STREAMING-DLLM: ACCELERATING DIFFUSION LLMS VIA SUFFIX PRUNING AND DYNAMIC DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Large Language Models (dLLMs) offer a compelling paradigm for natural language generation, leveraging parallel decoding and bidirectional attention to achieve superior global coherence compared to autoregressive models. While recent works have accelerated inference via KV cache reuse or heuristic decoding, they overlook the intrinsic inefficiencies within the block-wise diffusion process. Specifically, these methods suffer from spatial redundancy by modeling sparse suffix regions uniformly and temporal inefficiency by applying fixed denoising schedules across decoding. To address this, we propose Streaming-dLLM, a training-free framework that streamlines inference across both spatial and temporal dimensions. Spatially, we introduce approximated suffix pruning that leverages context modeling redundancy to compress suffix cost from linear to constant complexity, decoupling inference cost from sequence length. Temporally, we employ a dynamic confidence aware strategy that adaptively modulates denoising thresholds based on global mask state evolution, allowing the model to skip unnecessary iterations for converged tokens. Experiments on five text-only dLLMs and one diffusion multimodal large language model (dMLLM) across six benchmarks show that Streaming-dLLM matches or improves the accuracy of existing acceleration methods while achieving speedups of up to $5.4\times$ over these methods. For long sequence generation, it outperforms the strongest baseline by 2.5 accuracy points with $4.7\times$ higher throughput.

1 INTRODUCTION

Diffusion Large Language Models (dLLMs) have emerged as a promising generative paradigm in natural language processing (Nie et al., 2025; Ye et al., 2025; Zhu et al., 2025a;b; You et al., 2026; Sahoo et al., 2024; Wang et al., 2026). In contrast to traditional autoregressive language models (LLMs) (Achiam et al., 2023; Guo et al., 2025; Grattafiori et al., 2024; Yang et al., 2025), dLLMs employ parallel decoding with bidirectional attention, enabling information integration across all sequence positions and fostering superior global coherence and semantic consistency (Li et al., 2022; Arriola et al., 2025). This architectural shift is expected to alleviate the sequential inference bottleneck and the unidirectional context dependency inherent in autoregressive generation, especially for long sequences.

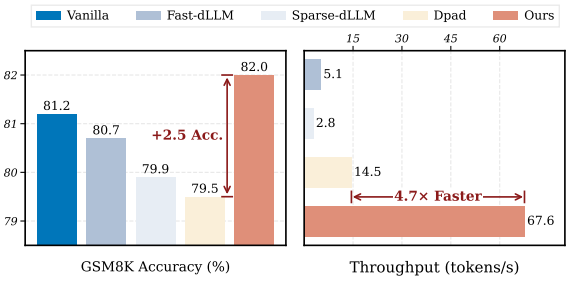


Figure 1: GSM8K accuracy and decoding throughput across dLLM acceleration methods. Compared with Dpad, Streaming-dLLM gains 2.5 percentage points in accuracy and achieves a 4.7× speedup.

Despite these advantages, existing dLLMs lag behind autoregressive models in inference speed (Li et al., 2025; Zhang et al., 2025). This gap primarily arises from the intrinsic nature of diffusion decoding. In particular, token representations evolve continuously across denoising steps, which prevents the effective reuse of the optimized Key–Value (KV) Cache. Furthermore, parallel gener-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ation amplifies sensitivity to decoding uncertainty during each denoising step, often necessitating extra denoising iterations to preserve generation quality.

Consequently, improving the inference efficiency of dLLMs has attracted substantial attention recently (Wu et al., 2023; 2025a; Liu et al., 2025a; Zhao et al., 2025; Liu et al., 2025b; Wei et al., 2025; Chen et al., 2025a; Tian et al., 2025; Ma et al., 2025b; 2026). For instance, dKV-Cache (Ma et al., 2025a) exploits the partial stability of token states to enable limited cache reuse, while Fast-dLLM (Wu et al., 2025b) adopts semi-autoregressive block-wise generation combined with confidence decoding. While these approaches achieve noticeable acceleration, they overlook the fundamental redundancy deeply rooted in the block-wise diffusion process. **Spatially**, mask tokens in the suffix region carry limited and homogeneous information, yet standard attention mechanisms process every suffix token at every step. Since the suffix region scales with the target generation length, the model wastes substantial computation on low value mask positions during the early and middle stages of diffusion. **Temporally**, the uncertainty within each generation block converges at varying rates. However, conventional thresholding strategies apply a static criterion, failing to adapt to this dynamic behavior. Consequently, high confidence tokens are often forced back into a masked state in subsequent iterations, which restricts the effective throughput of parallel decoding and may disrupt the refinement of truly uncertain tokens.

To address these challenges, we propose **Streaming-dLLM**, a training-free framework for fast and scalable inference. Our method is motivated by two key observations: the diminishing attention influence of distant suffix tokens and the progressive refinement of token states during diffusion. To tackle spatial inefficiency, we introduce approximated suffix pruning. Instead of attending to the entire suffix, we preserve only a small set of informative suffix blocks alongside positional cues of the final length to approximate global context. This design drastically reduces the computational cost of iterative inference without compromising generation quality. To address temporal inefficiency, we propose a dynamic confidence aware parallel decoding strategy. This mechanism adaptively adjusts decoding thresholds based on both the diffusion stage and the intra-block confidence distribution. Furthermore, we incorporate an early exit mechanism to terminate decoding upon predicting the End-of-Sequence (EOS) token, avoiding unnecessary computation for the remainder of the block sequence. Streaming-dLLM integrates these complementary components to optimize decoding along both spatial and temporal dimensions. Extensive experiments across diverse models, tasks, and generation lengths demonstrate that Streaming-dLLM consistently improves inference efficiency while preserving accuracy, with larger gains for longer sequences.

Our contributions are summarized as follows:

- We identify two fundamental inefficiencies in block-wise diffusion decoding: spatial redundancy from repeatedly processing low information suffix tokens and temporal redundancy from applying static decoding criteria throughout denoising.
- We design Streaming-dLLM, a training-free framework comprising approximated suffix pruning (ASP) and dynamic confidence aware parallel decoding (DCAD) to eliminate redundant computation across both dimensions.
- Experiments on five text-only dLLMs and one diffusion multimodal large language model (dMLLM) across six benchmarks demonstrate that Streaming-dLLM matches or improves the accuracy of existing acceleration methods while achieving speedups of up to $5.4\times$ over these methods. At a generation length of 2,048 tokens, it outperforms the strongest acceleration baseline by 2.5 accuracy points while achieving a $4.7\times$ speedup.

2 Related Works

2.1 Diffusion Language Models

Diffusion models have emerged as a highly promising generative paradigm for text, serving as an alternative to autoregressive models via a forward masking and reverse denoising process (Li et al., 2025; Zhu et al., 2025c). In general, existing dLLMs are broadly categorized into continuous space (Li et al., 2022; Yu et al., 2022; Lin et al., 2023; Gao et al., 2024; Mahabadi et al., 2024) and discrete space (Austin et al., 2021a; Nie et al., 2025; Song et al., 2025; Ye et al., 2025; Zhu et al., 2025a) approaches, depending on the space in which diffusion is performed.

Focusing on continuous space, Diffusion-LM (Li et al., 2022) pioneers this direction by applying Gaussian diffusion directly to token embeddings. GENIE (Lin et al., 2023) introduces a large scale pretraining strategy with a paragraph level denoising objective that reconstructs corrupted text spans, enabling effective learning from massive corpora. In contrast, discrete space dLLMs define diffusion directly over the token vocabulary. D3PM Austin et al. (2021a) defines the forward noising procedure as a discrete Markov chain with a predefined transition matrix. LLaDA Nie et al. (2025) further advances this line of work by training discrete diffusion language models from scratch using a masked-token cross entropy objective. Overall, dLLMs have demonstrated strong empirical performance across multiple benchmarks and are becoming increasingly competitive with leading autoregressive language models.

2.2 Acceleration Techniques for dLLMs

To fully unleash the potential of dLLMs, recent studies have explored various methods (Feng et al., 2025). In terms of caching mechanisms, dKV-Cache (Ma et al., 2025a) introduces a delayed KV Cache to reduce redundant memory usage, while Elastic-Cache (Nguyen-Tri et al., 2025) leverages attention aware drift to perform adaptive KV refreshes, thereby improving cache efficiency. Regarding parallel decoding, Fast-dLLM (Wu et al., 2025b) proposes a confidence threshold parallel decoding to accelerate the diffusion process. ReMDM (Wang et al., 2025) introduces a remasking sampler at inference, allowing the model to revise previously generated tokens and achieve a flexible trade-off between computational cost and generation quality. For model pruning, Dpad (Chen et al., 2025b) relies on a window sampling that scales with sequence length, which may lose positional information in the suffix region. Sparse-dLLM (Song et al., 2026) follows a compute-then-evict paradigm, requiring full attention computation before eviction, which incurs additional computational overhead. In general, these methods remain constrained by the intrinsic characteristics of the block-wise diffusion decoding, often exhibiting spatial redundancy and temporal inefficiency during generation, which limits their potential for further acceleration.

3 Method

3.1 Preliminary

Contrary to autoregressive LLMs, dLLMs perform inference through iterative decoding, where masked tokens are progressively reconstructed into the final output over discrete diffusion steps T. For a generation task with target length L, the initial sequence is defined as:

$$x^{(0)} = [p_0, [\text{MASK}], \ldots, [\text{MASK}]], \tag{1}$$

where $x^{(t)}$ denotes the state of the entire sequence at time $t \in [0, T]$, p_0 represents the input prompt and $[\text{MASK}]$ indicates a masked token. The masked sequence of length L is partitioned into N non-overlapping blocks with K tokens in each block, such that $L = N \times K$. For block $B_n^{(t)}, n \in \{0, \ldots, N-1\}$:

$$\begin{aligned} I_n &= \{p_L + nK, \ldots, p_L + (n+1)K - 1\}, \\ B_n^{(t)} &= x_{I_n}^{(t)} = \left[x_{p_L+nK}^{(t)}, \ldots, x_{p_L+(n+1)K-1}^{(t)}\right]. \end{aligned} \tag{2}$$

Here, I_n denotes the token indices in the n-th block, and p_L is the index offset corresponding to the prompt length. This generation paradigm is referred to as block-wise diffusion decoding. In particular, when $K = 1$, the decoding process degenerates into an autoregressive-like generation scheme. Conversely, when $K = L$, it simplifies to pure diffusion decoding with all tokens generated in parallel. Each block is decoded over M diffusion steps ($T = N \times M$). At iteration $t \in \{0, \ldots, M-1\}$, the model predicts logits for all positions:

$$z^{(t)} = f_\theta(x^{(t)}), \tag{3}$$

where f_θ denotes the dLLMs parameterized by θ. Sampling and token prediction are then performed only for positions $i \in B_n$ where $x_i^{(t)} = [\text{MASK}]$:

$$\begin{aligned} c_i^{(t)} &= \max(\text{Softmax}\left(z_i^{(t)}\right)), \\ \hat{x}_i^{(t)} &= \arg\max(\text{Softmax}\left(z_i^{(t)}\right)). \end{aligned} \tag{4}$$

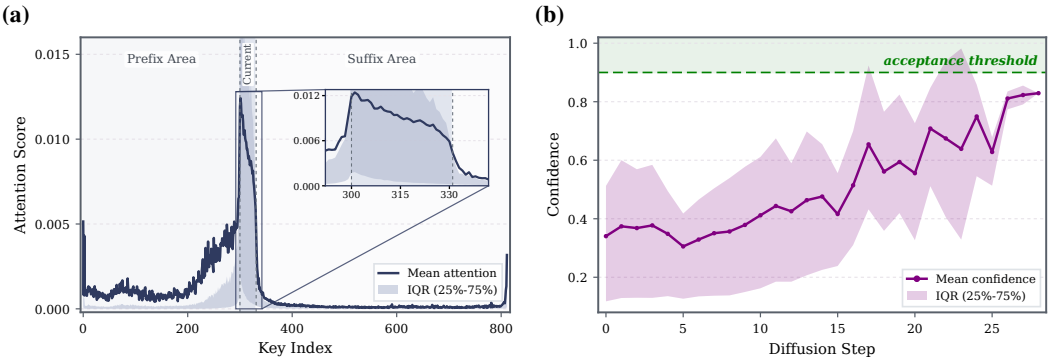


Figure 2: **(a) Spatial redundancy.** Attention distribution at Layer 31 of LLaDA-1.5 between the current block and the full input sequence. The solid line and shaded region denote the mean and interquartile range, respectively. Attention concentrates on nearby suffix blocks and the final token, revealing substantial redundancy in intermediate suffix positions. **(b) Temporal dynamics.** Token confidence during block-wise generation on GSM8K with a generation length of 256 and a block size of 32. Confidence increases as the mask ratio decreases, suggesting that a fixed high threshold can delay the acceptance of sufficiently confident tokens.

