VoidPadding: Let [VOID] Handle Padding in Masked Diffusion Language Models so that [EOS] Can Focus on Semantic Termination

Anonymous ACL submission

Abstract

Masked diffusion language models (MDLMs) generate text by denoising a preallocated masked response canvas, making response-length modeling central to instruction tuning. Existing MDLMs often inherit the autoregressive convention of using repeated [EOS] tokens for padding during instruction tuning, giving [EOS] a dual role as both a semantic terminator and a padding token. We show that this dual role is a root cause of [EOS] overflow under large-block decoding. To decouple these roles, we propose VoidPadding, which introduces [VOID] for padding and reserves [EOS] for termination. During inference, the learned [EOS] signal enables early stopping, while the learned [VOID] signal guides adaptive response canvas expansion. On Dream-7B-Instruct, VoidPadding improves the block-size-averaged four-task mean across mathematical reasoning and code generation benchmarks by $+17.84$ points over the original model and $+6.95$ points over RainbowPadding, while reducing decoding NFE by 55.7% on average.

1 Introduction

Masked diffusion language models (Austin et al., 2021a; Wang et al., 2025; Lou et al., 2023; Liu et al., 2025) (MDLMs) offer a promising alternative to autoregressive language models (Radford et al., 2018) (ARLMs). Whereas ARLMs generate tokens sequentially from left to right, MDLMs use bidirectional attention to denoise the entire sequence in parallel (Bie et al., 2026). This bidirectional conditioning, however, makes MDLMs sensitive to the padding tokens introduced in instruction tuning.

During instruction tuning (Chung et al., 2024), instruction–response pairs are typically processed in batches for efficiency and padded to the maximum sequence length within each mini-batch. Following a convention inherited from ARLM training, MDLMs often use repeated [EOS] tokens as padding (Nie et al., 2026). In ARLMs, causal masking (Vaswani et al., 2017) prevents earlier positions from conditioning on [EOS] padding, so [EOS] primarily serves as a semantic terminator. In MDLMs, however, [EOS] padding is also included in the training target. Thus, [EOS] takes on two roles: the semantic terminator role and the padding role.

During inference, MDLMs generate on a predefined response canvas whose positions are jointly modeled under bidirectional attention (Cheng et al., 2025). The padding role of [EOS] introduced during instruction tuning therefore offers a length signal: high [EOS] confidence near the tail indicates that the current canvas may exceed the desired output length (Li et al., 2025a; Yang et al., 2026). However, this signal becomes fragile under large-block decoding. Once high-confidence tail [EOS] tokens are generated, they become part of the denoising context and can further increase nearby [EOS] probabilities, leading to short effective responses and repeated [EOS]. This failure mode is known as [EOS] overflow (Kim et al., 2025a).

Based on these insights, we propose **VoidPadding**, a lightweight training–inference co-design that introduces [VOID] as a new padding token and reserves [EOS] solely for semantic termination. Training sequences are formatted as [prompt] [response] [EOS] [VOID]*, where each example contains a single [EOS] followed by [VOID] padding. During inference, [VOID] generation is banned. We introduce **[EOS]Termination**, which stops decoding once no masks remain before the leftmost visible [EOS]. This prevents [EOS] overflow and avoids redundant generation after the effective response. Additionally, we propose an optional inference-time **VoidExpansion** strategy, which uses the [VOID] length signal to expand insufficient canvases before denoising.

Our contributions are:

1. We systematically analyze the mechanism be-

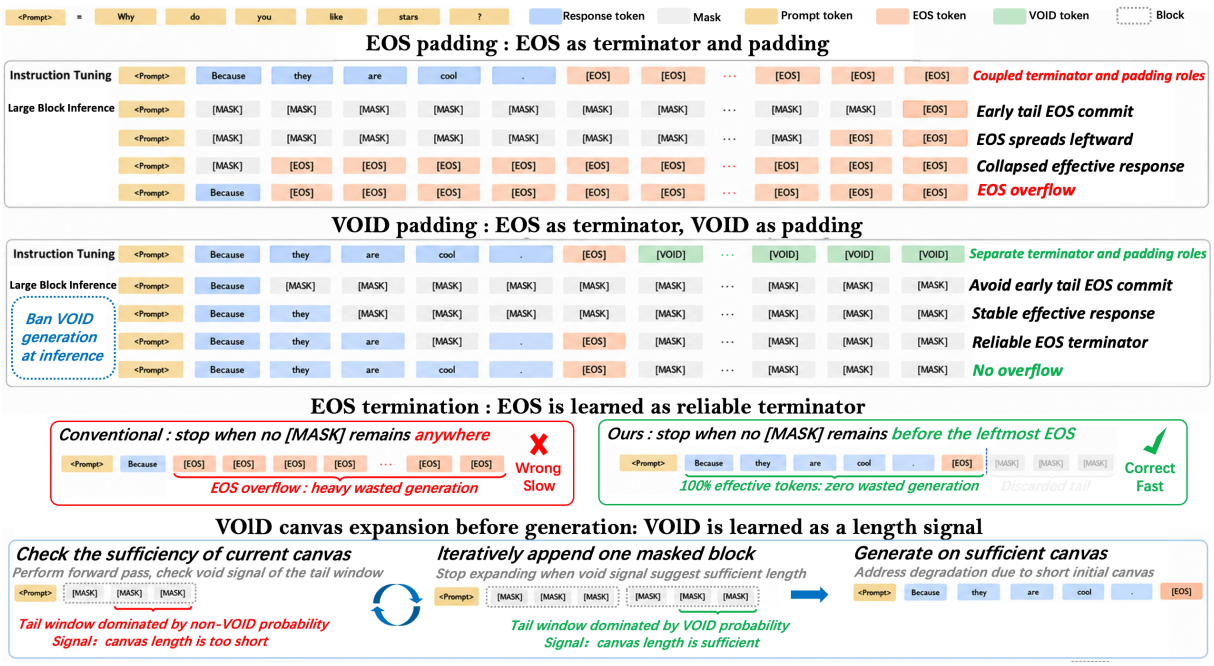

Figure 1: Overview of VoidPadding. VoidPadding decouples termination from padding by using [EOS] only for stopping and [VOID] only for padding. During inference, [VOID] token generation is banned, decoding stops at the leftmost visible [EOS], and the canvas may be expanded based on tail [VOID] signals.

hind [EOS] overflow and show that it stems from the dual use of [EOS] for termination and padding during instruction tuning.

2. We propose VoidPadding, a training–inference co-design that decouples the two roles by using [VOID] for padding and reserving [EOS] for termination, enabling [EOS]Termination for early stopping and VoidExpansion for adaptive canvas expansion.

3. Experiments show that VoidPadding prevents [EOS] overflow, improves generation quality and decoding efficiency, and remains robust across block sizes and canvas lengths.

2 Preliminary

2.1 Masked Diffusion Modeling

MDLMs model generation as a denoising process over discrete sequences (Sahoo et al., 2024). Let $\mathcal{V} = \{1, \ldots, V\}$ be the token vocabulary, $\mathsf{M} \notin \mathcal{V}$ the mask token, and $\mathbf{z}_t \in (\mathcal{V} \cup \{\mathsf{M}\})^L$ the sequence state at diffusion step $t \in \{0, \ldots, T\}$, where T is the total number of denoising steps. Here, $t = 0$ denotes clean data and $t = T$ the fully masked sequence. The masked and visible positions are denoted by $\mathcal{M}_t = \{i : z_t^i = \mathsf{M}\}$ and $\bar{\mathcal{M}}_t = \{1, \ldots, L\} \setminus \mathcal{M}_t$, respectively.

The forward process gradually corrupts a clean sequence $\mathbf{z}_0 \sim p_{\text{data}}$ by independently masking each position over T steps. At each step, tokens are progressively replaced by M according to a fixed schedule, so that $\mathbf{z}_T$ is fully masked. During training, a timestep t is sampled and the model predicts the original token at each masked position, yielding the denoising cross-entropy objective $\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{z}_0, t, \mathbf{z}_t}\left[\sum_{i \in \mathcal{M}_t} -\log p_\theta(z_0^i \mid \mathbf{z}_t)\right]$. This objective trains the denoiser on various masking patterns, enabling p_θ to predict masked tokens from different visible contexts (Kim et al., 2025c).

MDLM decoding procedures include block diffusion (Arriola et al., 2025) and pure diffusion (Ye et al., 2025). Both of them start from a fully masked canvas of length L with block size B, where pure diffusion corresponds to $B = L$. At each step, the model uses bidirectional attention to condition on the whole sequence and predicts $p_\theta(\cdot \mid \mathbf{z}_t)$ for all $i \in \mathcal{M}_t$ in parallel, so one denoising step corresponds to one model forward pass. Decoding then proceeds in two steps within the current block. First, it selects commitment positions, commonly using confidence (Cai and Li, 2026) or negative entropy scores (Ben-Hamu et al., 2026). Second, it selects tokens for these positions, typically by argmax under p_θ. Decoding moves to the next block once the current block is fully committed

and terminates when all masks have been filled.

2.2 Instruction-Tuning Data Construction

Instruction tuning (IT) trains pretrained language models on prompt–response pairs, improving instruction following and zero-shot generalization (Chung et al., 2024). For efficient batching, prompt–response sequences are terminated by [EOS] and padded to a common length. MDLM IT often follows the ARLM convention of using [EOS] as the padding token. Let $B = \{(p_k, r_k)\}_{k=1}^{b}$ be a mini-batch of size b, where p_k is the prompt of length P_k and r_k is the raw response of length ℓ_k. Over the IT data distribution, let ℓ denote the raw response length before the terminal [EOS], and define its CDF as $F(m) = \Pr(\ell \leq m)$. After appending one semantic terminator [EOS], each full sequence is padded with [EOS] to the maximum full-sequence length in the mini-batch, $S_B = \max_{1\leq k\leq b}(P_k + \ell_k + 1)$. Thus, example k has padded response $y_k = r_k \oplus \texttt{[EOS]} \oplus \texttt{[EOS]}^{L_{B,k}-\ell_k-1}$ of length $L_{B,k} = S_B - P_k$. During IT, both noise and loss are applied only to y_k.

3 Analysis of [EOS] Overflow

[EOS] overflow is a common issue of MDLM inference under large block sizes (Kim et al., 2025a). In a typical case, the model produces a very short answer followed by repeated [EOS] tokens, as shown in Figure 3(a). However, [EOS] overflow is not observed in ARLMs, even though both ARLMs and MDLMs use [EOS] as padding tokens during IT. This motivates us to ask: Why is [EOS] padding safe in ARLMs but fragile in MDLMs?

Observation 1: [EOS] overflow is not caused by limited large-block generation capacity.

We first test whether [EOS] overflow is caused by insufficient large-block generation capacity. Specifically, we ban [EOS] commitment in LLaDA-8B-Instruct by setting the [EOS] logit to $-\infty$ throughout decoding, and then evaluate on GSM8K (Cobbe et al., 2021) with $L = B = 512$. Although this removes the semantic stopping signal and often causes repeated reasoning traces and answers, it raises the accuracy of the first extracted answer from 28.43% to 70.66%. Thus, the model is not inherently unable to generate large blocks.

Observation 2: Training with [EOS] padding teaches [EOS] to act as a length signal.

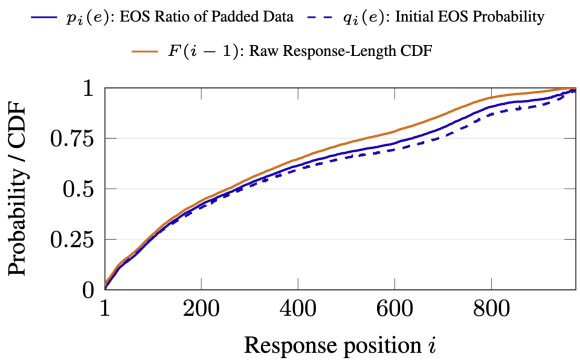

Figure 2: [EOS] as a learned length signal. Initial [EOS] confidence aligns with the [EOS] label ratio of padded data and the raw response-length CDF.

