SEMANTIC-AWARE DIFFUSION LLM INFERENCE WITH ADAPTIVE BLOCK SIZE

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion-based large language models (dLLMs) are gaining attention for their inherent capacity for parallel decoding, offering a compelling alternative to autoregressive LLMs. Among various decoding strategies, blockwise semi-autoregressive (semi-AR) approaches are widely adopted due to their natural support for KV caching and their favorable accuracy–speed trade-off. However, this paper identifies two fundamental limitations in the conventional semi-AR decoding approach that applies a fixed block size: *i)* late decoding overhead, where the unmasking of high-confidence tokens outside the current block is unnecessarily delayed; and *ii)* premature decoding error, where low-confidence tokens inside the current block are committed too early, leading to incorrect tokens. This paper presents the first systematic investigation challenging the fixed block size assumption in semi-AR decoding. Through a statistical analysis of confidence dynamics during the denoising process, we identify a volatility band (VB) region during dLLM decoding, which encodes local semantic structure and can be used to guide adaptive block sizing. Leveraging these insights, we introduce *AdaBlock-dLLM*, a training-free, plug-and-play scheduler that adaptively aligns block boundaries with semantic steps by adjusting block size during runtime. Extensive experiments across diverse benchmarks show that *AdaBlock-dLLM* achieves up to 5.3% accuracy improvement under the same throughput budget. Beyond inference-time optimization, we hope our semantics-aware adaptive scheduling approach and confidence-based analysis will inspire future training strategies for dLLMs.

1 INTRODUCTION

Diffusion-based Large Language models (dLLMs) have recently emerged as a promising alternative to autoregressive models, offering parallel decoding, improved controllability, and greater data efficiency in low-resource settings (Zhang et al., 2025; Prabhudesai et al., 2025). Open-source dLLMs such as LLaDA (Nie et al., 2025; Zhu et al., 2025) and Dream (Ye et al., 2025) have shown comparable performance to autoregressive models of similar scale. Notably, in structured generation tasks such as coding, proprietary models including Seed Diffusion (Song et al., 2025b) and Gemini Diffusion (Gemini Diffusion, 2025) demonstrate throughput exceeding $1{,}400$ tokens per second. These advances highlight the potential of dLLMs to deliver efficient inference while maintaining competitive algorithmic performance.

Recent works have widely adopted a semi-autoregressive (semi-AR) decoding paradigm that combines blockwise KV caching (Wu et al., 2025; Chen et al., 2025; Song et al., 2025a) and confidence-based dynamic sampling (Wang et al., 2025d; Wu et al., 2025; Wei et al., 2025) to improve inference efficiency. However, semi-AR decoding enforces block-level causality: the current block must be finalized before decoding the next block. We take the first attempt to identify two **fundamental issues** introduced by conventional semi-AR decoding with a fixed block size: ***i)* Late Decoding Overhead.** As shown in the upper-left of Figure 1, semi-AR decoding delays the unmasking of high-confidence tokens outside the current block. For instance, the second and third blocks are decoded in separate iterations, incurring unnecessary computational overhead to generate a simple complete sentence. ***ii)* Premature Decoding Error.** As shown in the lower-left of Figure 1, the autoregressiveness across blocks forces early commitment to low-confidence tokens within each block, and this suboptimal sampling often yields incorrect token predictions (Figure 5), particularly in reasoning tasks.

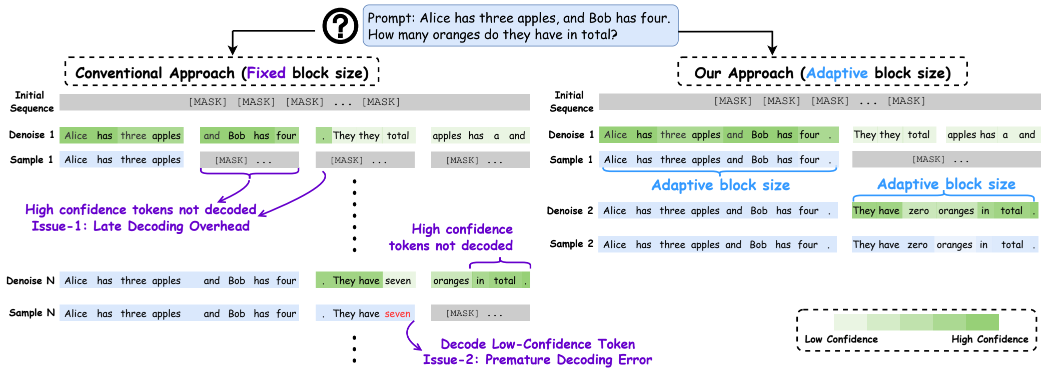

Prompt: Alice has three apples, and Bob has four. How many oranges do they have in total?
Conventional Approach (Fixed block size)
Our Approach (Adaptive block size)
Initial Sequence
[MASK] [MASK] [MASK] ... [MASK]
Denoise 1
Sample 1
High confidence tokens not decoded
Issue-1: Late Decoding Overhead
High confidence tokens not decoded
Denoise N
Sample N
Decode Low-Confidence Token
Issue-2: Premature Decoding Error
Adaptive block size
Denoise 2
Sample 2
Low Confidence
High Confidence

Figure 1: Illustrative examples of two key issues (left) and how they can be overcome with *AdaBlock-dLLM* (right). A real case study is provided in Appendix A.1.