Finally, token states are updated according to:

$$x_i^{(t+1)} = \begin{cases} \hat{x}_i^{(t)}, & i \in \{i_k \mid S(\hat{x}_{i_k}^{(t)}, c_{i_k}^{(t)})\}, \\ x_i^{(t)}, & \text{otherwise.} \end{cases} \tag{5}$$

where $S(\cdot)$ denotes the token selection strategy, which determines the subset of positions to be updated based on predicted tokens and their corresponding confidence scores. The decoding iteration for block n terminates when $\{i \in B_n \mid x_i = [\text{MASK}]\} = \emptyset$, after which the model proceeds to decode block $n+1$.

3.2 Rethinking Inference of dLLMs

Diffusion language models leverage bidirectional attention and parallel decoding to exploit global contextual information, thereby improving generation quality. However, whether this design remains necessary and efficient at every diffusion step during inference, especially for long text generation, has yet to be systematically examined. In the diffusion decoding framework (Equation 3), bidirectional attention is computed over the full sequence $x^{(t)}$ at each prediction. Unlike autoregressive models, the input sequence of dLLMs consists of a semantically dense prompt region and a large number of ungenerated mask tokens. When the prompt is short and the generation length is large, the model is forced to repeatedly attend to an extensive masked region, which incurs substantial computational overhead. As masked tokens, these positions also provide a large amount of homogeneous semantic information, while whether such information is actually necessary or effective during generation remains unclear.

To investigate this, we analyze attention distribution between the currently generated block and the entire input sequence. As shown in Figure 2a, attention over the suffix region decays with distance. Although the suffix constitutes most of the sequence, many intermediate positions receive near-zero scores. Conversely, the current generation block primarily attends to a few neighboring blocks within the suffix region, as well as the token at the end position. This suggests that only a limited portion of the suffix mask region contains meaningful structural information, while the remaining positions largely serve as redundant placeholders.

Meanwhile, we examine token confidence during parallel decoding within the block. Previous method (Wu et al., 2025b) typically uses a fixed confidence threshold to determine whether a token should be updated, which implicitly assumes that the confidence distribution remains relatively stable during decoding. In light of this, we collect statistics from the iterative decoding process on 100 GSM8K samples and visualize the first generated block. As diffusion progresses, the sequence becomes increasingly determined, which is empirically reflected by a consistent rise in token confidence within each generation block (Figure 2b). Moreover, we plot token confidences within the interquartile range (IQR, $25\% - 75\%$), which better reflects typical behavior of most tokens during

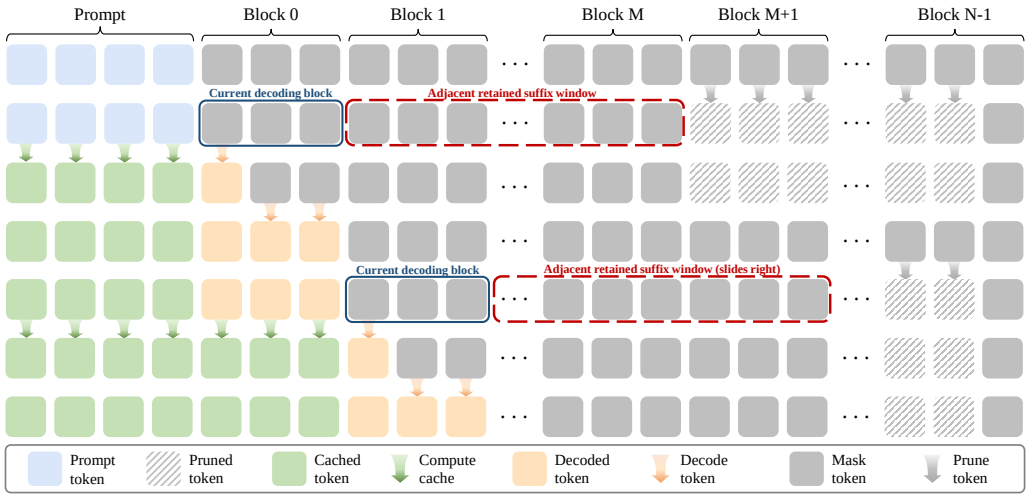


Figure 3: Illustration of approximated suffix pruning. For each block, the nearest neighboring region following the current block is retained using a sliding window (red dashed box) and concatenated with the trailing position to form an approximate suffix region.

decoding. These observations indicate that using a fixed confidence threshold is suboptimal: early steps require a high threshold to ensure generation quality, while later steps benefit from a lower threshold to allow timely token acceptance and improve parallel decoding efficiency. Additional analyses and visualizations across models, layers and datasets are provided in the Appendix A.

Overall, above analysis reveals two complementary inefficiencies in the reasoning process of dLLMs: (1) Spatially, a substantial amount of computation is devoted to suffix mask tokens that carry limited semantic information, leading to a clear mismatch between the computation and informative content. (2) Temporally, current confidence decoding strategy fails to capture the dynamic evolution of token masking states and confidence within a block, forcing tokens with sufficient confidence to undergo redundant diffusion steps. These observations motivate our redesign of both the suffix modeling approach and the decoding strategy for the reasoning stage.

3.3 STREAMING-DLLM

Motivated by the two inefficiencies identified above, we propose Streaming-dLLM, a training-free framework designed to accelerate dLLMs inference. The insight is to streamline block-wise diffusion decoding by progressively eliminating redundant computation along both spatial and temporal dimensions. In response to spatial redundancy in the suffix region, when decoding the n-th block, Streaming-dLLM first applies approximated suffix pruning to retain only a minimal set of structural cues that are most critical for generating the current block. In parallel, to accommodate the progressive evolution of token states during diffusion, we adaptively adjust decoding thresholds at each iteration based on the current masking status of the sequence. This allows high-confidence tokens to be finalized early, reducing unnecessary computation in subsequent decoding steps. Additionally, we incorporate an early exit mechanism to terminate decoding once the EOS token is predicted.

Approximated Suffix Pruning. Following Equations 1 and 2, the sequence at diffusion step t is:

$$x^{(t)} = [p_0, \hat{B}_0^{(t)}, \ldots, \hat{B}_{c-1}^{(t)}, B_c^{(t)}, \ldots, B_{N-1}^{(t)}], \tag{6}$$

where $\hat{B}_i^{(t)}, i \in [0, c-1]$ denotes the blocks that have already been decoded. Together with the input prompt p_0, they form the prefix region $S_{\text{prefix}}^{(t)}$. $B_c^{(t)}$ represents the block being decoded at diffusion step t, referred to as the current region $S_{\text{current}}^{(t)}$. The remaining blocks $B_j^{(t)}$ for $j \in (c, N-1]$ constitute the suffix region $S_{\text{suffix}}^{(t)}$.

For the suffix region, we adopt a sliding window strategy that retains a suffix window of size w, covering the tokens closest to current region. In addition, we preserve the positional information of the final token to provide a coarse representation of the entire suffix while maintaining the logical

ordering of tokens via RoPE (Su et al., 2024) position IDs:

$$\tilde{I} = \{l, \ldots, l+w-1\} \cup \{p_L + L - 1\}, \qquad \tilde{S}^{(t)}_{\text{suffix}} = \left(x_i^{(t)}\right)_{i \in \tilde{I}}, \tag{7}$$

here, $l = p_L + (c+1)K$ denotes the starting index of the suffix region. The approximated suffix is then concatenated with the prefix and current regions to form the input sequence at diffusion step t, which is subsequently fed into the diffusion language model for prediction.

$$\tilde{x}^{(t)} = \text{concat}\Big(S^{(t)}_{\text{prefix}}, S^{(t)}_{\text{current}}, \tilde{S}^{(t)}_{\text{suffix}}\Big). \tag{8}$$

This approximation follows the observation in Figure 2a. During generation of $S^{(t)}_{\text{current}}$, the suffix region provides coarse structural constraints, such as positional continuity and termination cues, rather than detailed semantic information. Meanwhile, tokens in the distant suffix remain uncertain throughout most diffusion steps, limiting their contribution to refining the current block. Accordingly, as shown in Figure 3, we retain only the suffix blocks adjacent to $S^{(t)}_{\text{current}}$ along with the terminal positional token, which preserves the essential structural signals required for stable decoding. This design reduces redundant computation in both attention and denoising, effectively compressing the suffix cost from linear to constant complexity with respect to sequence length, thereby decoupling inference cost from sequence length while maintaining global coherence across blocks.

Dynamic Confidence Aware Parallel Decoding. During iterative block-wise decoding, the KV representations of the prefix region are computed once at the first diffusion step and reused across all subsequent steps within the same block. This approach significantly reduces redundant attention computation on the prefix while preserving information from previously decoded tokens.