Next, we examine whether [EOS] padding provides an additional signal in MDLM training beyond semantic termination. Specifically, we train an [EOS]-padding diagnostic model on LLaDA-8B-Base, with training details provided in Appendix A. Following Section 2.2, for each mini-batch $B = \{(p_k, r_k)\}_{k=1}^{b}$, we construct the padded response $y_k = r_k \oplus \texttt{[EOS]} \oplus \texttt{[EOS]}^{L_{B,k}-\ell_k-1}$, where $L_{B,k} = S_B - P_k$ and $S_B = \max_{1\leq k\leq b}(P_k + \ell_k + 1)$. We define response-side positions using a 1-indexed coordinate $i \in \{1, \ldots, 1024\}$. For the padded data distribution, we record the raw response-length CDF $F(i-1) = \Pr(\ell \leq i-1)$ defined in Section 2.2, along with the [EOS] token ratio $p_i(e)$ at response-side position i. For the model distribution, we use the same batch-padded examples, mask the entire padded response y_k while keeping the prompt p_k visible, and report the initial [EOS] probability $q_i(e)$ from a single forward pass through the diagnostic model. As shown in Figure 2, the three curves closely align, $F(i-1) \approx p_i(e) \approx q_i(e)$, indicating that [EOS] padding teaches the model to associate [EOS] with desirable response length.

Observation 3: Large–block inference converts the [EOS] length signal into overflow.

We analyze a large-block decoding trajectory of LLaDA-8B-Instruct with $L = B = 512$. Figure 3(a) shows a two-stage [EOS] overflow pattern. In Stage 1, on the initial fully masked canvas, tail positions have very high [EOS] probability and [EOS] is committed in the initial steps. In Stage 2, on a canvas with a committed [EOS] suffix, [EOS] commitments move leftward: positions before the suffix also take [EOS] as their committed token, producing overflow. We then formalize this pat-

tern under the idealized version of Observation 2. Let i index response-side positions and let e denote [EOS]. For a training-time EOS-padded target, let $\bar{Y}_i$ be the token at position i, and write $p_i(e) = \Pr(\bar{Y}_i = e)$. During inference, let q_i be the model distribution at masked canvas position i, let $\hat{y}_i = \arg\max_v q_i(v)$ be its proposed token, and let Y_i be the token committed at canvas position i.

For Stage 1, on a fully masked response canvas, we idealize Observation 2 as

$$q_i(e) = p_i(e) = F(i-1).$$

Since standard MDLM decoding uses argmax token proposals (Section 2.1), $q_i(e) > 1/2$ implies $\hat{y}_i = e$. Assume the canvas is long enough that $F(L-1) > 1/2$. On the initial fully masked response canvas, we call the suffix of positions with $q_i(e) > 1/2$ the high-confidence [EOS] tail, and define its beginning as

$$r = \min\{i \in \{1, \ldots, L\} : F(i-1) > 1/2\}.$$

For Stage 2, let $S_e = \{Y_r = \cdots = Y_L = e\}$ denote the committed [EOS] suffix event during inference, and let $\bar{S}_e = \{\bar{Y}_r = \cdots = \bar{Y}_L = e\}$ denote the corresponding training-time [EOS] padding suffix event. We assume that after S_e occurs, the model's [EOS] probability for earlier positions matches the corresponding training-time [EOS]-padded conditional probability: for any $i < r$,

$$q_i(e \mid S_e) = p_i(e \mid \bar{S}_e) = \Pr(\bar{Y}_i = e \mid \bar{S}_e).$$

Theorem 1 (Stage 1: initial [EOS] commitments)**.** If the first block reaches the high-confidence [EOS] tail, $r \leq B$, then every selected position $j \in \{r, \ldots, B\}$ is committed as [EOS], i.e., $Y_j = e$.

Theorem 2 (Stage 2: leftward spread of [EOS] commitments)**.** If S_e has occurred, then for any $i < r$ satisfying $2F(i-1) > F(r-1)$, selecting position i in a later denoising step yields $Y_i = e$.

Proofs of the theorems are given in Appendix B.

We next perform an intervention motivated by Theorem 2: after a position is committed as [EOS], we set attention scores to that position to $-\infty$, as shown in Figure 3(b). Initial [EOS] commitments remain, but the model avoids [EOS] overflow and generates the correct answer. This shows that Stage 2 is the dominant driver of [EOS] overflow.

These observations reveal that [EOS] overflow stems from a training–inference role mismatch. In MDLM IT, bidirectional attention exposes padding

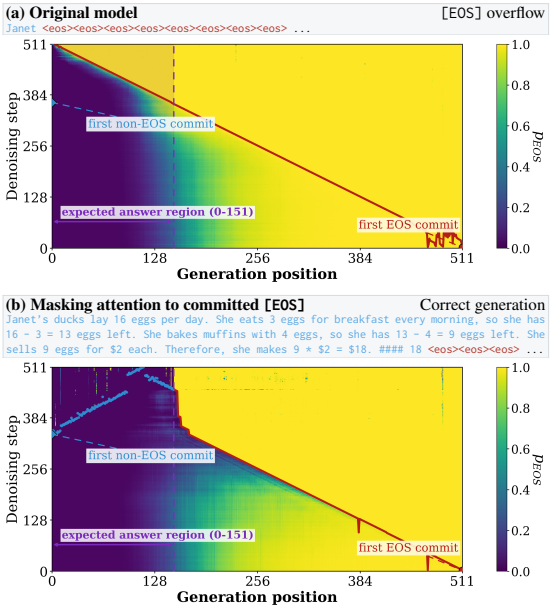

Figure 3: Attention intervention on the $B = L = 512$ [EOS] overflow trajectory. Both cases use the same prompt. Masking attention to committed [EOS] tokens avoids [EOS] overflow and recovers the correct answer.

[EOS] tokens as denoising targets, so [EOS] is trained as both a semantic terminator and a padding token. During inference, however, [EOS] is expected to serve only as a semantic terminator. In ARLM IT, causal masking hides padding [EOS] tokens from earlier positions, so [EOS] remains a semantic terminator and overflow does not arise.

4 Method

Motivated by this training–inference role mismatch, we propose **VoidPadding**, which introduces a new [VOID] token as the padding token, so that [EOS] serves only as the semantic terminator. We further design **VoidExpansion** for inference, which uses the [VOID] length signal learned during IT to expand an insufficient canvas before decoding. Together, this training–inference co-design decouples padding from semantic termination while preserving the hidden length signal carried by [VOID] tokens. Figure 1 provides an overview of the method, with details given in the following subsections.

4.1 VoidPadding

Instruction-Tuning Design. VoidPadding follows the IT data construction described in Section 2.2, except that padding [EOS] tokens are replaced by [VOID] tokens. Thus, for $B = \{(p_k, r_k)\}_{k=1}^{b}$ and $L_{B,k} = S_B - P_k$, the padded response is

$$y_k^{\text{VOID}} = r_k \oplus \texttt{[EOS]} \oplus \texttt{[VOID]}^{L_{B,k} - \ell_k - 1}.$$

We then follow the standard MDLM training procedure in Section 2.1, with noise injection and loss computation performed on y_k^{VOID}. Notably, we apply standard MDLM masking for non-[EOS] positions. The single [EOS] token in each example is always masked, forcing the model to predict it as a stable semantic terminator.

Inference Design. During inference, [VOID] remains a hidden padding signal and is never generated: setting its logit to $-\infty$ before token selection prevents the high-confidence [VOID] tail from becoming visible output. We also propose **[EOS]Termination**, which stops decoding once all positions before the leftmost visible [EOS] are unmasked and then discards the rest of the canvas. In contrast, **VanillaStopping**, the standard MDLM stopping rule, stops only after all masked positions are decoded (Section 2.1), causing redundant decoding after the semantic endpoint.

4.2 VoidExpansion

As discussed in Section 2.1, MDLM decoding starts from a pre-defined response canvas with fixed length L and block size B. When this canvas is too short, generation quality drops sharply (Appendix Figure 8). Motivated by this short-canvas degradation, VoidExpansion uses the learned [VOID] length signal to expand the canvas to a sufficient, block-aligned length $\hat{L} = nB$ before decoding.

Similar to how IT with [EOS] padding induces a length signal in [EOS] (Observation 2), IT with [VOID] padding induces a [VOID] length signal. Specifically, sufficient canvases have redundant tails that prefer [VOID], whereas insufficient canvases retain tail positions that favor non-[VOID] tokens. We therefore quantify this [VOID] length signal as follows. Let p denote the prompt tokens, M the mask token, and w the tail-window size. At canvas length ℓ, we run one forward pass on $p \oplus \mathsf{M}^\ell$ and define the tail window $\mathcal{T}(\ell)$ as the last $\min(w, \ell)$ response-canvas positions, indexed by i:

$$p_i^{\text{void}} = p_\theta(x_i = \texttt{[VOID]} \mid p \oplus \mathsf{M}^\ell),$$

$$p_i^{\text{nonvoid}} = \max_{a \in \mathcal{V} \setminus \{\texttt{[VOID]}\}} p_\theta(x_i = a \mid p \oplus \mathsf{M}^\ell),$$

$$s_{\text{nonvoid}}(\ell) = |\mathcal{T}(\ell)|^{-1} \sum_{i \in \mathcal{T}(\ell)} p_i^{\text{nonvoid}},$$

$$s_{\text{gap}}(\ell) = |\mathcal{T}(\ell)|^{-1} \sum_{i \in \mathcal{T}(\ell)} \left(p_i^{\text{void}} - p_i^{\text{nonvoid}} \right).$$

Here s_{nonvoid} measures tail preference for non-[VOID] tokens, while s_{gap} measures tail preference for [VOID] over the strongest non-[VOID] alternative at each position. Thus, high s_{nonvoid} or low s_{gap} suggests that the current canvas is insufficient.

Algorithm 1 VoidExpansion Before Decoding

Require: Prompt tokens p, model f_θ, initial canvas length L_0, maximum length $L_{\max}$, block size B, tail window w, thresholds τ_{nonvoid} and τ_{gap}
Ensure: Selected generation length $\hat{L}$

```
L ← ⌈L_0/B⌉B, L_cap ← ⌈L_max/B⌉B
while L ≤ L_cap do
    Construct x = p ⊕ M^L
    z ← f_θ(x)_resp                ▷ logits on response positions
    T(L) ← last min(w, L) positions of z
    Compute s_nonvoid(L) and s_gap(L) on T(L)
    if s_nonvoid(L) ≤ τ_nonvoid and s_gap(L) ≥ τ_gap then
        return L
    end if
    L ← L + B                      ▷ append one block before generation
end while
return L_cap
```

VoidExpansion uses this [VOID] length signal to expand the insufficient canvas, as shown in Algorithm 1. It defines a canvas as sufficient when $s_{\text{nonvoid}}(L) \leq \tau_{\text{nonvoid}}$ and $s_{\text{gap}}(L) \geq \tau_{\text{gap}}$. Starting from the block-aligned initial canvas length $L = \lceil L_0/B \rceil B$, VoidExpansion checks this criterion at the current L. If the criterion is not met, it expands the canvas by one block and repeats until the criterion is met or the maximum length restriction $L_{\text{cap}} = \lceil L_{\max}/B \rceil B$ is reached.

5 Experiments

Setup. Our main experiments use two instruction-tuned MDLMs: LLaDA-8B-Instruct (hereafter LLaDA) (Nie et al., 2026) and Dream-7B-Instruct (hereafter Dream) (Ye et al., 2025). Quality metrics are accuracy on GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021), and pass@1 on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b). The main efficiency metric is the average number of forward passes per example (hereafter NFE). In all tables, bold marks the best result and underline marks the second best within each comparison. For Dream, we use a decoding wrapper to enable the same block-wise decoding protocol as LLaDA; details are provided in Appendix C. We compare against the original models and RainbowPadding (Kim et al., 2025a), which aims to avoid [EOS] overflow by cyclically using 7 distinct padding tokens. On both LLaDA and Dream, RainbowPadding and VoidPadding use LoRA SFT (Hu et al., 2022) on the same 0.5M-example data and identical training configurations,

except that VoidPadding uses 67% of RainbowPadding's training steps. Detailed training configuration is provided in Appendix A.

