoning abilities of masked diffusion language models. *arXiv preprint arXiv:2510.13117*, 2025.

[68] Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pages 78–90, 2023.

[69] Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. *arXiv preprint arXiv:2503.00307*, 2025.

[70] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[71] Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream-coder 7b: An open diffusion language model for code, 2025.

[72] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report, 2024a. *URL https://arxiv. org/abs/2407.10671*, 6, 2023.

[73] Kaisen Yang, Jayden Teoh, Kaicheng Yang, Yitong Zhang, and Alex Lamb. Improving sampling for masked diffusion models via information gain. *arXiv preprint arXiv:2602.18176*, 2026.

[74] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.

[75] Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint arXiv:2410.14157*, 2024.

[76] 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.

[77] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

[78] Kevin Zhai, Sabbir Mollah, Zhenyi Wang, and Mubarak Shah. Core: Context-robust remasking for diffusion language models. *arXiv preprint arXiv:2602.04096*, 2026.

[79] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[80] 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*, 2025.

[81] Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Related Work

Masked diffusion models. Discrete diffusion models [2, 28] model discrete data distributions via a diffusion process, and have recently emerged as a promising alternative to autoregressive language models. Two main types of forward corruption processes are commonly used: uniform corruption, which replaces tokens with uniformly random tokens [38, 60], and masking corruption, which replaces tokens with a special mask token [59, 65]. Among these, masked diffusion models (MDMs) have become the dominant framework for discrete diffusion models due to their simplicity and strong empirical performance.

The standard masking-based formulation [48, 59] remains the most widely adopted backbone. Large-scale models such as LLaDA family [47, 80, 4], Seed Diffusion [66], and Dream [76] retain the basic generation scheme, where a fully masked sequence is iteratively denoised into tokens. Empirically, MDMs have become increasingly competitive with autoregressive models, with recent models matching or outperforming comparable AR baselines across multiple benchmarks [47, 76, 22, 71].

Looped transformers. Looping has long been explored as a way to increase the effective depth of transformers without increasing the number of parameters. Early examples include the Universal Transformer [15], which repeatedly applies a shared transition block and shows strong performance on algorithmic tasks, and ALBERT [34], which demonstrates the practical benefits of cross-layer parameter sharing in BERT-style pretraining. Subsequent work has studied looped transformers as programmable or iterative computation models [20, 74], and more recent large-scale models such as Ouro [81] and Huginn [19] suggest that recurrent depth can scale to modern language modeling regimes.

Broadly, existing work on looped transformers can be viewed along two related axes. The first emphasizes iterative refinement: repeated applications of the same block allow the model to progressively improve an internal representation. This perspective has been used to improve length generalization on algorithmic tasks [18], to implement token-level variants of chain-of-thought-style computation [45], and to study the theoretical power of repeated transformer computation [63]. The second emphasizes parameter efficiency: weight sharing allows a model to obtain greater effective depth under a fixed parameter budget. This idea predates modern LLMs and has been explored in recurrent or shared-parameter sequence models [13, 68], while recent work shows that looped language models can achieve strong reasoning ability under iso-FLOP or parameter-efficient regimes [63, 81, 55].

Our work combines these two perspectives in the setting of masked diffusion language models. Rather than using looping only to reduce parameters or to add generic latent depth, we study how looped computation interacts with the distinct structure of MDMs: the presence of many masked positions. These masked positions provide a latent workspace, while the looped block iteratively refines the hidden states on that workspace before token prediction. This lets us obtain both training efficiency through selective weight sharing and iterative refinement through computation over masked positions, which is not naturally present in standard autoregressive looping, where unresolved output positions are not maintained as an explicit masked workspace.

Test-Time Scaling in Masked Diffusion. Scaling inference-time computation has emerged as a complementary axis to model scaling. In autoregressive models, chain-of-thought prompting [70] increases computation by generating longer reasoning traces. Subsequent work explores more implicit forms of test-time compute, including pause or filler tokens [23, 54] and latent-space reasoning that does not rely on explicit natural language tokens [16, 24]. These results suggest that additional computation can be effectively allocated without being directly exposed in the output sequence.