In the prediction forward, the query vectors are constructed from the current generation region $S^{(t)}_{\text{current}}$ and the approximated suffix region $\tilde{S}^{(t)}_{\text{suffix}}$. These queries attend to the KV representations of the entire sequence $\tilde{x}^{(t)}$ to compute attention. Attention scores are processed by the remaining model components to produce the prediction logits. Following Equation 4, we obtain generated token $\hat{x}_i^{(t)}$ for each position within query area, along with its corresponding confidence score $c_i^{(t)}$. The confidence scores are used to determine whether a token can be finalized at the current diffusion step. We define a confidence-aware selection function $S(\cdot)$ that decides whether a predicted token $\hat{x}_{i_k}^{(t)}$ is accepted according to its confidence $c_{i_k}^{(t)}$. Tokens whose confidence exceeds the threshold $\tau^{(t)}$ are finalized, while the remaining tokens stay masked and continue to be refined in subsequent diffusion iterations. When no token satisfies the confidence threshold, we finalize the masked token with the highest confidence. Formally, the selection rule can be defined as:

$$S(\hat{x}_i^{(t)}, c_i^{(t)}) = \begin{cases} \hat{x}_i^{(t)}, & c_i^{(t)} \geq \tau^{(t)}, \\ \hat{x}^{(t)}_{\arg\max_i c_i^{(t)}}, & \max_j c_j^{(t)} < \tau^{(t)}, \\ \text{None}, & \text{otherwise.} \end{cases} \tag{9}$$

To align with the progressive refinement of token states during diffusion, we adopt an adaptive threshold that depends on the masking status of the sequence. Concretely, we quantify this status using the ratio of masked tokens r_{mask} in the current query region, which serves as a lightweight proxy for the overall uncertainty of the sequence. A higher r_{mask} indicates that more tokens remain unresolved and the model is still in an early diffusion stage, while a lower r_{mask} reflects a more determined sequence with reduced uncertainty. The threshold at step t is defined as:

$$\tau^{(t)} = \tau_0 \left(1 - \alpha(1 - r_{\text{mask}})\right), \tag{10}$$

where τ_0 denotes the base confidence threshold and α controls the adaptation strength. As diffusion gradually progresses, r_{mask} decreases, leading to a lower threshold that allows more valid tokens to be finalized. Moreover, α directly controls the trade-off between decoding efficiency and generation quality. This design reflects the intuition that later diffusion steps involve lower uncertainty and can tolerate more aggressive token commitment, thereby improving decoding efficiency while maintaining stable generation quality.

Early Exit For Block Diffusion. To further improve inference efficiency, we introduce an early exit mechanism for block-wise diffusion decoding. During generation, if a block predicts the EOS token,

Table 1: Main results at a generation length of 512 on Dream-Base, LLaDA-1.5, and LLaDA-Instruct. Each benchmark reports accuracy (Acc.) and decoding throughput (TPS), with relative speedup over the corresponding vanilla baseline shown in orange. The final group reports mean accuracy and mean TPS across the four benchmarks; its speedup is computed relative to the corresponding mean baseline TPS. Among acceleration methods, the best and second-best accuracy values within each model and benchmark are bolded and underlined, respectively. '*' marks results reproduced using the official implementation.

Method	HumanEval (0-shot)		GSM8K (5-shot)		MBPP (3-shot)		MATH (4-shot)		Average↑	
	Acc.	TPS	Acc.	TPS	Acc.	TPS	Acc.	TPS	Acc.	TPS
Dream-7B-Base										
Vanilla	54.3	13.7 (1×)	74.2*	7.1 (1×)	55.6	8.7 (1×)	39.8	8.6 (1×)	56.0	9.5 (1×)
+ dKV-Cache	49.4*	15.7 (1.1×)	71.6*	12.8 (1.8×)	53.0*	11.6 (1.3×)	38.5*	11.6 (1.3×)	53.1	12.9 (1.4×)
+ Fast-dLLM	54.3	40.2 (2.9×)	74.1*	41.7 (5.9×)	55.2	63.1 (7.3×)	39.3	54.4 (6.3×)	55.7	49.9 (5.2×)
+ Sparse-dLLM	52.4*	44.3 (3.2×)	72.2*	40.5 (5.7×)	51.4*	38.6 (4.4×)	35.7*	36.9 (4.3×)	52.9	40.1 (4.2×)
+ Dpad	52.4*	52.6 (3.8×)	72.1*	50.0 (7.0×)	54.8*	51.9 (6.0×)	39.2*	39.9 (4.6×)	54.6	48.6 (5.1×)
+ **Streaming-dLLM**	**54.6**	**72.3** (5.3×)	**74.7**	**94.1** (13.3×)	**55.8**	**92.4** (10.6×)	**39.4**	**96.0** (11.2×)	**56.1**	**88.7** (9.3×)
LLaDA-8B-Instruct										
Vanilla	43.9	11.9 (1×)	77.5	3.2 (1×)	14.8	4.2 (1×)	37.2	6.2 (1×)	43.4	6.4 (1×)
+ dKV-Cache	**45.1***	14.3 (1.2×)	77.6*	5.5 (1.7×)	14.6*	6.9 (1.6×)	**36.3***	10.1 (1.7×)	43.4	9.2 (1.4×)
+ Fast-dLLM	44.5	52.0 (4.4×)	77.2	32.1 (10.0×)	13.8	39.7 (9.4×)	36.0	43.2 (7.0×)	42.9	41.8 (6.5×)
+ Sparse-dLLM	42.1*	30.6 (2.6×)	76.0*	15.2 (4.8×)	13.0*	16.4 (3.9×)	34.6*	25.8 (4.2×)	41.4	22.0 (3.5×)
+ Dpad	43.5*	60.4 (5.1×)	78.2*	40.4 (12.7×)	39.4*	18.3 (4.4×)	35.6*	22.6 (3.6×)	49.2	35.4 (5.6×)
+ **Streaming-dLLM**	44.5	**75.8** (6.4×)	**78.7**	**65.9** (20.6×)	**40.8**	**66.7** (15.9×)	36.1	**63.2** (10.2×)	**50.0**	**67.9** (10.7×)
LLaDA-1.5-8B										
Vanilla	40.5*	2.9 (1×)	81.0*	2.5 (1×)	38.2*	0.9 (1×)	37.1*	4.8 (1×)	49.2	2.8 (1×)
+ dKV-Cache	**40.2***	3.3 (1.1×)	**81.3***	4.2 (1.7×)	38.1*	1.5 (1.7×)	**35.1***	7.5 (1.6×)	**48.7**	4.1 (1.5×)
+ Fast-dLLM	39.8*	13.6 (4.7×)	80.4	25.8 (10.3×)	38.1*	16.5 (18.3×)	**35.1**	38.3 (7.9×)	48.4	23.6 (8.5×)
+ Sparse-dLLM	39.0*	6.8 (2.3×)	81.2*	11.2 (4.5×)	37.8*	3.3 (3.6×)	33.5*	18.3 (3.8×)	47.9	9.9 (3.6×)
+ Dpad	39.6*	15.6 (5.4×)	81.2*	19.0 (7.6×)	38.2*	13.5 (15.0×)	34.8*	21.7 (4.5×)	48.5	17.5 (6.3×)
+ **Streaming-dLLM**	**40.2**	**26.7** (9.2×)	81.2	**69.8** (28.0×)	**38.4**	**61.4** (68.2×)	**35.1**	**62.4** (13.0×)	**48.7**	**55.1** (19.8×)

all subsequent blocks are skipped. This strategy prevents unnecessary computation on remaining blocks, reducing inference latency while ensuring that the generated sequence remains complete. By leveraging this property, Streaming-dLLM can adaptively reduce the decoding steps without compromising sequence integrity, further enhancing efficiency in text generation scenarios.

4 Experiment

4.1 Experimental Setup

To evaluate the effectiveness of Streaming-dLLM, we conduct experiments on dLLMs, including Dream-v0-7B-Base (Ye et al., 2025), LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2025a), openPangu (Tian et al., 2025), and Fast-dLLM v2 (Wu et al., 2025a). We further evaluate the applicability of our method to the multimodal diffusion model LLaDA-V (You et al., 2026). We compare our method with the vanilla implementations and dLLM inference acceleration methods (Ma et al., 2025a; Wu et al., 2025b; Song et al., 2026; Chen et al., 2025b; Ma et al., 2025b). The evaluation covers mathematical reasoning benchmarks (GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2020)), code generation benchmarks (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b)), and multimodal mathematical reasoning benchmarks for LLaDA-V (MathVerse (Zhang et al., 2024), MathVista (Lu et al., 2023)). In addition, we assess the models under varying generation lengths to examine the scalability and robustness of our approach.

Our metrics capture both generation quality and inference efficiency, measured by benchmark accuracy, throughput (tokens per second, TPS) and inference latency per sample. For throughput computation, we count only non EOS tokens across the entire generated sequence, which ensures the reliability and comparability of the reported results. All experiments are conducted within the

lm-eval framework to ensure consistency and reproducibility, using a single NVIDIA A800 80GB GPU. Additional optimization details are provided in the Appendix (section B.3).

Table 2: Long-sequence generation results on LLaDA-1.5 at a generation length of 2048.

Method	GSM8K Acc.	GSM8K TPS
Vanilla	81.2	0.3 (1.0×)
+ Fast-dLLM	80.7	5.1 (17.0×)
+ Sparse-dLLM	79.9	2.8 (9.3×)
+ Dpad	79.5	14.5 (48.3×)
+ Streaming-dLLM	**82.0**	**67.6** (225.3×)

Table 3: Acceleration results on recent pure block dLLMs.

Model	Method	GSM8K Acc.	GSM8K TPS	MATH Acc.	MATH TPS
Fast-dLLM v2	Vanilla	**82.0**	7.3 (1.0×)	38.3	8.9 (1.0×)
	+ Fast-dLLM	81.1	41.7 (5.7×)	38.1	37.2 (4.2×)
	+ Streaming-dLLM	81.5	**109.2** (15.0×)	**38.6**	**96.6** (10.9×)
openPangu	Vanilla	78.7	1.8 (1.0×)	46.0	2.7 (1.0×)
	+ Fast-dLLM	78.7	7.2 (4.0×)	46.3	10.4 (3.9×)
	+ Streaming-dLLM	**78.9**	**38.7** (21.5×)	**46.5**	**42.6** (15.8×)

4.2 Main Results

Standard and long-sequence benchmarks. Table 1 summarizes the results on three dLLM backbones and four standard benchmarks. Relative to their vanilla backbones, Streaming-dLLM achieves average speedups of 9.3×, 10.7×, and 19.8× on Dream-Base, LLaDA-Instruct, and LLaDA-1.5, respectively, while maintaining competitive average accuracy. Complete results at generation lengths 256 and 512 are reported in Appendix tables 6 to 8, where the same efficiency trend remains consistent across generation lengths. For long-sequence generation at length 2048, Streaming-dLLM achieves 67.6 TPS on LLaDA-1.5, delivering 4.7× and 13.3× higher throughput than Dpad and Fast-dLLM, respectively, with accuracy gains of 2.5 and 1.3 points, as shown in table 2. These results demonstrate that Streaming-dLLM consistently improves inference efficiency across model backbones and sequence lengths without compromising generation quality.

Recent pure block-diffusion models. We apply Streaming-dLLM to Fast-dLLM v2 and openPangu. As shown in table 3, it increases Fast-dLLM v2 throughput to 109.2 TPS on GSM8K and 96.6 TPS on MATH, approximately 2.6× that of Fast-dLLM, with comparable or better accuracy. On openPangu, it reaches 38.7 and 42.6 TPS, respectively 5.4× and 4.1× that of Fast-dLLM, while achieving the best accuracy on both benchmarks. These results confirm the applicability of our training-free strategy to pure block-diffusion architectures. Additional comparisons with dInfer (Ma et al., 2025b) and the dual-cache mechanism of Fast-dLLM (Wu et al., 2025b) are provided in the Appendix.

Table 4: Multimodal results on MathVerse and MathVista. Each benchmark reports Acc. and TPS; orange values denote speedup over Vanilla.