5.1 Fixed-Length Block-Size Robustness

We first evaluate fixed-length robustness with generation length $L = 512$ and a maximum denoising-step budget $T = 512$, while varying the block size B. Table 1 reports $B \in \{64, 128, 512\}$. Complete block-size results are in Appendix D. We compare against the original LLaDA and Dream models, as well as RainbowPadding applied to these models.

Accuracy Robustness. Table 1 shows that VoidPadding improves fixed-length robustness across both models. On LLaDA, `[VOID]` substantially mitigates large-block `[EOS]` overflow: the original model drops sharply from 76.80 at $B = 64$ to 28.43 at $B = 512$ on GSM8K, while `[VOID]` remains around 78. It gives the strongest mean on three of four tasks and remains competitive on HumanEval. Dream shows the same trend: the original model degrades as B grows, while `[VOID]` remains stable and attains the best mean score across tasks. Across all block sizes, `[VOID]` attains the highest four-task mean. On LLaDA, the block-size-averaged macro mean over four tasks improves by $+7.06$ points over the original model and $+3.59$ over Rainbow. On Dream, the corresponding block-size-averaged macro means are $+17.84$ and $+6.95$, respectively.

`[EOS]`Termination Efficiency. Table 2 reports the average NFE of Dream variants in Table 1 over $B \in \{64, 128, 512\}$. The original model and RainbowPadding use VanillaStopping, which fills the entire 512-token canvas. Since each denoising step requires one model forward pass, this setting requires 512 NFE under $T = 512$. VoidPadding instead uses `[EOS]`Termination, which reduces NFE by 69.3% on GSM8K, 27.5% on MATH500, 41.3% on HumanEval, and 84.6% on MBPP, averaged over $B \in \{64, 128, 512\}$. Appendix E reports per-block NFE results for both LLaDA and Dream.

5.2 Effectiveness of VoidExpansion

We next evaluate VoidExpansion on a short initial canvas and compare it against two inference-only variable-length generation methods: ρ-`[EOS]` (Yang et al., 2026) and DAEDAL (Li et al., 2025a). Both methods adjust the canvas using the `[EOS]` signal of the original model. In the main short-canvas experiment, all methods start from an initial length $L_0 = 64$. VoidExpansion uses a maximum length $L_{\max} = 512$, block size $B = 32$, and tail-window size $w = 16$. On LLaDA, fixed-L_0 decoding baselines are evaluated on both the original model and the `[VOID]`-padding checkpoint. We apply ρ-`[EOS]` and DAEDAL to the original model, and VoidExpansion to the `[VOID]`-padding checkpoint, comparing our `[VOID]`-based training–inference co-design with `[EOS]`-based inference-only length selection. For fixed-L_0 decoding and VoidExpansion, the maximum denoising-step budget is $T = L_0$ and $T = \hat{L}$, respectively, where $\hat{L}$ is the canvas length after VoidExpansion. For ρ-`[EOS]` and DAEDAL, we follow their adaptive decoding procedures. In all cases, NFE counts only decoding forward passes, while wall-clock time includes both canvas selection and decoding. Since the official implementations of ρ-`[EOS]` and DAEDAL do not support Dream, Dream is compared only against fixed-L_0 decoding.

VoidExpansion preserves quality while improving efficiency. Table 3 shows that fixed-L_0 decoding reaches only 39.38 mean accuracy for the original LLaDA and 38.61 for the `[VOID]`-padding checkpoint. Thus, VoidPadding alone does not improve short-canvas decoding. The benefit appears when the learned `[VOID]` signal is used for expansion. The full VoidPadding+VoidExpansion method slightly improves over DAEDAL in mean quality (51.34 vs. 50.77), while lowering mean NFE from 228.82 to 172.10 and running $2.12\times$ faster in geometric-mean wall time. VoidExpansion also achieves higher quality than ρ-`[EOS]` on all benchmarks while attaining a larger geometric-mean wall-clock speedup. Raw wall-clock results are shown in Figure 7. Table 4 shows the same pattern on Dream: VoidExpansion achieves the best score on all four benchmarks, improving the mean score over fixed-L_0 Dream (42.23) and fixed-L_0 VoidPadding (39.20) to 60.37.

To test large-block robustness, we set $L_0 = 256$ and $B = 256$. We apply ρ-`[EOS]` and DAEDAL to the original LLaDA model, and compare them with VoidExpansion on the VoidPadding checkpoint. Figure 4 shows that DAEDAL collapses to a 23.46 mean score, including 13.12 on GSM8K and 12.80 on MATH500. ρ-`[EOS]` is more stable but still underperforms VoidExpansion on every benchmark, with a 9.10-point lower mean score (40.94 vs. 50.04). This confirms that inference-only methods relying on the original `[EOS]` signal remain vulnerable to large-block `[EOS]` overflow.

We further evaluate the effectiveness of VoidPadding and VoidExpansion on LLaDA-8B-Base

Task	Method	Block size B			Mean
		64	128	512	
GSM8K	Original	76.80	70.96	28.43	58.73
	RainbowPadding	74.53	73.39	66.57	71.50
	VoidPadding	**77.79**	**78.39**	**78.62**	**78.27**
MATH500	Original	39.60	37.60	16.60	31.27
	RainbowPadding	33.20	34.00	**27.80**	31.67
	VoidPadding	**40.60**	**41.60**	27.60	**36.60**
HEval	Original	42.68	**43.90**	**44.51**	**43.70**
	RainbowPadding	**43.90**	43.29	35.98	41.06
	VoidPadding	41.46	42.07	41.46	41.66
MBPP	Original	41.48	39.12	37.68	39.43
	RainbowPadding	42.71	43.02	42.51	42.75
	VoidPadding	**44.66**	**44.87**	**44.97**	**44.83**
Mean over tasks	Original	50.14	47.90	31.81	43.28
	RainbowPadding	48.59	48.43	43.22	46.75
	VoidPadding	**51.13**	**51.73**	**48.16**	**50.34**

(a) LLaDA

Task	Method	Block length B			Mean
		64	128	512	
GSM8K	Original	73.39	33.89	46.70	51.33
	RainbowPadding	79.15	77.63	52.16	69.65
	VoidPadding	**80.44**	**80.89**	**78.09**	**79.81**
MATH500	Original	43.20	41.20	26.00	36.80
	RainbowPadding	33.60	33.40	31.40	32.80
	VoidPadding	**44.00**	**44.20**	**45.20**	**44.47**
HEval	Original	**62.80**	**60.37**	18.90	47.36
	RainbowPadding	**62.80**	**60.37**	50.00	57.72
	VoidPadding	59.76	59.15	**60.37**	**59.76**
MBPP	Original	41.00	31.00	30.00	34.00
	RainbowPadding	56.00	54.80	47.80	52.87
	VoidPadding	**57.60**	**58.00**	**54.80**	**56.80**
Mean over tasks	Original	55.10	41.62	30.40	42.37
	RainbowPadding	57.89	56.55	45.34	53.26
	VoidPadding	**60.45**	**60.56**	**59.62**	**60.21**

(b) Dream

Table 1: Fixed-length generation robustness experiments on LLaDA and Dream with $B \in \{64, 128, 512\}$.

Benchmark	Original/Rainbow	VoidPadding	NFE Reduction
GSM8K	512.0	157.3	69.3%
MATH500	512.0	371.0	27.5%
HumanEval	512.0	300.7	41.3%
MBPP	512.0	78.7	84.6%

Table 2: NFE for Dream variants in Table 1, averaged over $B \in \{64, 128, 512\}$.

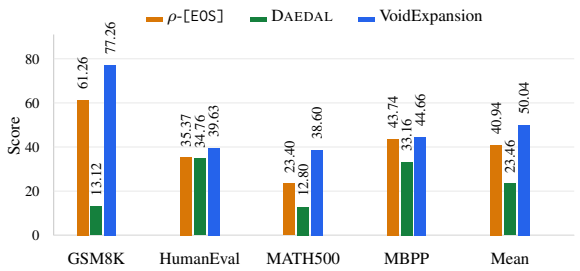

Figure 4: Large-block stress test with $L_0 = 256$ and $B = 256$. Lower scores indicate [EOS] overflow.

and Dream-7B-Base in Appendices F and G.

6 Ablation Studies

Setup. Our ablation experiments use the same benchmarks and metrics as Section 5 and compare the original LLaDA-8B-Instruct model with the RainbowPadding and VoidPadding checkpoints.

VoidPadding makes [EOS]Termination effective. Table 5 evaluates [EOS]Termination on a fixed 512-token canvas across different training settings. Relative to the VanillaStopping baseline defined in Section 4.1, [EOS]Termination reduces NFE for LLaDA and RainbowPadding but slightly degrades accuracy. In contrast, with VoidPadding, [EOS]Termination improves accuracy while reducing mean NFE by 63.8% across block sizes. This improvement indicates that VoidPadding aligns training and inference so that [EOS] becomes a reliable semantic terminator for early stopping. Per-benchmark NFE results are shown in Figure 17.

VoidExpansion is robust to hyperparameter choices. Starting from the VoidExpansion setting in Table 3, Table 6 varies one factor at a time and keeps the remaining settings fixed. The varied factors include threshold pairs, initial lengths L_0, tail window sizes w, and block sizes B. Mean quality and mean NFE over the four tasks remain stable across these changes, showing that the learned [VOID] length signal supports short-canvas expansion while preserving both quality and efficiency. See Appendix H for detailed results behind Table 6.

7 Related Work

Diffusion Language Models. Diffusion language models (DLMs) can be broadly categorized into continuous DLMs (Guo et al., 2026; Hu et al., 2026; Jo and Hwang, 2026; Shen et al., 2026), which operate in embedding spaces, and discrete DLMs (Hoogeboom et al., 2021; Sahoo et al., 2025; Do et al., 2025), which operate over discrete token spaces. Masked diffusion language models (Hersche et al., 2025; He et al., 2023; Austin et al., 2021a) (MDLMs) are a representative discrete class that iteratively predicts masked tokens. Recent large-scale MDLMs, including LLaDA series (Nie et al., 2026; Zhu et al., 2025; Bie et al., 2025, 2026), Dream series (Ye et al., 2025; Wu et al., 2026b),

Method	Fine-tuned	Expansion Signal	GSM8K	MATH500	HEval	MBPP	Mean
LLaDA fixed-64	No	None	60.20	26.80	29.27	41.27	39.38
VOID fixed-64	Yes	None	52.84	26.20	36.59	38.81	38.61
ρ-[EOS]	No	[EOS]	73.39	27.00	40.24	41.38	45.50
DAEDAL	No	[EOS]	**79.38**	37.40	**44.51**	41.79	50.77
VoidExpansion	Yes	[VOID]	78.32	**39.60**	42.07	**45.38**	**51.34**

(a) Task quality under a short initial canvas. Mean arithmetically averages the four benchmark scores.

Method	Mean Acc. ↑	GSM8K NFE/Wall	MATH NFE/Wall	HEval NFE/Wall	MBPP NFE/Wall	Mean NFE/Wall
DAEDAL	50.77	135.54 / 1.00×	269.65 / 1.00×	254.66 / 1.00×	255.43 / 1.00×	228.82 / 1.00×
ρ-[EOS]	45.50	**92.57** / **1.56×**	**104.30** / **3.04×**	191.52 / 1.34×	170.41 / 1.69×	**139.70** / 1.81×
VoidExpansion	**51.34**	133.78 / 0.92×	418.05 / 0.82×	**73.06** / **5.59×**	**63.50** / **4.80×**	172.10 / **2.12×**

(b) Efficiency of adaptive length selection. Each task cell reports NFE / wall speedup. Wall speedup is normalized to DAEDAL. In the Mean column, NFE is the arithmetic mean and wall speedup is the geometric mean over the four benchmarks.

Table 3: Short canvas expansion results on LLaDA. VoidExpansion uses $\tau_{\text{nonvoid}} = 0.09$, $\tau_{\text{gap}} = -0.45$.

Method	GSM8K	MATH500	HEval	MBPP	Mean
Dream fixed-64	53.22	22.80	43.90	49.00	42.23
VOID fixed-64	42.53	19.00	42.68	52.60	39.20
VoidExpansion	**79.08**	**44.00**	**60.98**	**57.40**	**60.37**

Table 4: Short canvas expansion results on Dream. VoidExpansion uses $\tau_{\text{nonvoid}} = 0.15$, $\tau_{\text{gap}} = 0.55$.