In masked diffusion, most existing test-time scaling methods focus on the sampling procedure itself. Key design choices include how many tokens to unmask at each step, which positions to select, and whether to revise previously generated tokens. While confidence-based heuristics are commonly used, recent work shows that the unmasking schedule critically affects generation quality. This has led to adaptive timestep or parallel decoding strategies [29], dilated unmasking schedules that avoid resolving strongly coupled tokens simultaneously [39], remasking-based samplers that enable correction during inference [69], and search or planned unmasking policies [52, 53, 35, 36, 5].

B Theoretical Analysis

B.1 Additional Background: Padded Looped Transformers and Masked Diffusion

We briefly review the theoretical perspective that motivates our interpretation of masked positions as computational workspace.

Padding and looping as computational resources. Recent work on padded looped transformers studies blank or padding tokens as a form of parallel test-time computation. These padding tokens do not carry additional input content; rather, they can store intermediate information. In this sense, padding behaves like a workspace, or equivalently like an increase in computational width. Looping, on the other hand, repeatedly applies a shared transformer block and therefore increases effective computational depth without increasing the number of parameters. This distinction is explicit in the theory of padded looped transformers: padding controls the amount of parallel storage available to the model, while looping controls the number of iterative refinement steps [43].

Connection to masked diffusion models. This viewpoint has recently been connected to masked diffusion language models (MDMs). In the finite-precision logarithmic-width setting, MDMs and padded looped transformers (PLTs) are closely related computational models [67]. At a high level, both models perform iterative parallel refinement. A PLT refines continuous residual states over an input augmented with padding positions, while an MDM refines a partially masked or partially generated sequence through denoising steps. Under this correspondence, the generated or masked positions of an MDM play a role analogous to padding positions in a PLT: they provide additional locations where intermediate computation can be represented before the final prediction is read out.

More concretely, prior work shows that MDMs can be simulated by PLTs with comparable step and output budgets, and that PLTs can be simulated by MDMs with additional output space used to store the PLT residual stream [67]. Although the two formalisms differ in how information is represented, with PLTs using continuous residual states and MDMs operating over discrete masked or generated symbols across denoising steps, they share the same core computational structure. In both cases, additional positions provide storage for intermediate information, and repeated application of a model block refines that stored state.

Relation to our setting. Our model differs slightly from the standard PLT and MDM correspondence. We place looping inside a single MDM denoising network rather than increasing the number of external denoising steps. Nevertheless, the same conceptual decomposition is useful. The masked positions are present throughout the denoising network's forward computation, and their hidden states can be updated repeatedly by the looped mid-block before the model commits to discrete token predictions. We therefore view the masked positions as a latent workspace, and the internal loops as iterative computation over this workspace.

We use this perspective only as an explanatory tool. The goal of the separation below is not to give a tight characterization of practical LoopMDMs, nor to claim that all assumptions of the formal PLT and MDM models hold exactly in our architecture. Rather, the goal is to isolate a simple mechanism: masked positions can supply parallel workspace that a padding-free model lacks.

B.2 The k-Clique Workspace Task

Let G be an n-vertex graph represented by its adjacency matrix

$$A \in \{0,1\}^{n \times n}.$$

The input length is $N = \Theta(n^2)$. For a fixed constant k, define

$$\mathrm{CLIQUE}_{k,n}(A) = 1$$

if and only if there exist distinct vertices

$$v_1, \ldots, v_k \in [n]$$

such that

$$A_{v_r, v_s} = 1 \qquad \text{for all } 1 \le r < s \le k.$$

In the padded version of the task, we append n^k masked workspace positions and one masked answer position. The n^k workspace positions are indexed by ordered tuples

$$\mathbf{v} = (v_1, \ldots, v_k) \in [n]^k.$$

The intended role of the workspace position indexed by $\mathbf{v}$ is to store whether this tuple forms a k-clique.

B.3 Main Example

Example 1. *Fix a padding-free finite-precision logarithmic-width transformer architecture with a constant number of internal loop counts. Suppose that, on inputs of length N, this architecture is simulable by constant-depth AC^0 circuits of size $O(N^C)$ for some constant C.*

Then there exists a constant $k > 8C$ such that the corresponding padding-free architecture with no additional positions does not decide $\mathrm{CLIQUE}_{k,n}$ on n-vertex graphs.