Method	MathVerse Acc.	MathVerse TPS	MathVista Acc.	MathVista TPS
Vanilla	**59.7**	5.56 (1.0×)	28.5	5.17 (1.0×)
+ Fast-dLLM	56.6	28.2 (5.1×)	28.6	23.3 (4.5×)
+ Streaming-dLLM	58.1	**57.7** (10.4×)	**29.5**	**42.6** (8.2×)

Multimodal diffusion model acceleration. We further evaluate Streaming-dLLM on a multimodal diffusion model using MathVerse and MathVista. As shown in table 4, our method reaches 57.7 TPS on MathVerse, corresponding to a 10.4× speedup over vanilla and a 2.0× improvement over Fast-dLLM, while retaining an accuracy of 58.1. On MathVista, it improves accuracy from 28.5 to 29.5 and achieves 42.6 TPS, yielding an 8.2× speedup over vanilla and a 1.8× improvement over Fast-dLLM. These results demonstrate that the proposed acceleration strategy extends beyond text-only dLLMs to multimodal diffusion models.

4.3 Analysis

Inference latency evaluation. Extended per-sample latency results are provided in section B.2. Across the three backbones and four benchmarks, Streaming-dLLM reduces per-sample latency by 79.2%–98.4% relative to the corresponding vanilla models. Compared with Fast-dLLM, it reduces per-sample latency by up to 85.5%, showing that the throughput improvements also translate into substantially shorter user-perceived response times.

Ablation of each module. To evaluate our framework, we conduct ablation studies on three modules using LLaDA-1.5 with Prefix Cache: Approximated Suffix Pruning, Dynamic Confidence Aware Parallel Decoding, and Early Exit for Block Diffusion. Results are summarized in table 5. Indi-

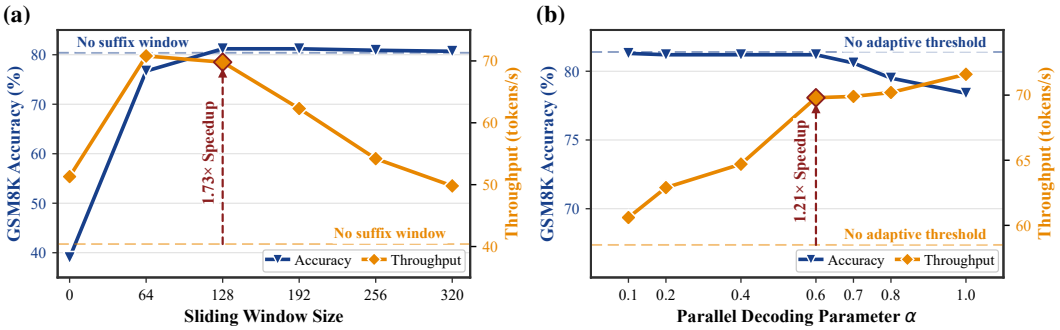


Figure 4: **(a)** Effect of the sliding window size in approximated suffix pruning, retaining the full suffix corresponds to a window size of 512. **(b)** Effect of the parallel decoding parameter α, disabling adaptive thresholding corresponds to $\alpha = 0$.

vidually, all three improve throughput. Suffix pruning removes redundant computation, increasing TPS from 8.2 to 24.3, while dynamic decoding reaches 44.6 TPS by adapting token acceptance to evolving confidence. Early exit achieves 15.1 TPS by avoiding unnecessary later block iterations. The pairwise results further show complementary benefits: combining suffix pruning with dynamic decoding or early exit yields 50.3 and 38.9 TPS, respectively, while dynamic decoding with early exit reaches 56.4 TPS. Enabling all three components produces the highest throughput of 69.8 TPS, corresponding to an $8.5\times$ speedup over the Prefix Cache baseline, while improving accuracy from 81.0 to 81.2. Overall, the three components provide complementary efficiency gains.

Hyperparameter sensitivity. We further conduct ablation studies on key hyperparameters using the GSM8K dataset with LLaDA-1.5. We first examine the effect of the sliding window size w in approximated suffix pruning. This parameter controls the fraction of the suffix region retained, balancing generation quality and inference efficiency. As shown in fig. 4a, both accuracy and throughput improve as the window size increases. Once the window exceeds a certain threshold, accuracy saturates while throughput begins to decline. When $w = 128$, a favorable balance is achieved, where accuracy is slightly improved and a 1.73× speedup is obtained relative to the full window size of 512. This result indicates that most suffix tokens contribute marginally to the generation of the current block, whose information can be sufficiently captured by a more compact near area.

Table 5: Component ablation on GSM8K using LLaDA-1.5 with Prefix Cache at a generation length of 512. **Suf.**, **Dyn.**, and **Exit.** denote approximated suffix pruning, dynamic confidence-aware parallel decoding, and early exit, respectively. Speedups are measured relative to the Prefix Cache baseline.

Method	Acc. (%)	TPS
LLaDA-1.5 w/ Prefix Cache	81.0	8.2 (1.0×)
+ Suf.	**81.2**	24.3 (3.0×)
+ Dyn.	81.0	44.6 (5.4×)
+ Exit.	80.4	15.1 (1.8×)
+ Suf. + Dyn.	**81.2**	50.3 (6.1×)
+ Suf. + Exit.	**81.2**	38.9 (4.7×)
+ Dyn. + Exit.	81.0	56.4 (6.9×)
+ Streaming-dLLM	**81.2**	**69.8** (8.5×)

We next study the effect of α, which controls parallel decoding. As reported in fig. 4b, increasing α from 0.1 to 0.6 improves throughput, while larger values degrade performance, indicating a trade-off between efficiency and generation quality. This is likely due to premature parallel decoding of tokens that have not fully converged, causing confidence misalignment that accumulates across subsequent iterations. Analyses of trailing positional information, prefill, generation length, and the accuracy–throughput trade-off are provided in the Appendix (sections B.4, B.7 and B.9).

5 Conclusion

We present Streaming-dLLM, a training-free framework that addresses spatial redundancy in suffix modeling and temporal redundancy in denoising. It combines approximated suffix pruning, dynamic confidence-aware parallel decoding, and early exit to reduce redundant computation while preserving generation quality. Experiments across text-only dLLMs and a multimodal diffusion language model show that Streaming-dLLM matches or improves existing acceleration methods while delivering consistent throughput gains, particularly for long-sequence generation. These results demonstrate its effectiveness and scalability for efficient dLLM deployment.

AI USE STATEMENT

Generative AI tools were used to improve the clarity, grammar, and readability of the manuscript. All AI-assisted revisions were reviewed and edited by the authors. The authors take full responsibility for the final content of this work.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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

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

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

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

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

Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai Li, Yiran Chen, et al. Dpad: Efficient diffusion language models with suffix dropout. *arXiv preprint arXiv:2508.14148*, 2025b.

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

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. Theoretical benefit and limitation of diffusion language model. *arXiv preprint arXiv:2502.09622*, 2025.

Zhujin Gao, Junliang Guo, Xu Tan, Yongxin Zhu, Fang Zhang, Jiang Bian, and Linli Xu. Empowering diffusion models on the embedding space for text generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4664–4683, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in neural information processing systems*, 35: 4328–4343, 2022.

Zhenghao Lin, Yeyun Gong, Yelong Shen, Tong Wu, Zhihao Fan, Chen Lin, Nan Duan, and Weizhu Chen. Text generation with diffusion language models: A pre-training approach with continuous paragraph denoise. In *International Conference on Machine Learning*, pp. 21051–21064. PMLR, 2023.

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

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

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating math reasoning in visual contexts with gpt-4v, bard, and other large multimodal models. *CoRR, abs/2310.02255*, 5: 15, 2023.

Linrui Ma, Yufei Cui, Kai Han, and Yunhe Wang. Diffusion in diffusion: Breaking the autoregressive bottleneck in block diffusion models. *arXiv preprint arXiv:2601.13599*, 2026.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025a.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, et al. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025b.

Rabeeh Karimi Mahabadi, Hamish Ivison, Jaesung Tae, James Henderson, Iz Beltagy, Matthew E Peters, and Arman Cohan. Tess: Text-to-text self-conditioned simplex diffusion. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2347–2361, 2024.

Quan Nguyen-Tri, Mukul Ranjan, and Zhiqiang Shen. Attention is all you need for kv cache in diffusion llms. *arXiv preprint arXiv:2510.14973*, 2025.

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

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

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

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.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yuchuan Tian, Yuchen Liang, Shuo Zhang, Yingte Shu, Guangwen Yang, Wei He, Sibo Fang, Tianyu Guo, Kai Han, Chao Xu, et al. From next-token to next-block: A principled adaptation path for diffusion llms. *arXiv preprint arXiv:2512.06776*, 2025.

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

Yunhe Wang, Kai Han, Huiling Zhen, Yuchuan Tian, Hanting Chen, Yongbing Huang, Yufei Cui, Yingte Shu, Shan Gao, Ismail Elezi, et al. Top 10 open challenges steering the future of diffusion language model and its variants. *arXiv preprint arXiv:2601.14041*, 2026.

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

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dllm v2: Efficient block-diffusion llm. *arXiv preprint arXiv:2509.26328*, 2025a.

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

Tong Wu, Zhihao Fan, Xiao Liu, Hai-Tao Zheng, Yeyun Gong, Jian Jiao, Juntao Li, Jian Guo, Nan Duan, Weizhu Chen, et al. Ar-diffusion: Auto-regressive diffusion model for text generation. *Advances in Neural Information Processing Systems*, 36:39957–39974, 2023.

Guanyu Xu, Jiawei Hao, Li Shen, Han Hu, Yong Luo, Hui Lin, and Jialie Shen. Lgvit: Dynamic early exiting for accelerating vision transformer. In *Proceedings of the 31st ACM International Conference on Multimedia*, pp. 9103–9114, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Zebin You, Shen Nie, Xiaolu Zhang, JUN ZHOU, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. Llada-v: Large language diffusion models with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10093–10105, 2026.

Peiyu Yu, Sirui Xie, Xiaojian Ma, Baoxiong Jia, Bo Pang, Ruiqi Gao, Yixin Zhu, Song-Chun Zhu, and Ying Nian Wu. Latent diffusion energy-based model for interpretable text modeling. *arXiv preprint arXiv:2206.05895*, 2022.

Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, et al. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv preprint arXiv:2508.08712*, 2025.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pp. 169–186. Springer, 2024.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.

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

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

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

APPENDICES

A Additional Details on Attention and Token Confidence Sampling **14**

A.1 Detailed Procedure 14

A.2 Additional Visualization and Analysis 14

B Additional Experiment Details **17**

B.1 Complete Main Results at Generation Lengths 256 and 512 17

B.2 Inference Latency Experiments for dLLMs 19

B.3 Optimization and Schedule 21

B.4 Additional Analysis for Prefill and Generation Length. 21

B.5 Additional Analysis with Existing Confidence-Based Decoding Methods. 22

B.6 Additional Comparisons with Recent dLLM Methods. 23

B.7 Additional Analysis on Accuracy-Throughput Trade-off 24

B.8 Statistical Stability Analysis 24

B.9 Additional Analysis for Trailing Positional Information 24

C Others **25**

C.1 Social Impact Analysis 25

C.2 Limitation 25

A Additional Details on Attention and Token Confidence Sampling

A.1 Detailed Procedure

For fig. 2a, we analyze the attention distribution at the final layer (Layer 31) of LLaDA-1.5, which captures interactions between the currently generated block and the full input sequence. Statistics are collected from 300 GSM8K samples across all heads with a generation length of 512. For each sample, the first 300 tokens of the prompt are treated as the prefix area; tokens in [300:300+block_size] correspond to the currently generated block, while tokens in [300+block_size:] form the suffix area. In the analysis, the solid line represents the mean attention score at each diffusion step, and the shaded area indicates the interquartile range (25%–75%). The gray dashed lines separate the prefix, current generation, and suffix regions. The current generation region is located between the prefix and suffix regions, with a size of *block_size*. The results indicate that attention is largely concentrated on a few neighboring suffix blocks and the final token, whereas most intermediate positions in the suffix area receive negligible attention. This pattern suggests substantial redundancy in the suffix region during block-wise generation.