Setting	Mean score			Mean NFE reduction
	VanillaStopping	[EOS]Termination	Δ	
LLaDA	43.28	43.15	-0.13	15.3%
RainbowPadding	46.75	46.74	-0.01	44.7%
VoidPadding	40.70	50.34	+9.64	63.8%

Table 5: [EOS]Termination ablation on LLaDA. Mean scores and mean NFE are averaged over the four benchmarks and $B \in \{64, 128, 512\}$. NFE reduction is relative to VanillaStopping.

Factor varied	Values tested	Task-mean quality range	Task-mean NFE range
Thresholds	16 threshold pairs (4×4)	51.16–51.78	169.39–175.50
Initial length L_0	64, 96, 128, 160	50.83–51.78	172.10–172.80
Tail window size w	8, 16, 24, 32	51.40–51.81	168.65–173.15
Block size B	16, 32, 64	50.80–51.78	170.74–172.35

Table 6: Hyperparameter robustness of VoidExpansion. Each row varies one factor from the setting in Table 3 while keeping the others fixed. Quality and NFE are averaged over the four benchmarks. Each range shows the four-benchmark mean quality or NFE obtained across the values of that factor listed in the row.

SDAR (Cheng et al., 2025), TraDo (Wang et al., 2025), and WeDLM (Liu et al., 2025), show strong complex reasoning and long-text generation abilities.

Efficient MDLMs. Although MDLMs support parallel token prediction, inference remains costly because generation usually requires many denoising iterations with full-sequence bidirectional attention. Work on efficient MDLM inference includes sampling methods that reduce denoising steps or finalize more tokens per step (Li et al., 2025b; Israel et al., 2026; Wei et al., 2025b), cache-based methods that reuse computation via approximate caching or KV caching (Wu et al., 2025; Hu et al., 2025; Song et al., 2026; Wu et al., 2026a; Luo et al., 2026; Wei et al., 2025a), and training-based distillation for fewer-step generation (Monsefi et al., 2025; Amin et al., 2026; Chen et al., 2025).

Variable-Length Generation. MDLMs often require a preallocated output canvas, wasting computation on over-allocated slots and limiting usability (Han et al., 2023). Length allocation methods address this issue (Cheng et al., 2026). Inference-only methods, including SmartCrop (Rossi et al., 2026), ρ-[EOS] (Yang et al., 2026), and DAEDAL (Li et al., 2025a), typically use [EOS] as a length signal, while training-based methods include dLLM-Var (Yang et al., 2025), which learns reliable [EOS] prediction for block-wise termination, and FlexMDM (Kim et al., 2025b), which decouples length prediction from token prediction.

8 Conclusion

We show that a root cause of [EOS] overflow in MDLMs is a training–inference role mismatch: [EOS] is trained as both padding and semantic terminator, but used only for termination at inference. VoidPadding instead uses [VOID] as padding tokens and reserves [EOS] for semantic endings, enabling VoidExpansion for canvas allocation and [EOS]Termination for early stopping. Together, our training–inference co-design improves large-block robustness, decoding efficiency, and variable-length generation.

Limitations

Many MDLM training and inference conventions are inherited from ARLMs. Our study focuses on one concrete failure mode of this transfer: [EOS] padding, which is largely benign under causal masking, can become fragile under bidirectional denoising because padding positions can serve as both context and denoising targets. Thus, [EOS] overflow should be viewed as one symptom of a broader class of ARLM-inherited assumptions that may fail in MDLMs. Other choices, including data formatting, special-token semantics, stopping rules, and batching strategies, may require similarly MDLM-specific design.

Our analysis also suggests that [EOS] overflow is not primarily a model capacity failure. Instead, it emerges from the interaction between a learned length prior, bidirectional context, and standard large-block sampling rules. VoidPadding mitigates the padding–termination mismatch, but the design of effective MDLM samplers remains open. Future samplers may need to jointly consider token confidence, commitment timing, canvas allocation, and stopping decisions under bidirectional denoising.

References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. 2024. Smollm-blazingly fast and remarkably powerful. *Hugging Face Blog*, 16.

Hasan Amin, Yuan Gao, Yaser Souri, Subhojit Som, Ming Yin, Rajiv Khanna, and Xia Song. 2026. Consistent diffusion language models. *arXiv preprint arXiv:2605.00161*.

Marianne Arriola, Aaron Gokaslan, Justin Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sahoo, and Volodymyr Kuleshov. 2025. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, volume 2025, pages 50726–50753.

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021b. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas S Nolte, and Brian Karrer. 2026. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *Advances in Neural Information Processing Systems*, 38:55981–56007.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, and 1 others. 2026. Llada2. 1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, and 1 others. 2025. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Tianqi Chen, Shujian Zhang, and Mingyuan Zhou. 2025. Dlm-one: Diffusion language models for one-step sequence generation. *arXiv preprint arXiv:2506.00290*.

Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and 1 others. 2025. Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation. *arXiv preprint arXiv:2510.06303*.

Zicong Cheng, Ruixuan Jia, Jia Li, Guo-Wei Yang, Meng-Hao Guo, and Shi-Min Hu. 2026. Improving variable-length generation in diffusion language models via length regularization. *arXiv preprint arXiv:2602.07546*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, and 1 others. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Duc Anh Do, Luu Anh Tuan, Wray Buntine, and 1 others. 2025. Discrete diffusion language model for efficient text summarization. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6278–6290.

Hongcan Guo, Qinyu Zhao, Yian Zhao, Shen Nie, Rui Zhu, Qiushan Guo, Feng Wang, Tao Yang, Hengshuang Zhao, Guoqiang Wei, and 1 others. 2026. Continuous latent diffusion language model. *arXiv preprint arXiv:2605.06548*.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. 2023. Ssd-lm: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11575–11596.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Michael Hersche, Samuel Moor-Smith, Thomas Hofmann, and Abbas Rahimi. 2025. Soft-masked diffusion language models. *arXiv preprint arXiv:2510.17206*.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. 2021. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in neural information processing systems*, 34:12454–12465.

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

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. 2026. Elf: Embedded language flows. *arXiv preprint arXiv:2605.10938*.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S Abdelfattah, Jae-sun Seo, Zhiru Zhang, and Udit Gupta. 2025. Flashdlm: Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. 2026. Accelerating diffusion llms via adaptive parallel decoding. *Advances in neural information processing systems*, 38:52870–52888.

Jaehyeong Jo and Sung Ju Hwang. 2026. Continuous diffusion model for language modeling. *Advances in Neural Information Processing Systems*, 38:97394–97430.

Bumjun Kim, Dongjae Jeon, Dueun Kim, Wonje Jeung, and Albert No. 2025a. Rainbow padding: Mitigating early termination in instruction-tuned diffusion llms. *arXiv preprint arXiv:2510.03680*.

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. 2025b. Any-order flexible length masked diffusion. *arXiv preprint arXiv:2509.01025*.

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

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. 2025a. Beyond fixed: Training-free variable-length denoising for diffusion large language models. *arXiv preprint arXiv:2508.00819*.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Soroush Vosoughi, and Shiwei Liu. 2025b. Diffusion language models know the answer before decoding. *arXiv preprint arXiv:2508.19982*.

Aiwei Liu, Minghua He, Shaoxun Zeng, Sijun Zhang, Linhao Zhang, Chuhan Wu, Wei Jia, Yuan Liu, Xiao Zhou, and Jie Zhou. 2025. Wedlm: Reconciling diffusion language models with standard causal attention for fast inference. *arXiv preprint arXiv:2512.22737*.

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

Lizhuo Luo, Shenggui Li, Yonggang Wen, and Tianwei Zhang. 2026. Dsb: Dynamic sliding block scheduling for diffusion llms. *arXiv preprint arXiv:2602.05992*.

Amin Karimi Monsefi, Nikhil Bhendawade, Manuel Rafael Ciosici, Dominic Culver, Yizhe Zhang, and Irina Belousova. 2025. Fs-dfm: Fast and accurate long text generation with few-step diffusion language models. *arXiv preprint arXiv:2509.20624*.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2026. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and 1 others. 2018. Improving language understanding by generative pre-training.

Vittorio Rossi, Giacomo Cirò, Davide Beltrame, Luca Gandolfi, Paul Röttger, and Dirk Hovy. 2026. Diffusion language models are natively length-aware. *arXiv preprint arXiv:2603.06123*.

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

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin Chiu, and Volodymyr Kuleshov. 2025. The diffusion duality. *arXiv preprint arXiv:2506.10892*.

Junzhe Shen, Jieru Zhao, Ziwei He, and Zhouhan Lin. 2026. Codar: Continuous diffusion language models are more powerful than you think. *arXiv preprint arXiv:2603.02547*.

Yuerong Song, Xiaoran Liu, Ruixiao Li, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2026. Sparse-dllm: Accelerating diffusion llms with dynamic cache eviction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 33038–33046.

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

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. 2025. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*.

Linye Wei, Wenjue Chen, Pingzhi Tang, Xiaotian Guo, Le Ye, Runsheng Wang, and Meng Li. 2025a. Orchestrating dual-boundaries: An arithmetic intensity inspired acceleration framework for diffusion language models. *arXiv preprint arXiv:2511.21759*.

Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Puyu Zeng, Yuxuan Wang, Biqing Qi, Dongrui Liu, and Linfeng Zhang. 2025b. Accelerating diffusion large language models with slowfast sampling: The three golden principles. *arXiv preprint arXiv:2506.10848*.

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

Tianyi Wu, Xiaoxi Sun, Yanhua Jiao, Yulin Li, Yixin Chen, Yun-Hao Cao, Yi-Qi Hu, and Zhuotao Tian. 2026a. Dynamic-dllm: Dynamic cache-budget and adaptive parallel decoding for training-free acceleration of diffusion llm. In *The Fourteenth International Conference on Learning Representations*.

Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and 1 others. 2026b. Dreamon: Diffusion language models for code infilling beyond fixed-size canvas. *arXiv preprint arXiv:2602.01326*.

Jingyi Yang, Yuxian Jiang, and Jing Shao. 2026. ρ-EOS: Training-free bidirectional variable-length control for masked diffusion llms. *arXiv preprint arXiv:2601.22527*.

Yicun Yang, Cong Wang, Shaobo Wang, Zichen Wen, Biqing Qi, Hanlin Xu, and Linfeng Zhang. 2025. Diffusion llm with native variable generation lengths: Let [eos] lead the way. *arXiv preprint arXiv:2510.24605*.

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

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and 1 others. 2025. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*.

A Training Configurations

Dataset. We use the same 0.5M-example instruction-tuning dataset as the official RainbowPadding (Kim et al., 2025a) implementation. The data pool combines Tulu3 (Lambert et al., 2024) and SmolLM2 (Allal et al., 2024). After removing examples longer than 4096 tokens and multi-turn dialogues, we randomly sample 0.5M examples and use this identical data for all models in our experiments.

Shared training setup. Following the official RainbowPadding training setup, we train LLaDA-8B-Instruct (Nie et al., 2026) and Dream-7B-Instruct (Ye et al., 2025) for the main experiments, and use LLaDA-8B-Base and Dream-7B-Base for additional diagnostic runs. All models are fine-tuned with LoRA (Hu et al., 2022) of rank 32 applied to all linear layers, without bias terms. We use AdamW with learning rate 5×10^{-5} and global batch size 48. LLaDA models are trained on two A800-80GB GPUs with mini-batch size 6 per GPU and 4 gradient-accumulation steps; Dream models are trained on three A800-80GB GPUs with mini-batch size 4 per GPU and 4 gradient-accumulation steps. For both models, one epoch takes approximately 32–34 A800 GPU hours.

Training budgets. RainbowPadding follows its official budget: one epoch for LLaDA-8B-Instruct and Dream-7B-Instruct, corresponding to 10,417 optimizer updates, and three epochs for LLaDA-8B-Base and Dream-7B-Base. All VoidPadding checkpoints use 67% of the corresponding RainbowPadding training budget: 7000 optimizer updates for LLaDA-8B-Instruct and Dream-7B-Instruct, and two epochs for LLaDA-8B-Base and Dream-7B-Base. The [EOS]-padding diagnostic models are trained for two epochs on LLaDA-8B-Base and Dream-7B-Base.