On the other hand, there exists a one-step masked-workspace transformer architecture with $n^k + O(1)$ masked positions, the same asymptotic finite-precision and logarithmic-width setting, a constant loop budget, and constant local resources depending only on k, that decides $\mathrm{CLIQUE}_{k,n}$.

Proof of Example. The key distinction is that masked workspace provides an explicit set of n^k hypothesis slots, one for each candidate k-tuple of vertices. This allows the model to evaluate all candidates in parallel.

In contrast, a base transformer must encode all intermediate combinatorial structure within a logarithmic-width representation. Under finite precision and a constant loop budget, this restricts the model to computations simulable by constant-depth AC^0 circuits, which are known to be insufficient for solving k-Clique for large enough k.

Upper bound: masked workspace solves the task. We construct a padded looped transformer that decides $\mathrm{CLIQUE}_{k,n}$.

The graph is represented by edge tokens $e_{a,b}$, one for each ordered pair $(a, b) \in [n]^2$. The token $e_{a,b}$ stores the edge bit $A_{a,b}$, and its positional encoding stores the pair (a, b).

We append n^k masked workspace positions and one masked answer position. Each workspace position is indexed by a tuple

$$\mathbf{v} = (v_1, \ldots, v_k) \in [n]^k.$$

For example, the t-th workspace position can be assigned the tuple whose coordinates are

$$v_r(t) = 1 + \left\lfloor \frac{t-1}{n^{r-1}} \right\rfloor \bmod n, \qquad r = 1, \ldots, k.$$

Since k is constant, this tuple information requires only $O(k \log n)$ bits, which is compatible with logarithmic width.

For every pair $1 \le r < s \le k$, the workspace token $z_{\mathbf{v}}$ uses one attention head to retrieve the edge token e_{v_r,v_s}. This can be implemented by position-matching attention: the query at $z_{\mathbf{v}}$ encodes the pair (v_r, v_s), while the key at edge token $e_{a,b}$ encodes (a, b). Hence the head retrieves the value A_{v_r,v_s}.

After retrieving all edge bits

$$A_{v_r,v_s}, \qquad 1 \le r < s \le k,$$

a position-wise feedforward network computes

$$c_{\mathbf{v}} = \left(\bigwedge_{1 \le r < s \le k} A_{v_r,v_s} \right) \wedge \left(\bigwedge_{1 \le r < s \le k} [v_r \neq v_s] \right).$$

Because k is constant, this is a constant-size Boolean computation. Thus $c_{\mathbf{v}} = 1$ if and only if the tuple $\mathbf{v}$ forms a valid k-clique.

The answer position then attends to all workspace positions and computes

$$\bigvee_{\mathbf{v}\in[n]^k} c_{\mathbf{v}}.$$

This value is 1 exactly when the graph contains a k-clique.

Lower bound: base constant-loop models cannot solve the task. We prove the lower bound for the constant $k > 8C$ chosen in Example 1. Suppose, for contradiction, that the corresponding base looped transformer decides $\mathrm{CLIQUE}_{k,n}$.

The padding-free model receives only the graph tokens. Since the graph is represented by an $n \times n$ adjacency matrix, its input length is

$$N = \Theta(n^2).$$

Fixed padding-free architecture is simulable, on inputs of length N, by a constant-depth AC^0 circuit of size

$$O(N^C).$$

The constant loop budget is already accounted for in this hypothesis: unrolling a constant number of internal loop counts only changes the effective depth by a constant depending on the fixed architecture.

Therefore, if the padding-free transformer decided $\mathrm{CLIQUE}_{k,n}$, then there would exist a constant-depth AC^0 circuit family deciding $\mathrm{CLIQUE}_{k,n}$ with size

$$O(N^C) = O\big((n^2)^C\big) = O(n^{2C}).$$

We now relate this ordered-adjacency representation to the standard undirected k-Clique problem. Rossman's lower bound [58] is stated for simple undirected graphs. If a circuit decides $\mathrm{CLIQUE}_{k,n}$ on ordered adjacency matrices $A \in \{0,1\}^{n\times n}$, then it also yields a circuit for the standard undirected representation with variables $\{x_{\{u,v\}} : 1 \le u < v \le n\}$ by substituting

$$A_{u,v} = A_{v,u} = x_{\{u,v\}} \qquad \text{for } u < v,$$

and

$$A_{u,u} = 0 \qquad \text{for all } u \in [n].$$

This substitution only hard-wires and duplicates input variables; it does not increase the circuit depth or asymptotic size. Hence the assumed padding-free transformer would imply a constant-depth AC^0 circuit family of size $O(n^{2C})$ for the standard k-Clique problem on n-vertex graphs.