For fig. 2b, we further examine the token confidence distribution during iterative block-wise generation of LLaDA-1.5 on GSM8K with a generation length of 256. Statistics are collected from the iterative decoding process of 100 samples, focusing on the first generated block. The results show that the mean token confidence steadily increases as iterations progress. While applying a high confidence threshold helps ensure generation quality, it can also be conservative, potentially delaying the acceptance of tokens that are already sufficiently confident.

A.2 Additional Visualization and Analysis

To provide a more comprehensive and systematic understanding, we extend our analysis along various dimensions.

Attention. We evaluate attention patterns across model architectures, layers, and tasks. Specifically, we compare Dream-Base and LLaDA-1.5 on HumanEval and analyze layers 0, 7, 15, and 23 of LLaDA-1.5 on GSM8K. While different models (Dream-Base fig. 5a and LLaDA-1.5 fig. 5b) exhibit some variation in attention distributions on the same dataset (HumanEval, generation length = 512), these differences are primarily concentrated in the prefix region. We conjecture that this discrepancy may stem from differences in their training paradigms, which can lead to distinct conditioning behaviors over the prefix tokens. Despite this variation, both models consistently allocate the majority of attention mass to the current generation region and the prefix region, indicating a shared tendency to rely on already generated and conditioned context. In contrast, the suffix region mostly receives little to no attention, suggesting a high degree of redundancy in this region.

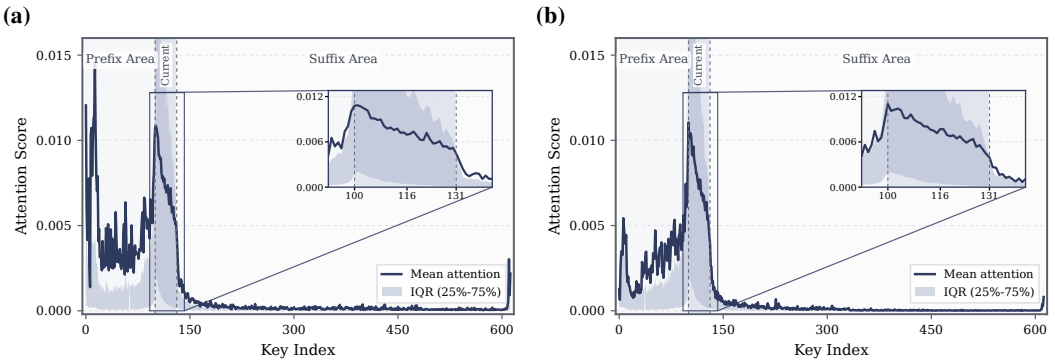


Figure 5: Attention scores at the final layer on HumanEval with a generation length of 512. **(a)** Dream-Base. **(b)** LLaDA-1.5.

Furthermore, we extend the analysis to multiple layers of LLaDA-1.5 (layers 0, 7, 15, and 23) and benchmarks (HumanEval and GSM8K), as illustrated in figs. 2a, 5b and 6. A consistent pattern emerges: attention is predominantly concentrated on nearby suffix blocks and the final token, while

intermediate positions receive minimal contribution. Notably, this locality bias persists from lower to higher layers, suggesting it is an intrinsic property of the model's diffusion-based decoding process rather than a layer-specific phenomenon. Moreover, the same trend holds across both code generation and mathematical reasoning tasks, indicating that this behavior is robust to variations in data distribution and task characteristics, and thus reflects a fundamental inductive bias of the model rather than dataset-specific artifacts.

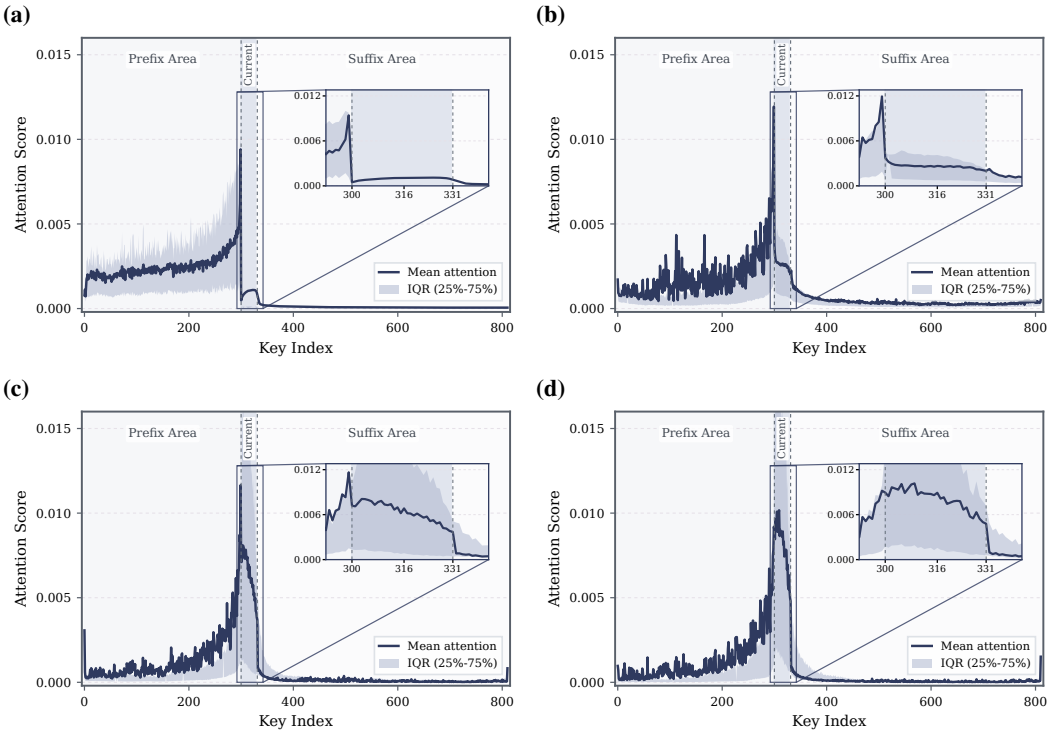


Figure 6: Attention scores across layers of LLaDA-1.5 on GSM8K with a generation length of 512. **(a)** Layer 0. **(b)** Layer 7. **(c)** Layer 15. **(d)** Layer 23.

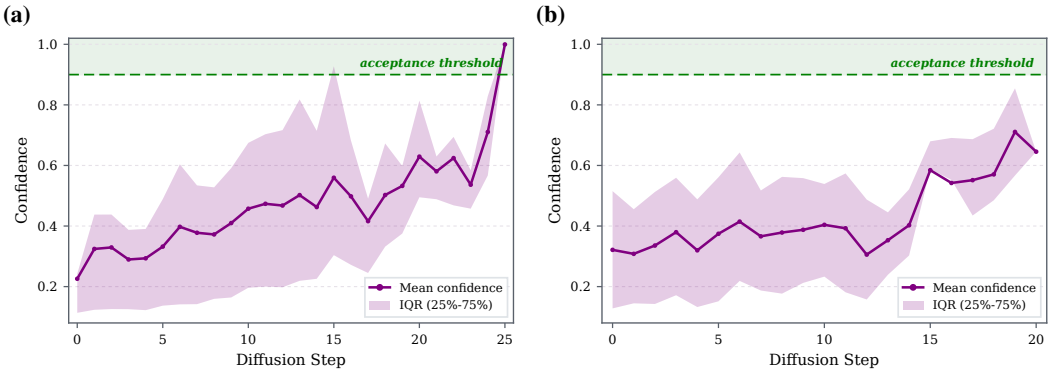


Figure 7: Token confidence for Block 0 on HumanEval with a generation length of 256. **(a)** LLaDA-1.5. **(b)** Dream-Base.

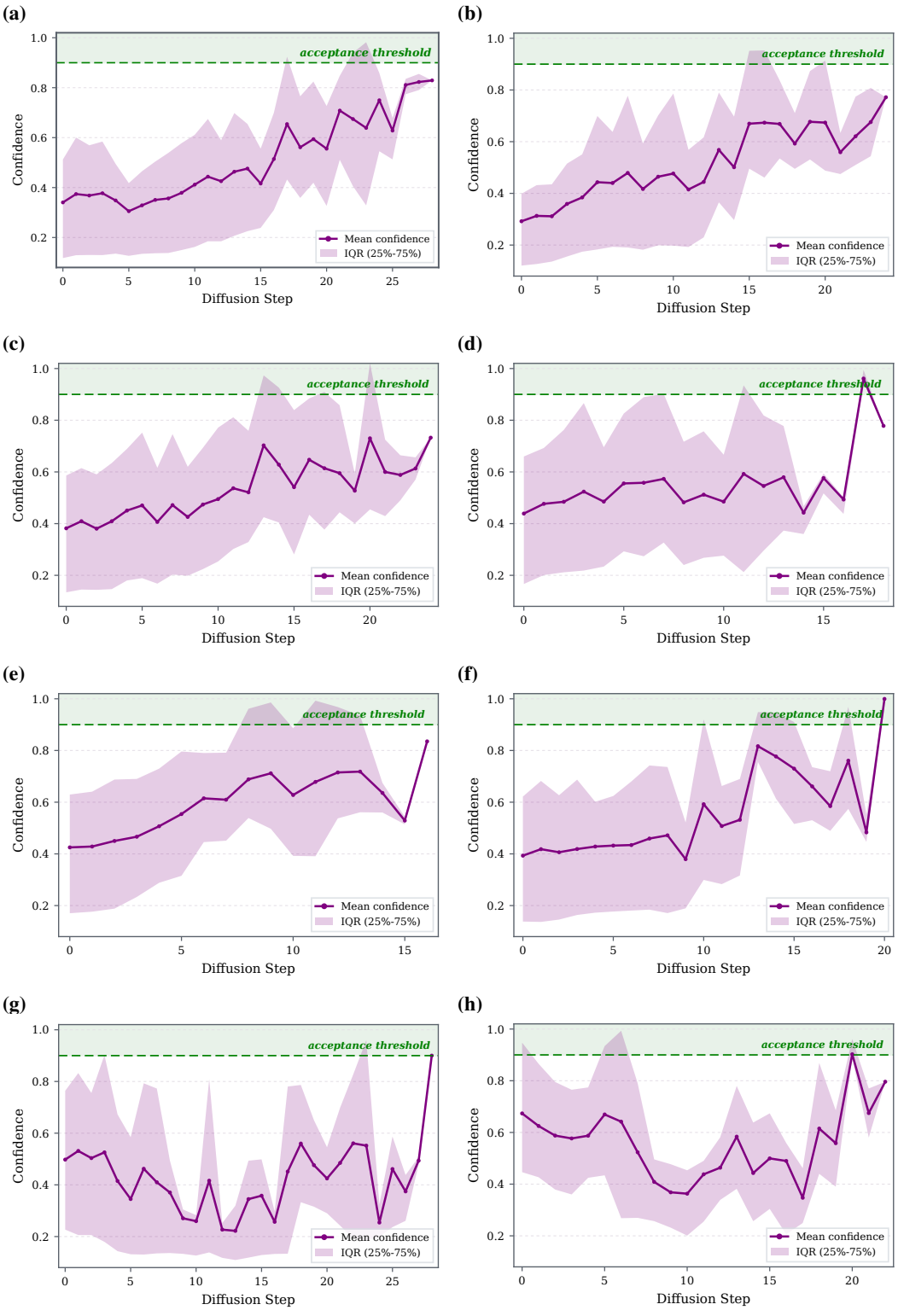


Figure 8: Token confidence across generation blocks of LLaDA-1.5 on GSM8K with a generation length of 256. **(a)–(h)** Blocks 0–7, respectively.