67% training-budget comparison. Figures 13 and 14 compare Early RainbowPadding and VoidPadding under the same training budget. Specifically, Early RainbowPadding uses 7000 optimizer updates, matching the VoidPadding training budget and corresponding to 67% of the official RainbowPadding budget for both LLaDA-8B-Instruct and Dream-7B-Instruct. All other settings are kept identical, including the training data, LoRA configuration, optimizer, batch size, fixed generation length, and the absence of VoidExpansion, matching the fixed-length generation robustness experiments in Table 1. Under this matched setup, VoidPadding achieves better accuracy than Early RainbowPadding on both models.

B Proofs for the [EOS] Overflow Analysis

This appendix proves Theorem 1 and Theorem 2. We use the notation and assumptions introduced in Section 3: e denotes [EOS], $F(m) = \Pr(\ell \le m)$, and $r = \min\{i : F(i-1) > 1/2\}$. For position i, $\bar{Y}_i$ is the training-time padded target token, p_i is the training-time token distribution with $p_i(e) = \Pr(\bar{Y}_i = e)$, q_i is the inference-time model distribution at a masked canvas position, and $q_i(e)$ is its [EOS] probability. The argmax proposal is $\hat{y}_i = \arg\max_v q_i(v)$, and Y_i is the committed token. We write $S_e = \{Y_r = \cdots = Y_L = e\}$ for the inference-time committed [EOS] suffix event and $\bar{S}_e = \{\bar{Y}_r = \cdots = \bar{Y}_L = e\}$ for the corresponding training-time [EOS] padding suffix event. A selected masked position is filled with its argmax proposal. We also use the elementary fact that if $q_i(e) > 1/2$, then no other token can have probability at least $q_i(e)$. Thus e is the unique argmax proposal at position i, and if position i is selected, then $Y_i = e$.

Proof of Theorem 1. Consider any selected position $j \in \{r, \ldots, B\}$. Since $j \ge r$ and F is non-decreasing,

$$F(j-1) \ge F(r-1) > 1/2.$$

By the Stage 1 assumption in Section 3, $q_j(e) = p_j(e) = F(j-1)$. Hence $q_j(e) > 1/2$, so e is the unique argmax proposal at position j. Because j is selected, the committed token is

$$Y_j = e.$$

The argument applies to every selected $j \in \{r, \ldots, B\}$, proving the claim. □

Proof of Theorem 2. Assume S_e has occurred, and consider any selected position $i < r$ satisfying $2F(i-1) > F(r-1)$. By the Stage 2 assumption,

$$q_i(e \mid S_e) = \Pr(\bar{Y}_i = e \mid \bar{S}_e).$$

For this theorem, we use the Stage 2 conditional assumption from Section 3, together with its length-signal form for the suffix event $\bar{S}_e$: for $i < r$,

$$\begin{aligned} q_i(e \mid S_e) &= p_i(e \mid \bar{S}_e) \\ &= \frac{F(i-1)}{F(r-1)}. \end{aligned}$$

The condition $2F(i-1) > F(r-1)$ is exactly

$$\frac{F(i-1)}{F(r-1)} > 1/2.$$

Thus $q_i(e \mid S_e) > 1/2$, so e is the unique argmax proposal at position i. Since i is selected, the decoder commits

$$Y_i = e.$$

This proves the claim. □

C Dream Decoding Wrapper

Algorithm 2 summarizes the blockwise commit wrapper used for Dream evaluations.

Dream still predicts all masked positions globally at each denoising step, but the wrapper restricts the TopK commit candidates to the masked positions inside the current block.

D Fixed-Length Block-Size Sweeps on Instruction-Tuned Models

This section reports the full fixed-length block-size sweeps behind Table 1. The following LLaDA-8B-Instruct and Dream-7B-Instruct sweeps follow the

Algorithm 2 Blockwise Commit Wrapper for Dream Decoding

Require: Prompt $x_{1:p}$, generation length L, block size B, total denoising steps T

Initialize $x_{p+1:p+L} \leftarrow [M]$
Split the generation span into $K = L/B$ blocks $\{\mathcal{B}_k\}_{k=1}^{K}$; set $S = T/K$
for $k = 1$ to K **do**
  for $s = 1$ to S **do**
    Run Dream on the full sequence x
    Obtain predictions $\hat{x}_i$ and negative entropy confidence scores c_i for masked positions
    $\mathcal{E}_k \leftarrow \{i \in \mathcal{B}_k \mid x_i = [M]\}$
    if $\mathcal{E}_k = \emptyset$ **then**
      break
    end if
    $C_s \leftarrow \mathrm{TopK}_{i \in \mathcal{E}_k}(c_i)$
    $x_i \leftarrow \hat{x}_i, \quad \forall i \in C_s$
    Keep all positions $j \notin C_s$ unchanged
  end for
end for
return $x_{1:p+L}$

same setup: all runs use a fixed generation length of 512 without VoidExpansion; we vary the block size over $B \in \{32, 64, 128, 256, 512\}$ and report the mean, best, and worst scores across the sweep. Mean is the arithmetic average over the five block sizes, while Best and Worst are the maximum and minimum scores over the five block sizes, respectively. Reasoning benchmarks report accuracy, and code-generation benchmarks report pass@1.

D.1 LLaDA-8B-Instruct Fixed-512 Block-Size Sweep

For LLaDA-8B-Instruct and RainbowPadding experiments in the tables, rows with the `[EOS]` term. label apply `[EOS]`Termination, while rows without this label follow the standard VanillaStopping rule. For VoidPadding, the default row includes `[EOS]`Termination, and the w/o `[EOS]` term. row reports the ablation without this termination rule.

Table 7 reports GSM8K accuracy, Table 8 reports MATH500 accuracy, Table 9 reports HumanEval pass@1, and Table 10 reports MBPP pass@1.

D.2 Dream-7B-Instruct Fixed-512 Block-Length Sweep

Figure 11 reports the corresponding Dream-7B-Instruct fixed-length block-length sweep under our block-wise Dream decoding wrapper.

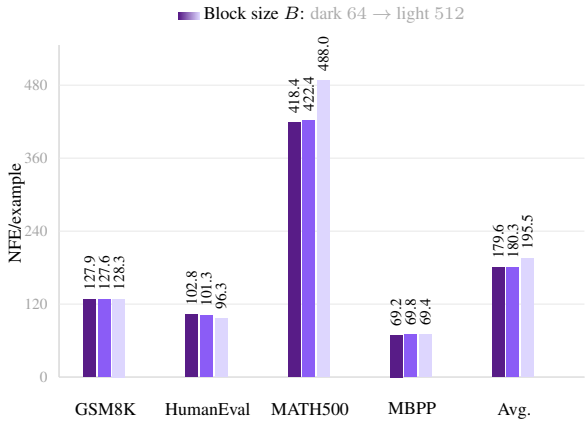

Figure 5: LLaDA-8B-Instruct VoidPadding NFE/example for $B \in \{64, 128, 512\}$. Avg. is arithmetic over four benchmarks.

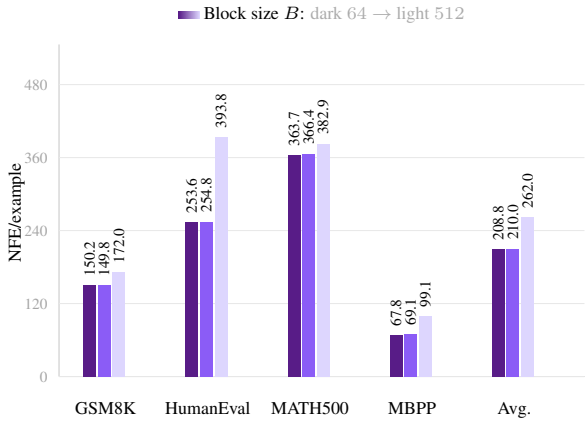

Figure 6: Dream-7B-Instruct VoidPadding NFE/example for $B \in \{64, 128, 512\}$. Avg. is arithmetic over four benchmarks.

E `[EOS]`Termination Efficiency

This appendix reports per-block fixed-length NFE/example values for the Table 1 settings over $B \in \{64, 128, 512\}$. Figure 5 shows the LLaDA-8B-Instruct per-block NFE values, and Figure 6 shows the Dream-7B-Instruct per-block NFE values behind Table 2. Avg. entries are arithmetic averages over benchmarks.

F Fixed-Length Block-Size Sweeps on Base Models

The sweep configuration matches Section D: all runs use a fixed generation length of 512 without VoidExpansion, and we vary the block size over $B \in \{32, 64, 128, 256, 512\}$. The only difference is that these diagnostics start from base models and compare `[EOS]` Padding, RainbowPadding,

and VoidPadding variants.

F.1 LLaDA-8B-Base Fixed-512 Block-Size Sweep

This subsection reports the LLaDA-8B-Base fixed-length block-size sweep. For [EOS] Padding and RainbowPadding, rows with the [EOS] term. label apply [EOS]Termination, while rows without this label follow the standard VanillaStopping rule. For VoidPadding, the default row includes [EOS]Termination, and the w/o [EOS] term. row reports the ablation without this termination rule. GSM8K reports flexible exact match, MATH500 reports accuracy, and HumanEval and MBPP report pass@1. Table 11 reports GSM8K, Table 12 reports MATH500, Table 13 reports HumanEval, and Table 14 reports MBPP.

F.2 Dream-7B-Base Fixed-512 Block-Length Sweep

Figure 12 reports the corresponding Dream-7B-Base fixed-length block-length sweep under the same block-wise Dream decoding wrapper.

G Base-Model VoidExpansion Effectiveness and Hyperparameter Robustness

Unless otherwise stated, the base-model VoidExpansion sweeps use initial length $L_0 = 64$, tail-window size $w = 16$, block size $B = 32$, and maximum length $L_{\max} = 512$. The fixed-64 baseline denotes decoding from a length-64 canvas with VoidExpansion disabled.

G.1 LLaDA-8B-Base

The common LLaDA-8B-Base VoidExpansion setting uses $L_0 = 64$, $w = 16$, $B = 32$, $L_{\max} = 512$, and $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.09, -0.45)$. Figure 23 reports the corresponding threshold sweep: it keeps L_0, w, B, and $L_{\max}$ fixed and varies τ_{nonvoid} and τ_{gap}. Each panel compares against the fixed-64 baseline and reports the benchmark mean as the arithmetic average over GSM8K, MATH500, HumanEval, and MBPP. Figure 15 reports the initial-length ablation: it keeps w, B, $L_{\max}$, and the threshold pair fixed and varies L_0 over $64, 128, 192$, comparing fixed-length decoding with VoidExpansion at the same lengths. Across both sweeps, the four-task mean changes by only about two points, and VoidExpansion remains consistently stronger than the corresponding fixed-length baselines.

G.2 Dream-7B-Base

The Dream-7B-Base threshold sweep is shown in Figure 25. Its selected setting is $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.12, 0.65)$, and the red ticks again mark the corresponding fixed-64 baseline, i.e., $L_0 = 64$ with VoidExpansion disabled. As with LLaDA-8B-Base, the threshold sweep shows only modest accuracy variation across settings, while VoidExpansion remains consistently stronger than the fixed-64 baseline.

H VoidExpansion Hyperparameter Robustness

This appendix reports the complete VoidExpansion sweeps behind the hyperparameter summary in Table 6. We first present VoidPadding-finetuned LLaDA-8B-Instruct sweeps over initial length (L_0), tail-window size w, block size (B), and threshold pairs. We then report the corresponding VoidPadding-finetuned Dream-7B-Instruct sweeps over threshold pairs.

H.1 LLaDA-8B-Instruct

This subsection provides the full LLaDA-8B-Instruct results summarized in Section 6. All runs use the VoidPadding-finetuned LLaDA-8B-Instruct model with maximum generation length $L_{\max} = 512$. The default expansion setting, matching Table 3, uses initial length $L_0 = 64$, tail window $w = 16$, block size $B = 32$, and $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.09, -0.45)$.