Rossman's lower bound states [58] that, for every fixed constant k, the k-Clique problem on n-vertex graphs requires constant-depth circuits of size

$$\omega(n^{k/4}).$$

Choose $k > 8C$. Then

$$\frac{k}{4} > 2C,$$

and therefore

$$O(n^{2C}) = o(n^{k/4}).$$

This contradicts Rossman's lower bound. Thus the padding-free constant-loop predictor cannot decide $\mathrm{CLIQUE}_{k,n}$. □

C Experimental Details

C.1 Experimental details for language modeling

Datasets and preprocessing. We evaluate on One Billion Word (LM1B) [8], OpenWebText (OWT) [21], and FineWeb-Edu [51]. For LM1B, we follow prior work [38, 59] and detokenize the dataset, then tokenize using the BERT-BASE-UNCASED tokenizer [26]. Sequences are concatenated and packed to a fixed length of 128 using sentence packing [57]. For OWT and FineWeb-Edu, we use the GPT-2 tokenizer [56] and pack sequences to a context length of 1,024, inserting an `eos` token between documents. Following prior work, we reserve the last 100K documents of OWT for validation; FineWeb-Edu uses its standard split.

Model and training. We use a 170M-parameter diffusion transformer (DiT) [50] with rotary positional encoding and adaptive layer normalization, consistent across all datasets. The model has 12 layers with hidden size 768 and 12 attention heads. For LoopMDM, we adopt an iso-parameter setting by looping two layers 1–2 in zero-based indexing (starting from the second layer) with a maximum loop count $S_{\max} = 12$. All experiments were conducted using 4 NVIDIA A100 GPUs.

The baseline is trained for 1M steps; LoopMDM is trained for proportionally fewer steps to match total training FLOPs with batch size 512 using the AdamW optimizer. We use a linear warmup schedule to $3\mathrm{e}{-4}$ over 2,500 steps, following prior works [59, 38, 60]. All looped models are trained under matched training FLOPs to ensure fair comparison with the baseline.

C.2 Experimental details of math reasoning

Dataset. We use TinyGSM [37] as the pretraining corpus, which consists of 11.8M GSM8K-style problems paired with structured Python solutions. Following prior works [37, 31], this formulation improves learnability at small model scales and enables controlled comparison of reasoning performance.

Model and training. We use a 125M-parameter masked diffusion model with Qwen2-style attention, consisting of 14 layers, hidden size 512, 8 attention heads, and maximum sequence length 512. All models share the same architecture, optimizer, and training setup. All experiments were conducted using NVIDIA A100 GPUs.

For LoopMDM, we adopt an iso-parameter setting by looping two layers 1–2 in zero-based indexing, while all other components remain unchanged. We train for 900k iterations using AdamW with learning rate $3\mathrm{e}{-4}$, batch size 32 per GPU on 4 GPUs, and apply exponential moving average (EMA) with decay 0.9999.

Evaluation. Evaluation is performed on the GSM8K test set every 5k steps using Top-2 and Top-3 unmasking with maximum token-wise probability as confidence. All experiments were conducted using NVIDIA A100 GPUs. All results are reported using EMA checkpoints.

C.3 Experimental details of downstream task evaluation

We evaluate downstream performance using the models pretrained on FineWeb-Edu for 1M steps under the same setup as described in the language modeling experiments. No additional fine-tuning is performed.

Benchmarks. We evaluate on nine widely used benchmarks spanning three categories. For *commonsense reasoning*, we use HellaSwag [77], PIQA [6], SIQA [62], and WinoGrande [61]. For *reading comprehension*, we use BoolQ [9] and RACE [33]. For *scientific question answering*, we use OpenBookQA [44], ARC-Easy, and ARC-Challenge [10].