Confidence. We further examine decoding confidence across models (Dream-Base and LLaDA-1.5) and tasks (GSM8K and HumanEval), as illustrated in figs. 7 and 8. Results show consistent patterns: later blocks start with higher confidence and converge faster.

B Additional Experiment Details

B.1 Complete Main Results at Generation Lengths 256 and 512

Table 6 reports the complete Dream-Base results at generation lengths 256 and 512. Streaming-dLLM preserves competitive accuracy across all four benchmarks while delivering 3.7×–13.3× the throughput of the vanilla backbone. At length 512, it also remains faster than the strongest competing acceleration method on every benchmark, with particularly clear gains on GSM8K and MATH.

Table 6: Comparison of Dream-Base suite performance across four benchmarks at different generation lengths (256 and 512). Each cell reports accuracy (top) and decoding throughput with relative speedup over the Dream baseline (bottom; blue: tokens/s, orange: speedup). Results in bold indicate the highest score, while those underlined denote the second-best performance for each method. '*' marks results we reproduced using the official implementation.

Benchmark	Gen Length	Dream	dKV-Cache	Fast-dLLM	Sparse-dLLM	Dpad	Ours
HumanEval (0-shot)	256	49.4	48.2*	**54.3**	49.4*	<u>51.8</u>*	**54.3**
		20.4 (1×)	21.5 (1.1×)	53.7 (2.6×)	46.2 (2.3×)	57.1 (2.8×)	**74.7** (**3.7**×)
	512	<u>54.3</u>	49.4*	54.3	52.4*	52.4*	**54.6**
		13.7 (1×)	15.7 (1.1×)	40.2 (2.9×)	44.3 (3.2×)	52.6 (3.8×)	**72.3** (**5.3**×)
GSM8K-CoT (5-shot)	256	**74.8***	73.6*	73.5*	72.8*	72.5*	<u>74.0</u>
		9.0 (1×)	17.0 (1.9×)	47.9 (5.3×)	42.5 (4.7×)	32.2 (3.6×)	**75.5** (**8.4**×)
	512	<u>74.2</u>*	71.6*	74.1*	72.2*	72.1*	**74.7**
		7.1 (1×)	12.8 (1.8×)	41.7 (5.9×)	40.5 (5.7×)	50.0 (7.0×)	**94.1** (**13.3**×)
MBPP (3-shot)	256	**56.6**	54.0*	<u>56.4</u>	49.0*	54.0*	<u>56.4</u>
		11.0 (1×)	14.7 (1.3×)	67.2 (6.1×)	40.4 (3.7×)	34.7 (3.2×)	**80.2** (**7.3**×)
	512	<u>55.6</u>	53.0*	55.2	51.4*	54.8*	**55.8**
		8.7 (1×)	11.6 (1.3×)	63.1 (7.3×)	38.6 (4.4×)	51.9 (6.0×)	**92.4** (**10.6**×)
MATH (4-shot)	256	**38.4**	36.8*	<u>37.6</u>	35.2*	37.3*	<u>37.6</u>
		10.5 (1×)	14.6 (1.4×)	62.6 (6.0×)	37.8 (3.6×)	33.0 (3.1×)	**78.4** (**7.5**×)
	512	**39.8**	38.5*	39.3	35.7*	39.2*	<u>39.4</u>
		8.6 (1×)	11.6 (1.3×)	54.4 (6.3×)	36.9 (4.3×)	39.9 (4.6×)	**96.0** (**11.2**×)

Table 7 presents the corresponding results for LLaDA-1.5. The proposed method achieves substantial throughput improvements across tasks and generation lengths, reaching a 68.2× speedup on MBPP at length 512, while keeping accuracy comparable to the strongest baselines. These results show that the efficiency gains transfer consistently to a second backbone without requiring model retraining.

Table 8 summarizes the LLaDA-Instruct results. Streaming-dLLM achieves 5.1×–20.6× speedup over the vanilla backbone across the reported settings and generally outperforms Dpad and Sparse-dLLM in both accuracy and throughput. On MBPP, it improves accuracy by 13.8–27.0 points over Fast-dLLM, which is consistent with the benefit of suppressing interference from semantically weak, high-entropy masked tokens in the distant suffix region.

Table 7: Comparison of LLaDA-1.5 suite performance across four benchmarks at different generation lengths (256 and 512). Each cell reports accuracy (top) and decoding throughput with relative speedup over the LLaDA-1.5 baseline (bottom; blue: tokens/s, orange: speedup). Results in bold indicate the highest score, while those underlined denote the second-best performance for each method. '*' marks results we reproduced using the official implementation.

Benchmark	Gen Length	LLaDA-1.5	dKV-Cache	Fast-dLLM	Sparse-dLLM	Dpad	Ours
HumanEval (0-shot)	256	**43.9***	40.2*	37.2*	35.9*	37.2*	39.0
		6.4 (1×)	6.6 (1.0×)	19.1 (3.0×)	10.9 (1.7×)	19.2 (3.0×)	**34.1** (**5.3**×)
	512	**40.5***	40.2*	39.8*	39.0*	39.6*	40.2
		2.9 (1×)	3.3 (1.1×)	13.6 (4.7×)	6.8 (2.3×)	15.6 (5.4×)	**26.7** (**9.2**×)
GSM8K (5-shot)	256	80.5*	80.7*	80.7	80.4*	80.7*	**80.8**
		6.3 (1×)	10.8 (1.7×)	50.0 (7.9×)	26.1 (4.1×)	19.3 (3.1×)	**66.2** (**10.5**×)
	512	81.0*	**81.3***	80.4	81.2*	81.2*	81.2
		2.5 (1×)	4.2 (1.7×)	25.8 (10.3×)	11.2 (4.5×)	19.0 (7.6×)	**69.8** (**28.0**×)
MBPP (3-shot)	256	38.0*	**38.2***	37.6*	37.0*	37.6*	37.8
		2.2 (1×)	3.5 (1.6×)	29.5 (13.4×)	8.0 (3.6×)	9.2 (4.2×)	**54.7** (**24.9**×)
	512	38.2*	38.1*	38.1*	37.8*	38.2*	**38.4**
		0.9 (1×)	1.5 (1.7×)	16.5 (18.3×)	3.3 (3.6×)	13.5 (15.0×)	**61.4** (**68.2**×)
MATH (4-shot)	256	32.7*	31.8*	32.6	33.0*	33.5*	**33.7**
		7.8 (1×)	12.4 (1.6×)	47.1 (6.0×)	27.3 (3.5×)	20.8 (2.7×)	**66.2** (**8.5**×)
	512	**37.1***	35.1*	35.1	33.5*	34.8*	35.1
		4.8 (1×)	7.5 (1.6×)	38.3 (7.9×)	18.3 (3.8×)	21.7 (4.5×)	**62.4** (**13.0**×)

Table 8: Comparison of LLaDA-Instruct suite performance across four benchmarks at different generation lengths (256 and 512). Each cell reports accuracy (top) and decoding throughput with relative speedup over the LLaDA baseline (bottom; blue: tokens/s, orange: speedup). Results in bold indicate the highest score, while those underlined denote the second-best performance for each method. '*' marks results we reproduced using the official implementation.

Benchmark	Gen Length	LLaDA	dKV-Cache	Fast-dLLM	Sparse-dLLM	Dpad	Ours
HumanEval (0-shot)	256	41.5	39.6*	43.3	36.6*	41.5*	**43.4**
		18.4 (1×)	18.9 (1.0×)	71.5 (3.9×)	29.9 (1.6×)	81.0 (4.4×)	**93.3** (**5.1**×)
	512	43.9	**45.1***	44.5	42.1*	43.5*	44.5
		11.9 (1×)	14.3 (1.2×)	52.0 (4.4×)	30.6 (2.6×)	60.4 (5.1×)	**75.8** (**6.4**×)
GSM8K (5-shot)	256	**79.3**	76.5*	78.5	77.0*	78.1*	78.5
		6.6 (1×)	11.6 (1.8×)	50.9 (7.7×)	27.8 (4.2×)	39.3 (6.0×)	**66.2** (**10.0**×)
	512	77.5	77.6*	77.2	76.0*	78.2*	**78.7**
		3.2 (1×)	5.5 (1.7×)	32.1 (10.0×)	15.2 (4.8×)	40.4 (12.7×)	**65.9** (**20.6**×)
MBPP (3-shot)	256	29.4	26.2*	28.2	28.0*	40.4*	**42.0**
		5.9 (1×)	10.0 (1.7×)	44.3 (7.5×)	18.4 (3.1×)	15.4 (2.6×)	**64.2** (**10.9**×)
	512	14.8	14.6*	13.8	13.0*	39.4*	**40.8**
		4.2 (1×)	6.9 (1.6×)	39.7 (9.4×)	16.4 (3.9×)	18.3 (4.4×)	**66.7** (**15.9**×)
MATH (4-shot)	256	**33.5**	33.3*	33.2	32.7*	33.3*	33.3
		8.7 (1×)	14.3 (1.6×)	49.3 (5.7×)	31.6 (3.6×)	22.9 (2.6×)	**71.0** (**8.2**×)
	512	**37.2**	36.3*	36.0	34.6*	35.6*	36.1
		6.2 (1×)	10.1 (1.7×)	43.2 (7.0×)	25.8 (4.2×)	22.6 (3.6×)	**63.2** (**10.2**×)

B.2 Inference Latency Experiments for dLLMs

In addition, we compare Streaming-dLLM and several other baselines in terms of per sample inference latency. This metric better reflects user-perceived responsiveness in practical scenarios, highlighting the suitability of our method for real-time deployment (Xu et al., 2023). As shown in table 9, Streaming-dLLM achieves a 4.8×–21.0× speedup across all benchmarks compared with the vanilla backbone. Relative to the strongest acceleration baseline, it provides an additional 1.3×–3.7× speedup on tasks with a generation length of 512, while maintaining comparable or slightly improved accuracy.

We observe even faster speed gains on LLaDA-1.5, as reported in table 10. In particular, Streaming-dLLM attains a 52.7× speedup on MBPP at generation length 512, markedly exceeding that of Fast-dLLM (Wu et al., 2025b). Consistent improvements are also evident on LLaDA-Instruct of table 11, where our method achieves 5.3×–64.1× speedup across all tasks. By comparison, the second-best method, Fast-dLLM, reaches only 3.9×–9.9× speedup with slightly lower accuracy. We find that the speedup in inference latency differs from that in throughput. A plausible explanation is that the early-exit mechanism prevents all blocks from being fully traversed, which partially limits the improvement in throughput measured. This further indicates that inference latency provides a more faithful reflection of per-sample computational efficiency and real-world responsiveness than throughput alone.

Overall, these results demonstrate that Streaming-dLLM consistently improves inference efficiency across different dLLM backbones and tasks without sacrificing generation quality. The substantial gains in both throughput and per sample latency highlight its practicality for real-world deployment, particularly in latency-sensitive and interactive applications.