Initial-length robustness. Figure 16 varies the initial length $L_0 \in \{64, 96, 128, 160\}$ while keeping the other expansion hyperparameters fixed, and compares no-expansion fixed-length decoding against VoidExpansion at the same initial lengths. Figure 19 reports the corresponding average decode NFE.

Window-size robustness. Figure 9 varies the tail-window size $w \in \{8, 16, 24, 32\}$ while keeping the other settings fixed. Figure 10 reports the corresponding decode NFE with the same benchmark grouping. These figures provide the full data for the tail-window row in Table 6.

Block-size robustness. Figure 18 varies the block size $B \in \{16, 32, 64\}$ while keeping the other settings fixed. Figure 20 reports the corresponding decode NFE with the same benchmark grouping. These figures provide the full data for the block-size row in Table 6.

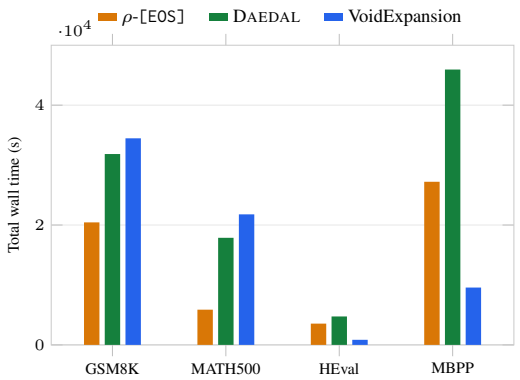

Figure 7: Total wall time on LLaDA-8B-Instruct under the init-64 setting. Bars compare ρ-[EOS], DAEDAL, and VoidExpansion.

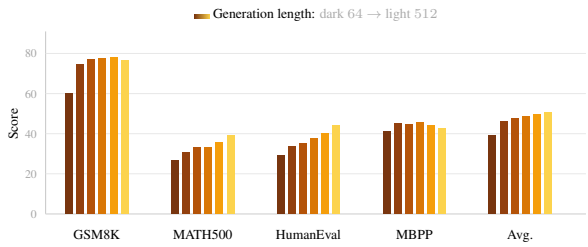

Figure 8: Fixed generation length sensitivity for LLaDA-8B-Instruct with block size $B = 32$ and generation length $L \in \{64, 128, 160, 192, 256, 512\}$. Darker orange denotes shorter L; Avg. is the arithmetic average over benchmarks.

Threshold robustness. Figure 21 varies τ_{nonvoid} and τ_{gap} around the selected setting from Table 3, $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.09, -0.45)$, while keeping L_0, w, B, and $L_{\max}$ fixed. Figure 22 reports the corresponding NFE results. These two figures provide the detailed threshold row behind Table 6.

H.2 Dream-7B-Instruct

Figure 24 reports the VoidPadding-finetuned Dream-7B-Instruct + VoidExpansion threshold sweep under our block-wise Dream decoding wrapper. The selected setting from Table 4, $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.15, 0.55)$, is marked in blue. The red ticks show the fixed-64 baseline, i.e., decoding from a length-64 canvas with VoidExpansion disabled, using the same VoidPadding-finetuned checkpoint. The scores remain stable across the threshold grid, showing that Dream-7B-Instruct is also robust to the VoidExpansion threshold choices.

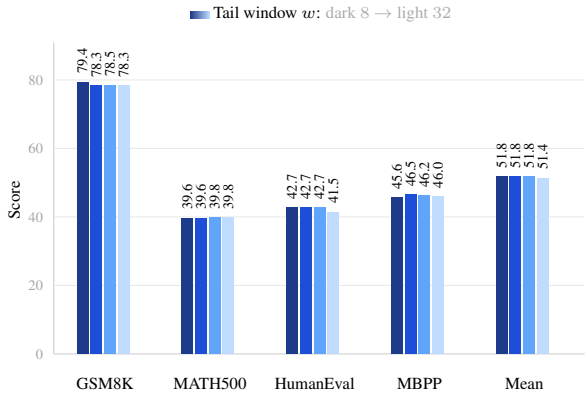

Figure 9: Window-size accuracy statistics for VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion. Bars are grouped by benchmark; within each group, the four bars vary the tail window $w \in \{8, 16, 24, 32\}$. Darker blue denotes smaller w. Labels give the exact score. GSM8K and MATH500 report accuracy, HumanEval and MBPP report pass@1, and Mean is the arithmetic average over the four benchmarks.

I Responsible Research and Artifact Use

Artifacts, licenses, and intended use. Our experiments use previously released research artifacts: LLaDA-8B-Instruct, LLaDA-8B-Base, Dream-7B-Instruct, Dream-7B-Base, the official Rainbow-Padding setup (Kim et al., 2025a), the Tulu3 and SmolLM2 instruction-tuning resources, and the GSM8K, MATH500, HumanEval, and MBPP evaluation benchmarks. Our training code builds on the RainbowPadding training implementation (Kim et al., 2025a), and our evaluation code builds on the Fast-dLLM evaluation framework (Wu et al., 2025). We use the official ρ-[EOS] (Yang et al., 2026) and DAEDAL (Li et al., 2025a) implementations as inference-only baselines. We cite the creators of these artifacts in Sections 5 and 3 and Appendix A. We used the artifacts only for research fine-tuning, controlled diagnostic analysis, and benchmark evaluation, which is consistent with their documented research use. We followed the publicly stated licenses and terms of the source artifacts, including the subset-specific terms of the Tulu3 mixture, the published terms of the SmolLM2 resources, MIT-licensed LLaDA, GSM8K, HumanEval, and RainbowPadding artifacts, Apache-2.0-licensed Dream, Fast-dLLM, ρ-[EOS], and DAEDAL artifacts, and CC-BY-4.0-licensed MBPP. For artifacts whose hosting pages do not specify a standalone license beyond their source benchmark or repository terms, we used them only for non-commercial research

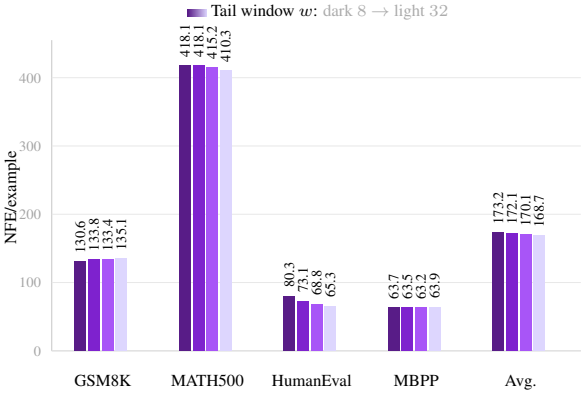

Figure 10: Window-size NFE statistics for VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion. Bars are grouped by benchmark; within each group, the four bars vary the tail window $w \in \{8, 16, 24, 32\}$. Darker purple denotes smaller w. Labels report average decode NFE. Avg. is the arithmetic average over the four benchmarks.

evaluation and cite the original creators.

Data documentation and privacy. Appendix A describes the instruction-tuning data sources, filtering rule, sample count, models, optimizer, batch sizes, and training budgets. Section 5 describes the evaluation benchmarks, which cover English mathematical reasoning and Python code generation. We did not collect new data from individuals and did not conduct new annotation or human-subject studies. Because the instruction-tuning data and benchmarks are previously released public artifacts, we rely on their original curation and documentation for consent, privacy, and content filtering procedures. We did not attempt to identify individuals in the data and removed only length- and format-based examples as described in Appendix A. Residual personally identifying, offensive, or otherwise sensitive content may still be inherited from the source artifacts, and downstream users should apply task-appropriate filtering before deployment.

Potential risks. VoidPadding and VoidExpansion are designed to improve the quality, robustness, and efficiency of masked diffusion language models. These benefits can also lower the cost of generating large amounts of text or code, which may increase the feasibility of spam, misinformation, low-quality content generation, or misuse of generated code. The method does not add safety filtering, factuality guarantees, or policy enforcement beyond the behavior of the underlying models. As with other language-model adaptation methods, outputs may be incorrect, biased, unsafe, or unsuitable for high-stakes use without additional validation and safeguards.

Evaluation and reporting details. Reported benchmark numbers are from experiments under the decoding and training configurations described in Sections 5 and 6 and Appendices A and H. We do not report confidence intervals or multi-seed error bars. GSM8K and MATH500 are reported with answer accuracy, while HumanEval and MBPP are reported with pass@1. Existing package or implementation choices are used with their documented default evaluation behavior unless a paper-specific setting is specified in the corresponding experiment description, such as block size, generation length, maximum length, tail-window size, and VoidExpansion thresholds.

Use of AI assistants. Codex with GPT-5 was used for coding support, language editing, and preparation of the responsible-research checklist. The authors reviewed and edited the resulting text, code, and checklist responses, and the reported claims and experimental results remain the authors' responsibility.

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
LLaDA-8B-Instruct	76.72	76.80	70.96	44.96	28.43	59.57	76.80	28.43
LLaDA-8B-Instruct + `[EOS]` term.	**77.71**	76.65	69.60	39.95	28.13	58.41	77.71	28.13
RainbowPadding	73.77	74.53	73.39	72.25	66.57	72.10	74.53	66.57
RainbowPadding + `[EOS]` term.	73.77	74.53	73.39	72.25	66.57	72.10	74.53	66.57
VoidPadding w/o `[EOS]` term.	57.32	57.16	57.24	55.95	65.66	58.67	65.66	55.95
VoidPadding w/o VOID ban	75.06	64.67	14.71	11.14	14.40	36.00	75.06	11.14
VoidPadding	77.41	**77.79**	**78.39**	**78.24**	**78.62**	**78.09**	**78.62**	**77.41**

Table 7: GSM8K block-size sweep for LLaDA-8B-Instruct; cells report accuracy (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
LLaDA-8B-Instruct	39.00	39.60	37.60	30.60	16.60	32.68	39.60	16.60
LLaDA-8B-Instruct + `[EOS]` term.	39.00	39.60	36.80	30.40	16.60	32.48	39.60	16.60
RainbowPadding	33.20	33.20	34.00	34.00	**27.80**	32.44	34.00	**27.80**
RainbowPadding + `[EOS]` term.	33.20	33.20	34.00	34.00	**27.80**	32.44	34.00	**27.80**
VoidPadding w/o `[EOS]` term.	**39.60**	**40.60**	**41.60**	**40.20**	27.60	**37.92**	**41.60**	27.60
VoidPadding w/o VOID ban	**39.60**	39.80	38.20	32.20	14.20	32.80	39.80	14.20
VoidPadding	**39.60**	**40.60**	**41.60**	40.00	27.60	37.88	**41.60**	27.60

Table 8: MATH500 block-size sweep for LLaDA-8B-Instruct; cells report accuracy (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
pass@1								
LLaDA-8B-Instruct	43.90	42.68	**43.90**	42.07	**44.51**	**43.41**	44.51	**42.07**
LLaDA-8B-Instruct + `[EOS]` term.	42.68	42.68	43.29	40.24	43.90	42.56	43.90	40.24
RainbowPadding	**45.12**	**43.90**	43.29	**43.29**	35.98	42.32	**45.12**	35.98
RainbowPadding + `[EOS]` term.	**45.12**	**43.90**	43.29	**43.29**	35.98	42.32	**45.12**	35.98
VoidPadding w/o `[EOS]` term.	19.51	20.73	22.56	23.78	20.73	21.46	23.78	19.51
VoidPadding w/o VOID ban	29.88	17.07	10.37	6.10	6.10	13.90	29.88	6.10
VoidPadding	42.07	41.46	42.07	42.07	41.46	41.83	42.07	41.46