Evaluation protocol. We adopt a likelihood-based evaluation. For each multiple-choice question, we estimate the likelihood of each candidate answer by conditioning on the input context and selecting the option with the highest likelihood. Specifically, we compute likelihoods over up to 128 tokens per candidate. Since the model is trained with a maximum sequence length of 1024, all inputs are padded to this length during evaluation.

C.4 Experimental details of analysis

Mask count analysis. To analyze the effect of looping across different masking regimes, we partition sequences by the number of masked positions. Specifically, we vary the number of masked tokens from 1 to 1024 and group sequences accordingly. For each mask count, we measure the improvement in NLL on the OpenWebText validation set, comparing different inference-time loop counts.

Mask-to-mask attention analysis. To study how looping affects interactions between masked positions, we analyze attention patterns across loop counts. We sample 10,000 timesteps uniformly

Table 6: **Effect of the adaptive stopping threshold ϵ.** Smaller ϵ increases the average number of loop counts and improves performance, while larger ϵ reduces compute at the cost of degraded perplexity. We use $\epsilon = 0.1$ in the main paper as a balanced operating point, retaining most of the gains of fixed $S = 12$ while reducing average compute by roughly half.

Zero-shot PPL Ablation			
Mode	Avg PPL ↓	$\bar{S}$ ↓	Compute
$S = 6$	56.80	6.00	50.0%
$S = 12$	55.58	12.00	100.0%
$\epsilon = 0.02$	55.57	11.73	97.7%
$\epsilon = 0.04$	55.68	9.60	80.0%
$\epsilon = 0.06$	55.94	8.02	66.9%
$\epsilon = 0.08$	56.28	6.96	58.0%
$\epsilon = 0.10$	56.68	6.20	51.6%
$\epsilon = 0.12$	57.16	5.49	45.7%
$\epsilon = 0.14$	57.64	5.02	41.8%
$\epsilon = 0.16$	58.21	4.47	37.3%
$\epsilon = 0.18$	58.81	4.12	34.3%
$\epsilon = 0.20$	59.45	3.74	31.2%

Generative PPL Ablation			
Mode	Gen PPL ↓	$\bar{S}$ ↓	Compute
$S = 6$	63.60	6.00	50.0%
$S = 12$	62.44	12.00	100.0%
$\epsilon = 0.02$	62.66	10.73	89.5%
$\epsilon = 0.04$	64.17	8.33	69.4%
$\epsilon = 0.06$	63.16	7.10	59.2%
$\epsilon = 0.08$	64.11	6.21	51.7%
$\epsilon = 0.10$	64.39	5.45	45.4%
$\epsilon = 0.12$	65.04	4.86	40.5%
$\epsilon = 0.14$	64.68	4.52	37.7%
$\epsilon = 0.16$	64.93	3.94	32.9%
$\epsilon = 0.18$	65.51	3.74	31.2%
$\epsilon = 0.20$	66.03	3.41	28.4%

from $t \in [0, 1]$ and compute the average attention weights at each loop counts. In particular, we measure mask-to-mask attention by averaging the attention mass assigned from masked-query tokens to masked-key tokens.

D Omitted Results

D.1 Ablation on Adaptive Looping

Trade-off controlled by the threshold ϵ. Table 6 reports the effect of the adaptive stopping threshold ϵ on average zero-shot perplexity and generative perplexity. Smaller ϵ increases the average number of loop counts and improves performance, while larger ϵ reduces computation with gradually degraded perplexity. Across a broad range of thresholds, the adaptive strategy remains competitive with fixed high loop counts while using substantially fewer iterations on average.

In the main paper, we use $\epsilon = 0.1$, which provides a balanced trade-off between performance and computation. At this operating point, adaptive looping reduces the average number of iterations to roughly half of that for a fixed $S = 12$ while maintaining comparable zero-shot and generative perplexity.

Adaptive allocation across timesteps. Figure 6 shows the average number of loop counts allocated by the adaptive strategy as a function of timestep. We evaluate the allocation behavior on OpenWebText using $\epsilon = 0.1$, averaging over 100 sampled sequences for both zero-shot perplexity and 1024-step generative perplexity settings.