Table 9: Comparison of Dream-Base suite performance across four benchmarks at different generation lengths (256 and 512). Each cell reports accuracy (top) and inference latency with relative speedup over the Dream baseline (bottom; green: s, orange: speedup). Results in bold indicate the highest score, while those underlined denote the second-best performance for each method. '*' marks results we reproduced using the official implementation.

Benchmark	Gen Length	Dream	dKV-Cache	Fast-dLLM	Sparse-dLLM	Dpad	Ours
HumanEval (0-shot)	256	49.4	48.2*	**54.3**	49.4*	<u>51.8</u>*	**54.3**
		11.9 (1×)	11.9 (1.0×)	4.6 (2.6×)	5.5 (2.2×)	3.2 (3.7×)	**2.5** (**4.8**×)
	512	<u>54.3</u>	49.4*	54.3	52.4*	52.4*	**54.6**
		34.9 (1×)	32.5 (1.1×)	12.1 (2.9×)	11.5 (3.0×)	4.1 (8.5×)	**3.1** (**11.3**×)
GSM8K-CoT (5-shot)	256	**74.8***	73.6*	73.5*	72.8*	72.5*	<u>74.0</u>
		18.6 (1×)	15.0 (1.2×)	5.3 (3.5×)	6.0 (3.1×)	7.0 (2.7×)	**2.1** (**8.9**×)
	512	<u>74.2</u>*	71.6*	74.1*	72.2*	72.1*	**74.7**
		50.3 (1×)	39.9 (1.3×)	10.7 (4.7×)	12.6 (4.0×)	8.8 (5.7×)	**2.4** (**21.0**×)
MBPP (3-shot)	256	**56.6**	54.0*	<u>56.4</u>	49.0*	54.0*	<u>56.4</u>
		23.3 (1×)	17.4 (1.3×)	3.8 (6.1×)	6.3 (3.7×)	7.4 (3.1×)	**3.3** (**7.1**×)
	512	<u>55.6</u>	53.0*	55.2	51.4*	54.8*	**55.8**
		58.6 (1×)	44.0 (1.3×)	8.1 (7.2×)	13.2 (4.4×)	9.9 (5.9×)	**5.6** (**10.5**×)
MATH (4-shot)	256	**38.4**	36.8*	<u>37.6</u>	35.2*	37.3*	<u>37.6</u>
		24.5 (1×)	17.5 (1.4×)	4.2 (5.8×)	6.8 (3.6×)	7.7 (3.2×)	**3.2** (**7.7**×)
	512	**39.8**	38.5*	39.3	35.7*	39.2*	<u>39.4</u>
		59.3 (1×)	44.1 (1.3×)	9.4 (6.3×)	13.9 (4.3×)	12.7 (4.7×)	**5.2** (**11.4**×)

Table 10: Comparison of LLaDA-1.5 suite performance across four benchmarks at different generation lengths (256 and 512).Each cell reports accuracy (top) and inference latency with relative speedup over the LLaDA-1.5 baseline (bottom; green: s, orange: speedup). Results in bold indicate the highest score, while those underlined denote the second-best performance for each method. '*' marks results we reproduced using the official implementation.

Benchmark	Gen Length	LLaDA-1.5	dKV-Cache	Fast-dLLM	Sparse-dLLM	Dpad	Ours
HumanEval (0-shot)	256	**43.9***	40.2*	37.2*	35.9*	37.2*	39.0
		12.7 (1×)	12.7 (1.0×)	4.2 (3.0×)	7.1 (1.8×)	3.7 (3.4×)	**2.3** (**5.5**×)
	512	**40.5***	40.2*	39.8*	39.0*	39.6*	40.2
		38.2 (1×)	32.4 (1.2×)	7.6 (5.0×)	14.8 (2.6×)	6.5 (5.9×)	**2.9** (**13.2**×)
GSM8K (5-shot)	256	80.5*	80.7*	80.7	80.4*	80.7*	**80.8**
		34.5 (1×)	20.1 (1.7×)	4.4 (7.8×)	8.2 (4.2×)	8.3 (4.2×)	**2.0** (**17.3**×)
	512	81.0*	**81.3***	80.4	81.2*	81.2*	81.2
		81.8 (1×)	48.1 (1.7×)	7.8 (10.5×)	17.3 (4.7×)	8.2 (10.0×)	**2.2** (**37.2**×)
MBPP (3-shot)	256	38.0*	**38.2***	37.6*	37.0*	37.6*	37.8
		28.7 (1×)	17.7 (1.6×)	2.3 (12.5×)	7.9 (3.6×)	7.3 (3.9×)	**1.2** (**23.9**×)
	512	38.2*	38.1*	38.1*	37.8*	38.2*	**38.4**
		68.5 (1×)	43.0 (1.6×)	4.1 (16.7×)	16.3 (4.2×)	4.8 (14.3×)	**1.3** (**52.7**×)
MATH (4-shot)	256	32.7*	31.8*	32.6	33.0*	33.5*	**33.7**
		27.9 (1×)	17.6 (1.6×)	4.7 (5.9×)	7.9 (3.5×)	9.4 (3.0×)	**2.7** (**10.3**×)
	512	**37.1***	35.1*	35.1	33.5*	34.8*	35.1
		66.3 (1×)	42.1 (1.6×)	8.4 (7.9×)	16.7 (4.0×)	10.9 (6.1×)	**4.6** (**14.4**×)

Table 11: Comparison of LLaDA-Instruct suite performance across four benchmarks at different generation lengths (256 and 512).Each cell reports accuracy (top) and inference latency with relative speedup over the LLaDA baseline (bottom; green: s, orange: speedup). Results in bold indicate the highest score, while those underlined denote the second-best performance for each method. '*' marks results we reproduced using the official implementation.

Benchmark	Gen Length	LLaDA	dKV-Cache	Fast-dLLM	Sparse-dLLM	Dpad	Ours
HumanEval (0-shot)	256	41.5	39.6*	43.3	36.6*	41.5*	**43.4**
		13.3 (1×)	12.9 (1.0×)	3.4 (3.9×)	7.0 (1.9×)	3.0 (4.4×)	**2.5** (**5.3**×)
	512	43.9	**45.1***	44.5	42.1*	43.5*	44.5
		39.4 (1×)	32.9 (1.2×)	9.0 (4.4×)	14.7 (2.7×)	6.4 (6.2×)	**5.2** (**7.6**×)
GSM8K (5-shot)	256	**79.3**	76.5*	78.5	77.0*	78.1*	78.5
		35.5 (1×)	20.2 (1.8×)	4.6 (7.7×)	8.3 (4.3×)	4.2 (8.5×)	**2.2** (**16.1**×)
	512	77.5	77.6*	77.2	76.0*	78.2*	**78.7**
		84.1 (1×)	49.0 (1.7×)	8.5 (9.9×)	17.4 (4.8×)	5.7 (14.8×)	**2.3** (**36.6**×)
MBPP (3-shot)	256	29.4	26.2*	28.2	28.0*	40.4*	**42.0**
		29.4 (1×)	17.6 (1.7×)	3.8 (7.7×)	8.0 (3.7×)	5.9 (5.0×)	**0.9** (**32.7**×)
	512	14.8	14.6*	13.8	13.0*	39.4*	**40.8**
		70.5 (1×)	43.3 (1.6×)	7.6 (9.3×)	17.0 (4.1×)	6.0 (11.7×)	**1.1** (**64.1**×)
MATH (4-shot)	256	**33.5**	33.3*	33.2	32.7*	33.3*	33.3
		28.6 (1×)	17.4 (1.6×)	5.1 (5.6×)	7.9 (3.6×)	9.2 (3.1×)	**3.5** (**8.2**×)
	512	**37.2**	36.3*	36.0	34.6*	35.6*	36.1
		68.8 (1×)	42.7 (1.6×)	10.0 (6.9×)	16.4 (4.2×)	16.2 (4.2×)	**6.8** (**10.1**×)

B.3 Optimization and Schedule

We provide detailed configurations for evaluation on HumanEval, GSM8K, MBPP, and MATH, as summarized in Table 12. For each benchmark and backbone (Dream, LLaDA, and LLaDA-1.5), we report the generation length, sliding window size, initial confidence threshold τ_0, decay factor α, and block size used in our experiments. For all models, the block size is set to 32 across datasets, while other hyperparameters vary by benchmark and generation length. These settings are consistently applied for runs reported in the main paper.

For all compared methods, including Vanilla, KV-Cache (Ma et al., 2025a), Fast-dLLM (Wu et al., 2025b), Sparse-dLLM (Song et al., 2026) and Dpad (Chen et al., 2025b), we use the same temperature and block size for a fair comparison. For dKV-Cache, we set the cache refresh interval for dKV-Cache-Decode to 4 on Dream-Base. On LLaDA and LLaDA-1.5, the interval is set to 8, following the original paper.

Table 12: Configurations for different datasets.

Model	Benchmark	Gen Length	sliding window size	τ_0	α	block_size
Dream	HumanEval (0-shot)	256	192	0.9	0.7	32
		512	128	0.9	0.4	32
	GSM8K-CoT (5-shot)	256	32	0.9	0.3	32
		512	32	0.9	0.3	32
	MBPP (3-shot)	256	192	0.9	0.3	32
		512	192	0.9	0.6	32
	MATH (4-shot)	256	32	0.9	0.1	32
		512	32	0.9	0.3	32
LLaDA	HumanEval (0-shot)	256	192	0.9	0.3	32
		512	256	0.9	0.4	32
	GSM8K (5-shot)	256	96	0.9	0.3	32
		512	96	0.9	0.3	32
	MBPP (3-shot)	256	32	0.9	0.3	32
		512	32	0.9	0.3	32
	MATH (4-shot)	256	128	0.9	0.3	32
		512	256	0.9	0.2	32
LLaDA-1.5	HumanEval (0-shot)	256	96	0.9	0.3	32
		512	96	0.9	0.4	32
	GSM8K (5-shot)	256	96	0.9	0.4	32
		512	128	0.9	0.6	32
	MBPP (3-shot)	256	96	0.9	0.3	32
		512	96	0.9	0.3	32
	MATH (4-shot)	256	96	0.9	0.4	32
		512	192	0.9	0.3	32

B.4 Additional Analysis for Prefill and Generation Length.

Additional analysis for prefill length. We investigate the effect of prefill and generation length. As shown in the table 13, increasing the prefill length from 3-shot to 8-shot reduces throughput for all methods. Nevertheless, compared to Fast-dLLM, the speedup of our method increases from $2.1\times$ to $2.9\times$, indicating that the proposed mechanism remains effective at alleviating inference time computational overhead under longer prefill settings.

Additional analysis for generation length. Table 14 reports the accuracy and speedup of LLaDA on GSM8K (5-shot) under different generation lengths (512, 1024, and 2048). For each model, the top row shows the accuracy, while the bottom row reports the throughput (tokens per second) in

blue and the corresponding speedup relative to the vanilla backbone in orange. The results indicate that our method consistently maintains high accuracy across all generation lengths while achieving substantially larger speedup compared with both LLaDA and Fast-dLLM, especially as the generation length increases. As reported in table 15, our method exhibits a pronounced advantage over Fast-dLLM in long context generation. When the generation length reaches 2048 tokens, it achieves a speedup of up to 225.3× relative to the vanilla model. Moreover, across all settings, our approach outperforms Fast-dLLM in generation quality, demonstrating that our acceleration strategy not only improves efficiency but also enables dLLMs to better adapt to block-wise diffusion inference.