Table 9: HumanEval block-size sweep for LLaDA-8B-Instruct, with pass@1 (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
pass@1								
LLaDA-8B-Instruct	42.81	41.48	39.12	33.98	37.68	39.01	42.81	33.98
LLaDA-8B-Instruct + `[EOS]` term.	43.12	42.09	39.73	35.22	38.71	39.77	43.12	35.22
RainbowPadding	42.51	42.71	43.02	43.12	42.51	42.77	43.12	42.51
RainbowPadding + `[EOS]` term.	42.40	42.61	43.02	43.12	42.51	42.73	43.12	42.40
VoidPadding w/o `[EOS]` term.	**44.87**	**44.66**	**44.87**	**44.87**	**44.97**	**44.85**	**44.97**	**44.66**
VoidPadding w/o VOID ban	41.27	25.36	16.94	17.66	17.56	23.76	41.27	16.94
VoidPadding	**44.87**	**44.66**	**44.87**	**44.87**	**44.97**	**44.85**	**44.97**	**44.66**

Table 10: MBPP block-size sweep for LLaDA-8B-Instruct, with pass@1 (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
LLaDA-8B-Base + `[EOS]` Padding	**77.41**	58.76	15.31	7.88	11.83	34.24	**77.41**	7.88
LLaDA-8B-Base + `[EOS]` Padding + `[EOS]` term.	**77.41**	58.76	15.31	7.88	11.83	34.24	**77.41**	7.88
RainbowPadding	76.27	76.12	**76.12**	**75.74**	**75.51**	**75.95**	76.27	**75.51**
RainbowPadding + `[EOS]` term.	75.51	76.12	75.66	75.59	**75.51**	75.68	76.12	**75.51**
VoidPadding `w/o [EOS] term.`	3.11	3.03	3.41	3.26	2.88	3.14	3.41	2.88
VoidPadding w/o VOID ban	75.44	67.32	46.85	42.76	43.14	55.10	75.44	42.76
VoidPadding	75.51	**76.57**	75.82	75.28	75.28	75.69	76.57	75.28

Table 11: GSM8K block-size sweep for LLaDA-8B-Base after padding LoRA SFT; cells report accuracy (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
LLaDA-8B-Base + `[EOS]` Padding	37.80	36.80	34.00	29.40	20.60	31.72	37.80	20.60
LLaDA-8B-Base + `[EOS]` Padding + `[EOS]` term.	37.80	36.80	34.00	29.40	20.60	31.72	37.80	20.60
RainbowPadding	34.00	33.40	34.80	35.00	34.80	34.40	35.00	33.40
RainbowPadding + `[EOS]` term.	35.80	35.80	36.00	36.00	**36.60**	36.04	36.60	**35.80**
VoidPadding `w/o [EOS] term.`	**38.40**	**40.00**	**39.20**	**37.40**	34.00	**37.80**	**40.00**	34.00
VoidPadding w/o VOID ban	38.20	39.20	31.00	23.00	23.00	30.88	39.20	23.00
VoidPadding	**38.40**	**40.00**	**39.20**	**37.40**	34.00	**37.80**	**40.00**	34.00

Table 12: MATH500 block-size sweep for LLaDA-8B-Base after padding LoRA SFT; cells report accuracy (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
pass@1								
LLaDA-8B-Base + `[EOS]` Padding	37.20	26.83	15.85	13.41	12.20	21.10	37.20	12.20
LLaDA-8B-Base + `[EOS]` Padding + `[EOS]` term.	37.20	26.83	15.85	13.41	12.20	21.10	37.20	12.20
RainbowPadding	**38.41**	**39.02**	**38.41**	**38.41**	**38.41**	**38.53**	**39.02**	**38.41**
RainbowPadding + `[EOS]` term.	37.80	38.41	**38.41**	**38.41**	**38.41**	38.29	38.41	37.80
VoidPadding `w/o [EOS] term.`	14.63	17.07	22.56	23.17	8.54	17.19	23.17	8.54
VoidPadding w/o VOID ban	35.37	18.29	12.80	14.63	15.24	19.27	35.37	12.80
VoidPadding	37.80	37.20	37.80	**38.41**	**38.41**	37.92	38.41	37.20

Table 13: HumanEval block-size sweep for LLaDA-8B-Base after padding LoRA SFT, with pass@1 (%).

Method	Block size B					Summary		
	32	64	128	256	512	Mean	Best	Worst
pass@1								
LLaDA-8B-Base + `[EOS]` Padding	43.22	32.55	23.72	23.82	22.18	29.10	43.22	22.18
LLaDA-8B-Base + `[EOS]` Padding + `[EOS]` term.	43.22	32.55	23.72	23.82	22.18	29.10	43.22	22.18
RainbowPadding	43.33	43.53	43.74	43.53	43.94	43.61	43.94	43.33
RainbowPadding + `[EOS]` term.	43.33	43.53	43.74	43.53	43.94	43.61	43.94	43.33
VoidPadding `w/o [EOS] term.`	**44.35**	**44.87**	**44.66**	**44.76**	**44.76**	**44.68**	**44.87**	**44.35**
VoidPadding w/o VOID ban	43.63	29.98	21.56	20.64	18.38	26.84	43.63	18.38
VoidPadding	**44.35**	**44.87**	**44.66**	**44.76**	**44.76**	**44.68**	**44.87**	**44.35**

Table 14: MBPP block-size sweep for LLaDA-8B-Base after padding LoRA SFT, with pass@1 (%).

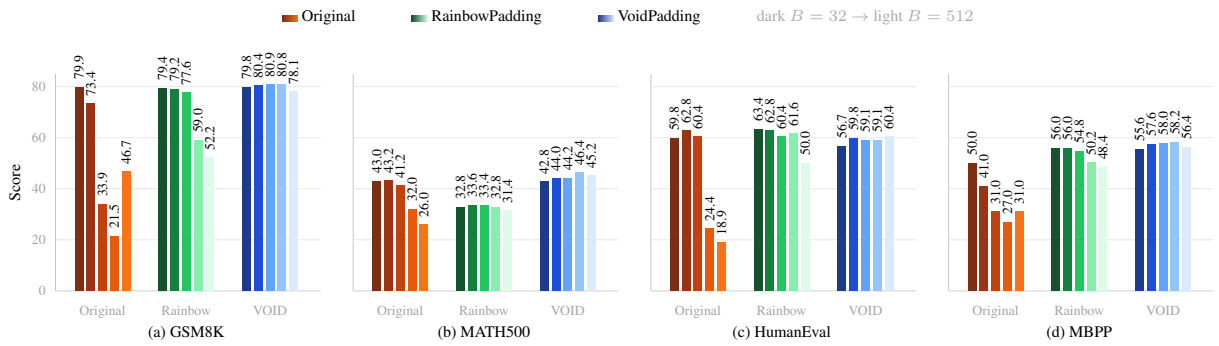

Figure 11: Dream-7B-Instruct fixed-512 block-length sweep comparing the original model with RainbowPadding and VoidPadding checkpoints. Bars vary $B \in \{32, 64, 128, 256, 512\}$, with darker colors denoting smaller blocks. Reasoning tasks report accuracy, and code tasks report pass@1.

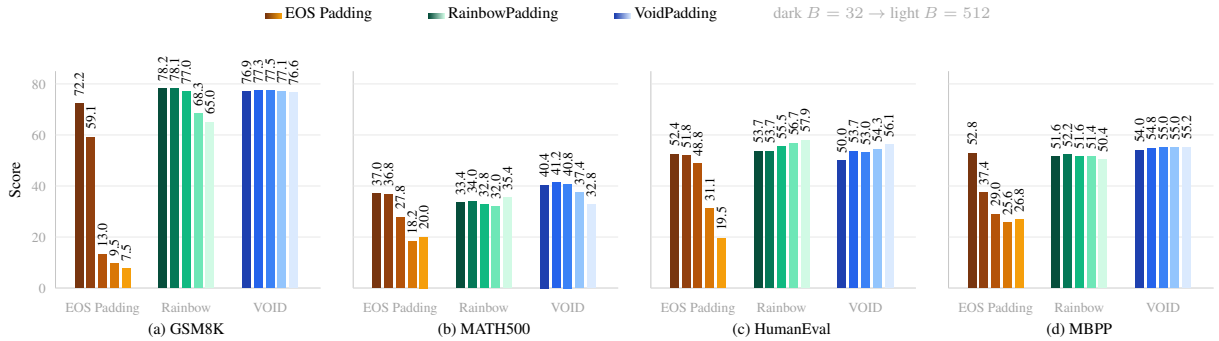

Figure 12: Dream-7B-Base fixed-512 block-length sweep comparing EOS Padding with RainbowPadding and VoidPadding LoRA SFT checkpoints. Bars vary $B \in \{32, 64, 128, 256, 512\}$, with darker colors denoting smaller blocks. Reasoning tasks report accuracy, and code tasks report pass@1.

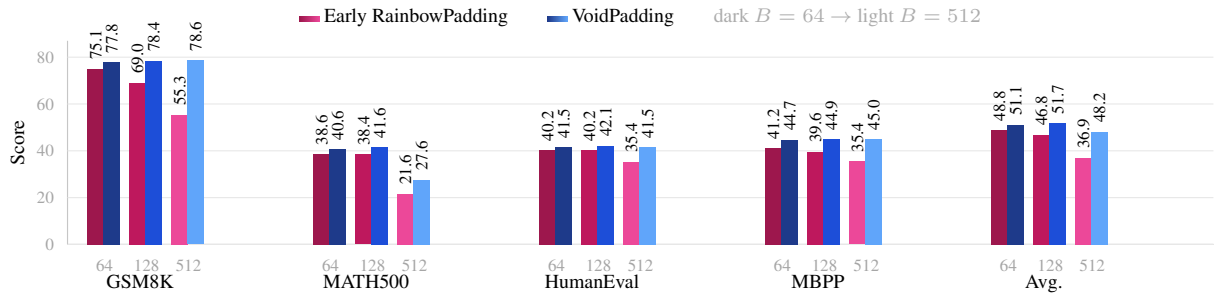

Figure 13: Same-step training-budget comparison on LLaDA-8B-Instruct. Bars compare Early RainbowPadding (67% training steps) with VoidPadding at $B \in \{64, 128, 512\}$; darker colors denote smaller B. Avg. is the arithmetic average over benchmarks.

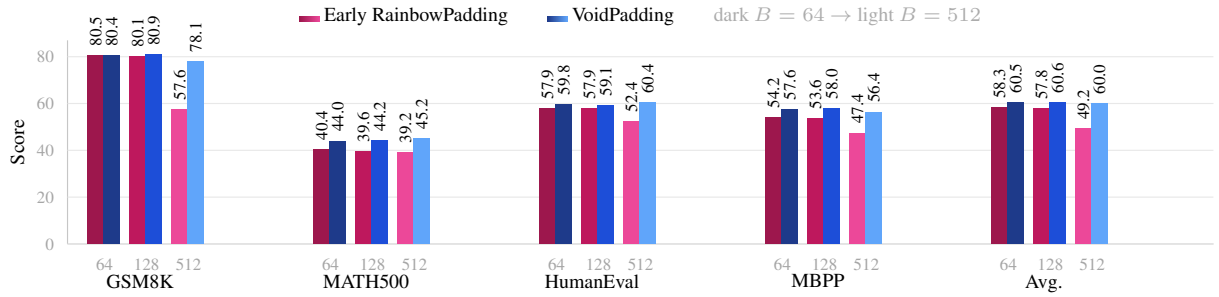

Figure 14: Same-step training-budget comparison on Dream-7B-Instruct. Bars compare Early RainbowPadding (67% training steps) with VoidPadding at $B \in \{64, 128, 512\}$; darker colors denote smaller B. Avg. is the arithmetic average over benchmarks.

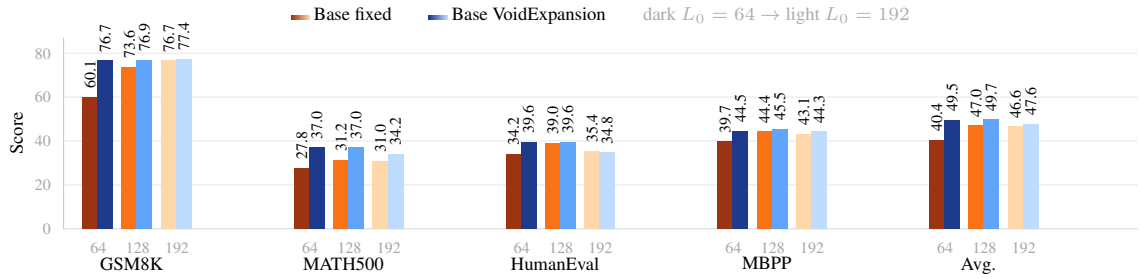

Figure 15: Initial-length ablation for VoidPadding-finetuned LLaDA-8B-Base + VoidExpansion. Bars compare fixed-length decoding with VoidExpansion at $L_0 \in \{64, 128, 192\}$; darker colors denote smaller L_0.