Across all evaluations, the adaptive strategy consistently allocates more iterations to intermediate timesteps and fewer to early and late timesteps. This matches the analysis in Section 4.3, where looping is most effective when both context and unresolved positions are sufficiently available. The average loop count remains substantially below the training maximum $S_{\max} = 12$, while still concentrating computation on the timesteps where looping provides the largest gains.

Together, these results suggest that adaptive looping provides an efficient and interpretable compute allocation strategy, automatically redistributing computation toward the most useful stages of the denoising process.

D.2 Compare With Other Looping Methods

Compare with input injection. In ARMs, particularly recurrent architectures inspired by equilibrium models [1], input injection is commonly used to stabilize iterative computation [19]. In MDMs,

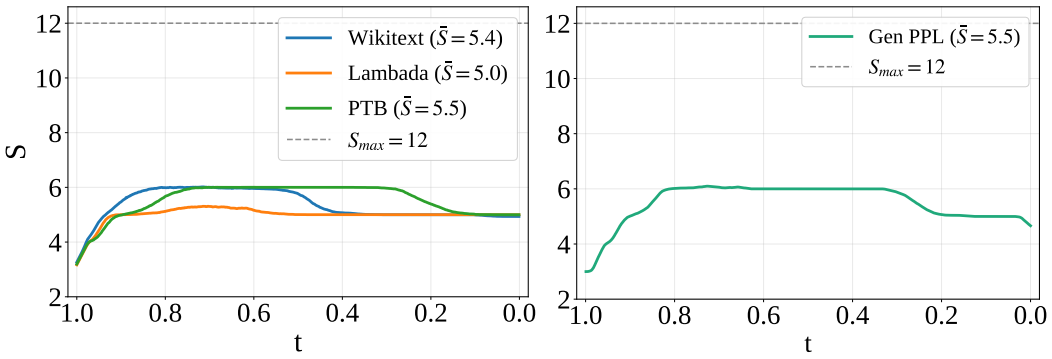


Figure 6: **Adaptive allocation of loop counts across timesteps.** Average loop count allocated by the adaptive strategy as a function of timestep t on OpenWebText with $\epsilon = 0.1$, measured over 100 sampled sequences. (Left) Zero-shot perplexity evaluation on WikiText, Lambada, and PTB. (Right) Generative perplexity evaluation. Across both settings, the adaptive strategy allocates more iterations at intermediate timesteps and fewer at early and late timesteps. The dashed line indicates the maximum training loop count $S_{\max} = 12$.

predicted positions already begin from a shared absorbing state through the mask token. A natural analogue in this setting is therefore to inject Gaussian noise into the mask embedding during training.

As shown in Figure 7, input injection degrades performance relative to the standard MDM baseline. This suggests that techniques developed for recurrent AR models do not transfer directly to the masked diffusion setting.

Compare with log-normal Poisson loop sampling. Recent looped transformers sample loop counts from a log-normal Poisson distribution to stabilize recurrent computation and improve test-time scaling [19, 41]. We compare this strategy against the uniform loop-count sampling used in LoopMDM.

Figure 7 shows that log-normal Poisson sampling yields only marginal improvement over the baseline and consistently underperforms the uniform sampling strategy used in LoopMDM. These results suggest that the masked diffusion setting benefits more from controlled exposure to different effective depths than from additional stochasticity in loop allocation.

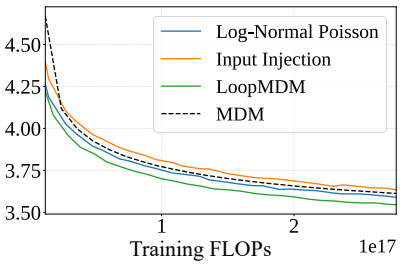


Figure 7: **NLL comparison with alternative looping strategies.**

D.3 Qualitative examples

To better understand how looping changes token predictions across iterations, we analyze per-token trajectories as the inference-time loop count increases. Figure 8 tracks both the predicted token and its NLL at a focal position for several representative examples from OpenWebText.

Across successful cases (A–D), predictions progressively evolve toward the ground-truth token as the number of loop counts increases. Different behaviors emerge across examples. In some cases, looping reinforces local copying patterns (A), while in others it resolves semantic ambiguity (B), refines syntactic category selection (C), or integrates broader contextual information (D). In all cases, lower NLL correlates with increased confidence in the final prediction.