Table 13: Impact of few-shots on Accuracy and Speedup for LLaDA-1.5 on GSM8K.

Model	3-shot	5-shot	8-shot
LLaDA-1.5	80.1 3.4 (1×)	81.0 2.5 (1×)	81.7 1.7 (1×)
Fast-dLLM	79.7 36.5 (10.7×)	80.4 25.8 (10.3×)	79.9 20.9 (12.3×)
Ours	80.2 77.4 (22.8×)	81.2 69.8 (28.0×)	81.5 61.3 (36.1×)

Table 14: Impact of generation length on Accuracy and Speedup for LLaDA on GSM8K.

Model	512	1024	2048
LLaDA	77.5 3.2 (1×)	77.6 1.1 (1×)	76.8 0.4 (1×)
Fast-dLLM	77.2 32.1 (10.0×)	75.3 15.0 (13.6×)	76.4 6.5 (16.3×)
Ours	78.7 65.9 (20.6×)	78.3 64.7 (58.8×)	78.7 67.1 (167.8×)

Table 15: Impact of generation length on Accuracy and Speedup for LLaDA-1.5 on GSM8K (5-Shot).

Model	512	1024	2048
LLaDA-1.5	81.0 2.5 (1×)	81.2 0.8 (1×)	81.2 0.3 (1×)
Fast-dLLM	80.4 25.8 (10.3×)	80.5 12.3 (15.4×)	80.7 5.1 (17.0×)
Ours	81.2 69.8 (28.0×)	81.2 64.9 (81.1×)	82.0 67.6 (225.3×)

B.5 Additional Analysis with Existing Confidence-Based Decoding Methods.

We conduct a detailed comparison with existing confidence-based decoding strategies, including fixed-threshold and factor-based methods. The factor-based rule adopted in Fast-dLLM is essentially a local heuristic $(n+1)(1-c^{(n)}) < f$, as it relies solely on token-level confidence at the current decoding step. However, uncertainty in diffusion decoding exhibits a global distribution, rather than being confined to individual tokens.

To address this limitation, our method incorporates the mask ratio as an indicator of the current decoding stage, enabling adaptive thresholding that aligns with the global uncertainty of the generation process. This leads to a clear empirical distinction: factor-based decoding suffers from an accuracy drop of 0.8–2.4 while yielding limited speed improvements, whereas our method achieves a more favorable trade-off, improving throughput while preserving accuracy (see Table 16).

Table 16: Comparison of fixed-threshold (Fixed), factor-based (Factor), and our method on LLaDA-Instruct for GSM8K and MATH. Each entry reports Accuracy (top) and Throughput with speedup (bottom).

Benchmark	Len	Fixed	Factor	Streaming (Ours)
GSM8K (5-shot)	256	**78.5** 50.9 (7.7×)	77.5 62.3 (9.5×)	**78.5** **66.2** (**10.0**×)
	512	77.2 32.1 (10.0×)	74.8 50.1 (15.6×)	**78.7** **65.9** (**20.6**×)
MATH (4-shot)	256	33.2 49.3 (5.7×)	32.0 68.3 (7.9×)	**33.3** **71.0** (**8.2**×)
	512	36.0 43.2 (7.0×)	35.2 53.8 (8.7×)	**36.1** **63.2** (**10.2**×)

B.6 Additional Comparisons with Recent dLLM Methods.

We further compare our approach with several stronger and representative dLLM-based acceleration methods, including block-wise diffusion models such as Fast-dLLM v2 (Wu et al., 2025a), system-level inference frameworks such as dInfer (Ma et al., 2025b), and suffix caching methods such as dual-cache (Wu et al., 2025b). As shown in Table 17, our method consistently achieves comparable or better accuracy while delivering significantly higher throughput, with up to 3×–6× speedup in long-sequence settings (e.g., 67.6 vs. 22.1 and 11.3).

It is worth noting that these methods differ substantially in design philosophy and practical requirements. Fast-dLLM v2 relies on approximately 1B-token finetuning to obtain an optimized diffusion model, whereas our method is entirely training-free and can be directly applied to existing pretrained dLLMs. dInfer primarily focuses on system-level optimizations, such as compilation and parallel execution, with limited algorithmic modifications. In contrast, our approach introduces algorithm-level improvements, including approximate suffix modeling and dynamic confidence-aware decoding, enabling better modeling of global uncertainty during generation.

For suffix caching methods such as dual-cache, while they can reduce redundant computation, they suffer from two inherent limitations: (1) due to bidirectional attention in dLLMs, cached suffix states must be frequently updated, otherwise leading to noticeable accuracy degradation; (2) in long-sequence generation, they still incur substantial recomputation overhead and high memory consumption. In contrast, our method achieves reduced suffix memory complexity while maintaining stable accuracy, resulting in more scalable and efficient inference.

Overall, these results highlight that our method provides a more favorable trade-off between efficiency and accuracy without requiring additional training or memory overhead. Moreover, our approach is orthogonal to existing methods and can be seamlessly integrated with them. We are actively exploring such combinations (e.g., with Fast-dLLM v2 and dInfer), and will explore these direction in the future work.

Table 17: Comparison of different acceleration methods on GSM8K (5-shot) for LLaDA-1.5. Each entry reports Accuracy (top) and Throughput (bottom).

Benchmark	Len	dInfer	Fast-dLLM v2	Dual-Cache	Ours
GSM8K (5-shot)	256	79.5 60.1	79.8 54.5	79.8 54.5	**80.8** **66.2**
	512	81.0 45.3	81.1 41.7	79.5 38.7	**81.2** **69.8**
	2048	81.8 17.7	82.0 22.1	78.8 11.3	**82.0** **67.6**

B.7 Additional Analysis on Accuracy-Throughput Trade-off

We provide additional analysis on the accuracy-throughput trade-off across different benchmarks and model variants. While our method is designed to improve decoding efficiency, the trade-off between accuracy and speed is inherently benchmark-dependent.

Overall, compared to Fast-dLLM, our method consistently achieves comparable or improved accuracy while delivering substantial throughput gains. In some cases, slight accuracy variations can be observed relative to the vanilla model. This behavior is largely related to the decoding mechanism: Fast-dLLM-style approaches rely on prefix KV-cache, which cannot fully capture the dynamic and iterative nature of diffusion-based decoding. Our method alleviates this limitation through stage-aware confidence modeling and adaptive decoding strategies. With appropriate hyperparameter configurations, it is able to recover or even surpass the vanilla accuracy, while maintaining significantly higher throughput.

Table 18 summarizes the trade-off across representative benchmarks. As shown, our method consistently achieves a better balance between accuracy and efficiency. In particular, it matches or exceeds vanilla performance on several tasks (e.g., HumanEval and MATH), while providing substantial speedups over both vanilla and Fast-dLLM. These results indicate that our method provides a flexible and effective mechanism for balancing accuracy and efficiency, achieving near-lossless performance while significantly improving decoding throughput.

Table 18: Accuracy–throughput trade-off across different models and benchmarks. Each entry reports Accuracy (top) and Throughput (bottom).

Model	Benchmark	Vanilla	Fast-dLLM	Ours
LLaDA	GSM8K-256	79.3 6.6	78.5 50.9	**79.5** **63.4**
	MATH-512	37.2 6.2	36.0 43.2	**37.5** **51.0**
LLaDA-1.5	HumanEval-256	43.9 6.4	37.2 19.1	**44.5** **25.8**
	MATH-512	37.1 4.8	35.1 38.3	**37.2** **54.4**

B.8 Statistical Stability Analysis

To further evaluate the statistical reliability of our method, we conduct additional experiments across three different random seeds and report the mean and standard deviation of both accuracy and throughput.

Results. As summarized in Table 19, our method demonstrates strong stability across all evaluated benchmarks. The variance in accuracy remains consistently low (standard deviation ≤ 0.1), while the variance in throughput ranges from 0.2 to 2.4, indicating limited sensitivity to randomness. These results suggest that the proposed method is robust and produces consistent performance under different initialization conditions.

Discussion. We note that the primary goal of our method is not marginal accuracy improvements, but rather achieving substantial inference acceleration while maintaining comparable performance to strong baselines such as Fast-dLLM. This is particularly evident in long-sequence settings, where our method consistently delivers significant speedups (e.g., $5.5\times$–$13.3\times$) while preserving accuracy.

B.9 Additional Analysis for Trailing Positional Information

Discussion on the necessity of retaining trailing positional information is summarized in table 20. It can be observed that omitting the trailing positional information leads to a noticeable drop in generation accuracy. Remarkably, this outcome is both intuitive and theoretically justified. When trailing positional information is not preserved, the model theoretical performance depends mainly on the

Table 19: Results across three random seeds for Streaming-dLLM. Its entries report mean ± standard deviation, while the vanilla and Fast-dLLM columns show reference point estimates. Each setting reports accuracy (top) and throughput (bottom).

Benchmark	Len	LLaDA-1.5	Fast-dLLM	Ours
HumanEval (0-shot)	256	43.9 6.4	37.2 19.1	39.0±0.0 34.5±0.4
	512	40.5 2.9	39.8 13.6	40.2±0.0 26.3±0.5
GSM8K (5-shot)	256	80.5 6.3	80.7 50.0	80.8±0.1 65.5±0.7
	512	81.0 2.5	80.4 25.8	81.3±0.1 68.9±0.9
MBPP (3-shot)	256	38.0 2.2	37.6 29.5	37.8±0.0 56.2±1.5
	512	38.2 0.9	38.1 16.5	38.4±0.0 63.8±2.4
MATH (4-shot)	256	32.7 7.8	32.6 47.1	33.8±0.1 65.9±0.3
	512	37.1 4.8	35.1 38.3	35.1±0.0 62.6±0.2

block size and the number of retained suffix blocks, rather than the overall generation length. In practice, existing dLLMs typically rely on entire sequence positional information for global modeling, yet the experimental results suggest that this is not strictly required, as a small portion of the suffix combined with the final positional information can approximate the overall behavior. This finding provides new insight into enabling dLLMs to strike a more suitable balance between autoregressive and pure diffusion generation paradigms.

Table 20: Impact of trailing positional information.

Model	Trailing Position	Accuracy (%)
Dream	×	72.8
	✓	**74.7**
LLaDA	×	77.5
	✓	**78.7**
LLaDA-1.5	×	79.6
	✓	**81.2**

C Others

C.1 Social Impact Analysis

This work aims to advance the field of machine learning by improving the efficiency of diffusion large language models (dLLMs), which may enable broader deployment and reduce energy consumption. At the same time, increased efficiency could lower the barrier to misuse, a concern that is widely recognized in generative modeling research. While our method does not introduce new risks beyond these general considerations, we emphasize the importance of responsible use and recommend further evaluation of safety, robustness, and potential misuse scenarios before deployment in sensitive applications.

C.2 Limitation

Our method employs suffix pruning guided by an estimated target length, which is effective when the generation length is reasonably bounded. In fully open-ended scenarios where such length cannot

be reliably specified, this mechanism may lead to suboptimal context approximation and potential degradation in long-range coherence.

We emphasize that this limitation is not specific to our approach, but is shared by a broader class of diffusion language model (dLLM) decoding frameworks that rely on length-related assumptions. Our method does not introduce additional dependency on target length, nor does it amplify this issue, but instead improves decoding efficiency under the same setting. Addressing open-ended generation without explicit length signals remains an important direction for future work.