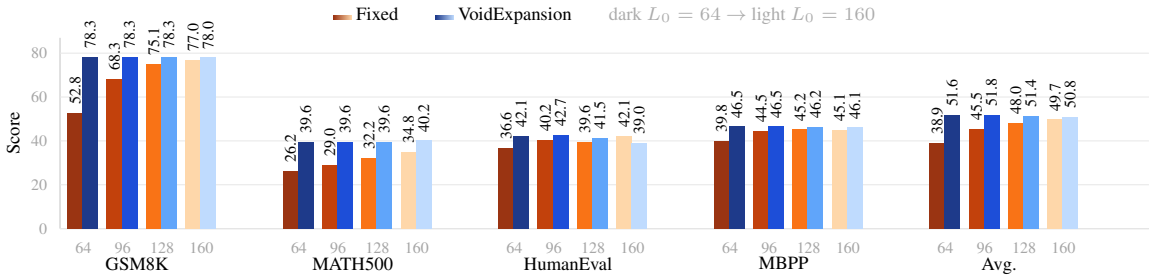

Figure 16: Initial-length ablation for VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion. Bars compare fixed-length decoding with VoidExpansion at $L_0 \in \{64, 96, 128, 160\}$; darker colors denote smaller L_0.

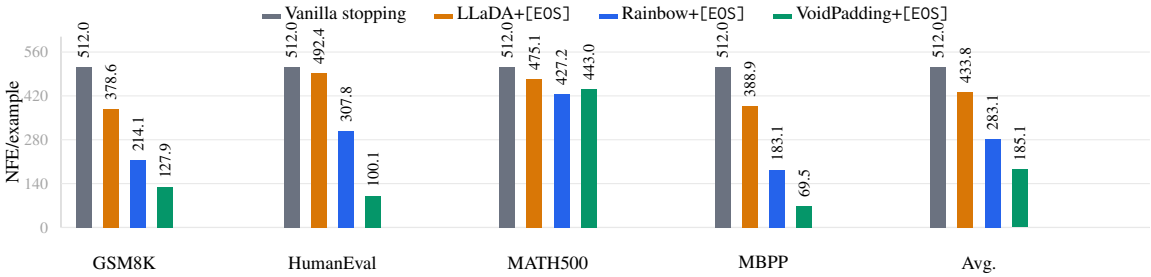

Figure 17: Fixed-length LLaDA-8B-Instruct NFE with $L_{\max} = 512$, averaged over $B \in \{64, 128, 512\}$. Bars compare vanilla stopping with [EOS]-termination variants; Avg. is the arithmetic average over benchmarks.

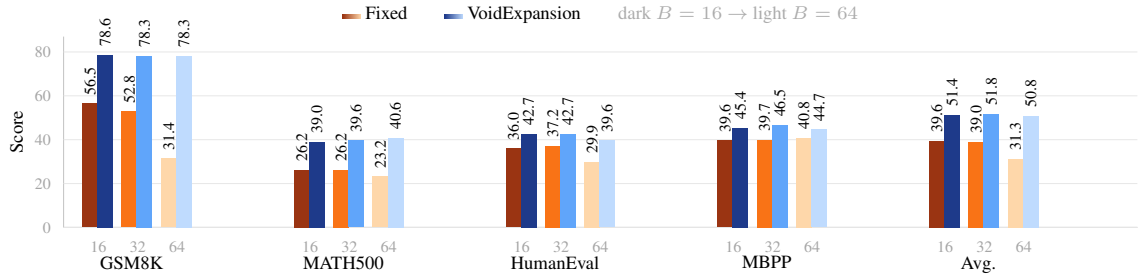

Figure 18: Block-size accuracy statistics for VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion. Bars compare fixed-length decoding with VoidExpansion at $B \in \{16, 32, 64\}$; darker colors denote smaller B.

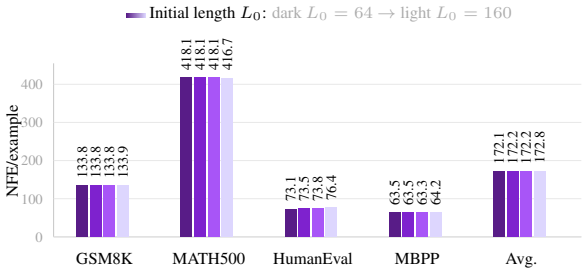

Figure 19: Initial-length NFE ablation results for VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion over $L_0 \in \{64, 96, 128, 160\}$. Avg. is the benchmark mean.

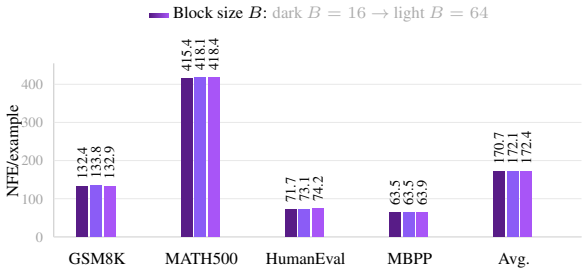

Figure 20: Block-size NFE ablation results for VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion over $B \in \{16, 32, 64\}$. Avg. is the benchmark mean.

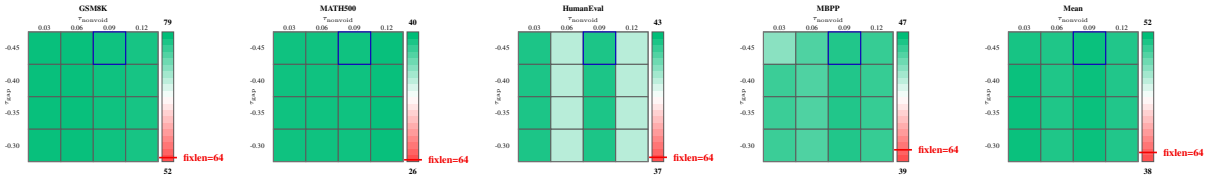

Figure 21: VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion threshold sweep. The sweep uses $\tau_{\text{nonvoid}} \in \{0.03, 0.06, 0.09, 0.12\}$ and $\tau_{\text{gap}} \in \{-0.45, -0.40, -0.35, -0.30\}$; the blue box marks the selected setting from Table 3, $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.09, -0.45)$. Cell color reports accuracy for reasoning tasks and pass@1 for code tasks. Red indicates lower scores and green indicates higher scores; the fixlen=64 markers are all red, showing that no-expansion fixed-64 is consistently low.

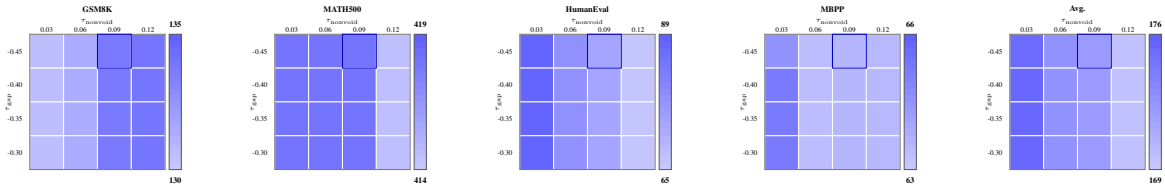

Figure 22: VoidPadding-finetuned LLaDA-8B-Instruct + VoidExpansion threshold-sweep average NFE. The sweep uses $\tau_{\text{nonvoid}} \in \{0.03, 0.06, 0.09, 0.12\}$ and $\tau_{\text{gap}} \in \{-0.45, -0.40, -0.35, -0.30\}$; the blue box marks the selected setting from Table 3, $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.09, -0.45)$. Cell color reports benchmark-level average decode NFE for the first four panels, and Avg. is the arithmetic average over the four benchmarks.

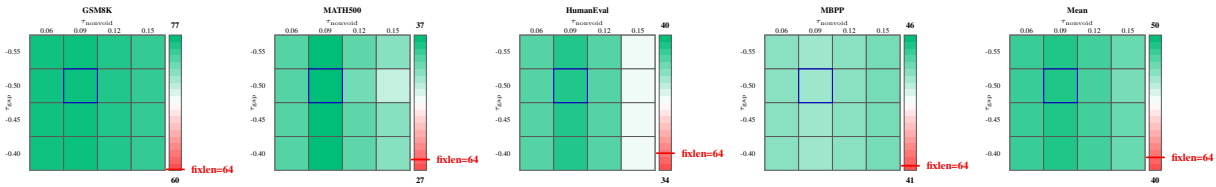

Figure 23: VoidPadding-finetuned LLaDA-8B-Base + VoidExpansion threshold sweep. The sweep uses $\tau_{\text{nonvoid}} \in \{0.06, 0.09, 0.12, 0.15\}$ and $\tau_{\text{gap}} \in \{-0.55, -0.50, -0.45, -0.40\}$; the blue box marks the selected setting, $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.09, -0.45)$. Cell color reports accuracy for reasoning tasks and pass@1 for code tasks. Red indicates lower scores and green indicates higher scores; the fixlen=64 markers are all red, showing that no-expansion fixed-64 is consistently low.

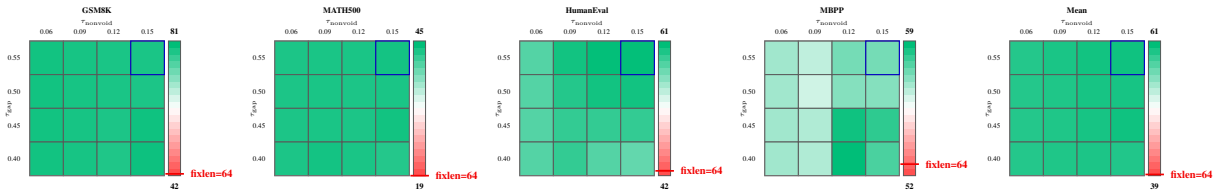

Figure 24: VoidPadding-finetuned Dream-7B-Instruct + VoidExpansion threshold sweep. The sweep uses $\tau_{\text{nonvoid}} \in \{0.06, 0.09, 0.12, 0.15\}$ and $\tau_{\text{gap}} \in \{0.55, 0.50, 0.45, 0.40\}$; the blue box marks the selected setting from Table 4, $(\tau_{\text{nonvoid}}, \tau_{\text{gap}}) = (0.15, 0.55)$. Cell color reports accuracy for reasoning tasks and pass@1 for code tasks. Red indicates lower scores and green indicates higher scores; the fixlen=64 markers are all red, showing that no-expansion fixed-64 is consistently low.

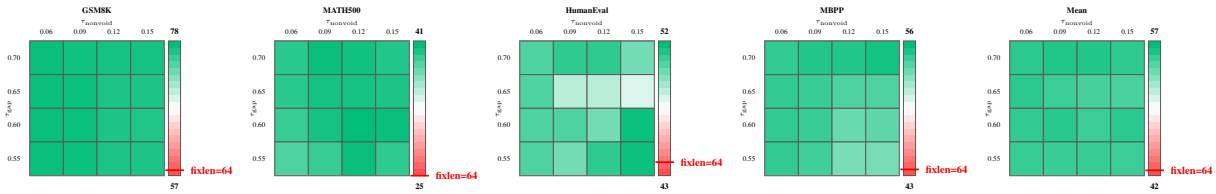

Figure 25: VoidPadding-finetuned Dream-7B-Base + VoidExpansion threshold sweep. The sweep uses $\tau_{\text{nonvoid}} \in \{0.06, 0.09, 0.12, 0.15\}$ and $\tau_{\text{gap}} \in \{0.70, 0.65, 0.60, 0.55\}$. Cell color reports accuracy for reasoning tasks and pass@1 for code tasks. Red indicates lower scores and green indicates higher scores; the fixlen=64 markers are all red, showing that no-expansion fixed-64 is consistently low.