At the same time, looping is not uniformly beneficial (E) shows a counterexample where additional iterations eventually move the prediction away from an initially correct token. This suggests that repeated application of the shared block can occasionally oversmooth or destabilize local predictions at large loop counts.

(A) Rescued — local copying. The focal slot lies inside *"half ___ school ___ **[focal]** book club"*; the loop ends up copying the repeated word.

GT: `half`

S	1	2	3	4	**6**	12	24
pred	a	a	a	a	**half** ✓	half ✓	half ✓
NLL	7.00	4.44	2.62	1.25	**0.38**	0.13	0.13

(B) Rescued — semantic selection. *"... at the numbers ___ **[focal]** delegates are ___ proportionately, it's ___ how much ___ change..."* The loop replaces `which` with the causal conjunction.

GT: `Because`

S	1	2	**3**	4	12	24
pred	which	which	**Because** ✓	Because ✓	Because ✓	Because ✓
NLL	6.44	4.00	**1.75**	1.12	0.50	0.38

(C) Rescued — syntactic refinement (POS). *"... Geers said. ___ **[focal]** present moment..."* Early iterations emit prepositions; later it locks onto the verb.

GT: `described`

S	1	2	3	4	6	**8**	24
pred	at	at	in	in	in	**described** ✓	described ✓
NLL	9.38	8.06	6.12	4.00	2.62	**1.88**	1.38

(D) Rescued — multi-hop / global context. *"... I had ___ Clinton last ___ Michigan and saw **[focal]** ___ a chance to ___ off ___-from-behind..."* The model resolves to a topic-specific weekday.

GT: `Tuesday`

S	1	2	4	6	8	**12**	24
pred	her	the	the	the	the	**Tuesday** ✓	Tuesday ✓
NLL	7.56	5.12	2.88	2.00	1.12	**0.62**	0.38

(E) Lost — counter-example. *"... voters were split on candidates' key ___ **[focal]** Clinton ___ as ___ electable..."* Further looping drifts to `whether`.

GT: `with`

S	**1**	2	3	4	6	8	24
pred	**with** ✓	with ✓	with ✓	with ✓	who	whether	whether
NLL	**0.75**	0.50	0.63	0.75	2.38	5.31	6.69

Figure 8: **Per-token loop trajectories across iterations.** Each panel tracks the predicted token (pred) and its NLL at the focal position as the number of loop counts S increases. Lower NLL indicates higher model confidence. (A–D) show successful refinement where predictions evolve toward the ground-truth token (GT) via different mechanisms: (A) local copying, (B) semantic selection, (C) syntactic refinement, and (D) global context integration. (E) illustrates a failure case where further iterations cause drift away from a correct early prediction despite initially low NLL.

E Limitations and Future Work

Our work focuses on a simple form of selective looping, where the looped block and adaptive stopping rule are manually specified. While this design already yields consistent improvements across multiple settings, more flexible strategies for allocating computation remain unexplored. In particular, learning where and how much to loop as a function of timestep, token, or input complexity may further improve the efficiency of masked diffusion models.

In addition, our experiments primarily study medium-scale diffusion language models. Recent progress in large-scale MDMs suggests that scaling behavior can change substantially with model size, raising important questions about how selective looping interacts with larger models, longer contexts, and more advanced diffusion architectures. Extending LoopMDM to billion-scale diffusion language models remains an important direction for future work.

Our analysis focuses on attention patterns and controlled reasoning tasks as evidence for how looping interacts with masked positions. Developing more precise theoretical or mechanistic characterizations

Table 7: Datasets and benchmarks used in this work, together with their licenses and usage terms.

Dataset / Benchmark	Usage	License / Usage Terms
Pretraining		
LM1B [8]	Pretraining	Not explicitly specified on the official benchmark page; derived from WMT 2011 News Crawl data.
OpenWebText [21]	Pretraining	CC0-1.0 for dataset packaging.
FineWeb-Edu [51]	Pretraining	ODC-By 1.0; CommonCrawl Terms of Use apply.
Downstream evaluation		
RACE [33]	Downstream evaluation	Non-commercial research use only.
WinoGrande [61]	Downstream evaluation	CC-BY for dataset (version unspecified); Apache-2.0 for code.
HellaSwag [77]	Downstream evaluation	MIT.
Social IQA [62]	Downstream evaluation	CC-BY-4.0.
PIQA [6]	Downstream evaluation	Academic Free License v3.0 (AFL-3.0).
BoolQ [9]	Downstream evaluation	CC BY-SA 3.0.
OpenBookQA [44]	Downstream evaluation	Apache-2.0 for repository; dataset license not separately specified.
ARC-Easy /-Challenge [10]	Downstream evaluation	CC-BY-SA-4.0.
Zero-shot evaluation		
PTB [40]	Zero-shot evaluation	LDC User Agreement for Non-Members.
WikiText [42]	Zero-shot evaluation	CC-BY-SA-4.0.
LAMBADA [49]	Zero-shot evaluation	CC-BY-4.0.
AG News [79]	Zero-shot evaluation	Research / non-commercial use.
Scientific Papers [12]	Zero-shot evaluation	Source-specific OA licenses; repository code under Apache-2.0.

of these effects, particularly regarding why selective looping in early-middle layers is effective, remains an open problem.

Finally, our work studies looping largely in isolation from existing test-time scaling and search methods for diffusion language models. Exploring how selective looping interacts with techniques such as adaptive unmasking [73, 35], remasking [69, 78], or search-based decoding [14, 36] may lead to more effective combinations of architectural and inference-time computation.

F Dataset Licenses and Usage Terms

Table 7 summarizes the datasets and benchmarks used in this work, together with their corresponding licenses and usage terms.

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: The abstract and introduction accurately summarize the main contributions of the paper, including the proposed selective looping design, the matched-compute experimental comparisons, the reasoning improvements beyond depth scaling, and the analysis of masked-position interactions and adaptive inference. The claims are supported by the theoretical discussion and empirical results presented in the paper.

   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: See Appendix E

   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:[Yes]

Justification: See Appendix B

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: See Appendix C

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:[No]

   Justification: We commit to releasing the code upon acceptance. In the meantime, we provide detailed experimental settings, model configurations, training procedures, and evaluation protocols in Appendix C to support reproducibility. Our method is also straightforward to implement, requiring only the insertion of a looped shared block and stochastic loop-count sampling into existing masked diffusion model codebases. For the language modeling experiments in Sections 4.1 and 4.3, we build on the public implementations of Sahoo *et al.* [59]. For the math experiments in Section 4.2, we follow the implementation of Kim et al. [31].

   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: See Appendix C

   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: We report variance for various experiments.

   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: The paper provides detailed experimental settings in Appendix C, including model configurations, training schedules, batch sizes, sequence lengths, optimizer settings, and GPU resources (NVIDIA A100 GPUs). We additionally report matched training FLOPs, training iterations, and evaluation protocols to support reproducibility.

   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: The authors have reviewed the NeurIPS Code of Ethics and believe that the research conducted in this work complies with its principles in all relevant respects.

   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: This work is methodological and is not tied to a deployed system or a specific high-risk application. Its primary positive impact is improving both the performance and training efficiency of masked diffusion language models, which may reduce computational cost and make such research more accessible. A potential negative impact is the general dual-use risk associated with improving generative language models, although this work does not introduce a new deployed model, dataset, or application-specific capability.

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: This work does not release new datasets, pretrained foundation models, or generative systems with elevated misuse risk. The experiments are conducted using existing publicly available datasets and baseline model implementations intended for research use.

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: All datasets, models, and codebases used in this work are properly cited and credited in the main paper and Appendix F. We additionally provide a summary of dataset licenses and usage terms, and follow the corresponding conditions for research use.

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: [N/A]

    Justification: This paper does not introduce or release new datasets.

    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: This work does not involve human subjects, user studies, crowdsourcing, medical data, or any form of participant interaction requiring IRB approval or informed consent.

    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: This work does not involve human subjects, user studies, crowdsourcing, or personally identifiable data, and therefore does not require IRB approval or participant consent.

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: Large language models were not used as part of the proposed method, training pipeline, evaluation protocol, or experimental analysis. Any use of LLM-based tools was limited to writing assistance and language editing, and did not affect the scientific content, methodology, or results of the paper.

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.