To address these limitations, we first investigate how confidence scores evolve during dLLM denoising and sampling. Our statistical analysis in Section 4.1 reveals a volatility band (VB), a region where confidence fluctuates dynamically. VB regions exhibit semantics structure, which can be exploited to dynamically adapt block size during runtime. Based on these insights and observations, we propose an adaptive block-size decoding method for dLLM inference, termed *AdaBlock-dLLM*, which adopts a semantic-aware approach that adaptively adjusts block boundaries (Figure 1, right). Specifically, *AdaBlock-dLLM* dynamically aligns block size with the length of semantic blocks, as categorized by special semantic tokens (e.g., periods and \n), enabling dLLM to perform efficient decoding while mitigating the limitations of fixed-size approaches. Importantly, *AdaBlock-dLLM* is a training-free and plug-and-play enhancement to the existing semi-AR decoding paradigm. Across comprehensive experiments on various benchmarks, we demonstrate that *AdaBlock-dLLM* improves accuracy by up to 5.3% while achieving throughput comparable to prior dLLM acceleration methods (Figure 2). Gains are especially pronounced under KV caching, where fixed block sizes further compromise semantic consistency. Our results motivate semantics-aware training objectives for block-diffusion models that emphasize context preservation.

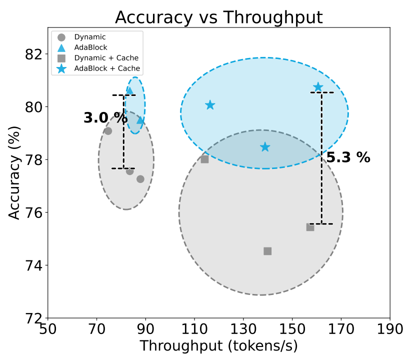


Figure 2: Performance improvement over Fast-dLLM (Wu et al., 2025).

In summary, our contributions are threefold:

- We systematically analyze the semi-autoregressive sampling paradigm, and identify the inaccuracy and inefficiency behind fixed block size settings (Section 4.1 and Section 4.2).
- We propose *AdaBlock-dLLM*, a training-free, plug-and-play technique that enhances existing semi-autoregressive decoding paradigm, which dynamically adjusts block sizes based on the confidence of semantic delimiter tokens (Section 4.3).
- We conduct extensive experiments demonstrating that *AdaBlock-dLLM* improves accuracy by up to 5.3% over state-of-the-art methods under the same speed budget (Section 5).

2 Related Works

2.1 Diffusion Language Models

Diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2021; Karras et al., 2022) have achieved high-fidelity generation across continuous data domains, including images and video (Peebles & Xie, 2023; Ho et al., 2022). Motivated by this success, a growing line of work adapts diffusion to NLP tasks, giving rise to masked diffusion models (MDMs) that iteratively denoise an initially masked token sequence into coherent output (Austin et al., 2021a; Hoogeboom et al., 2021; Lou et al., 2024). Recent efforts have scaled MDMs up to 8B parameters (Nie et al., 2025; Ye et al., 2025), highlighting their robustness and scalability. Current diffusion language models can be categorized into *i)* models trained from scratch (Nie et al., 2025; Yang et al., 2025); *ii)* models adapted from autoregressive (AR) models (Ye et al., 2025); and *iii)* block-diffusion models (Cheng et al., 2025; Wang et al., 2025d), which combine the training efficiency of AR models with the sampling efficiency of diffusion models.

2.2 Efficient Inference for Diffusion LLMs

Recent work has advanced inference for diffusion LLMs (dLLMs) in both speed and accuracy. In inference acceleration, research has focused on two directions: caching mechanisms and parallel decoding. Multiple caching mechanisms have been proposed to avoid recomputing key–value (KV) pairs at each denoising step, including delayed caching (Ma et al., 2025), verification-based caching (Liu et al., 2025), and block-level caching (Wu et al., 2025). Advances in parallel decoding include efficient threshold-based dynamic sampling (Wu et al., 2025; Wang et al., 2025c; Wei et al., 2025; Yu et al., 2025), sampler scheduling (Luxembourg et al., 2025), and guided diffusion (Hu et al., 2025; Israel et al., 2025). To improve accuracy, test-time strategies such as voting (Wang et al., 2025b), early committing (Li et al., 2025a) and remasking (Hong et al., 2025; Wang et al., 2025a; He et al., 2025) have also been explored.

2.3 Blockwise Semi-Autoregressive Decoding

Semi-autoregressive (semi-AR) decoding partitions the sequence into blocks. Decoding is autoregressive at the block level, but non-autoregressive within blocks, allowing tokens inside the block to be sampled in arbitrary order. This paradigm was first introduced in Block Diffusion (Arriola et al., 2025), which interpolates between autoregressive and fully diffusion-based decoding by applying a block-causal attention mask. Various diffusion LLMs, including LLaDA (Nie et al., 2025) and MMaDA (Yang et al., 2025), adopt semi-AR decoding; however, they predefine a fixed generation budget. Compared with random-order decoding over the full sequence, a key advantage of semi-AR decoding is that it naturally supports block-level KV caching (Wu et al., 2025; Chen et al., 2025; Song et al., 2025a), making it a prevalent scheme in recent dLLM research. To the best of our knowledge, prior semi-AR decoding uses a **fixed** block size. In contrast, this paper takes the first attempt to explore **adaptive** block-size decoding with **semantic-aware**, **training-free** method.

3 Preliminares

The decoding process of diffusion LLMs comprises two operations: *denoise* and *sample*. In text generation, the decoding process starts from a fully masked sequence, and the model iteratively employs the denoise–sample cycle until no mask token is left. Taking LLaDA (Nie et al., 2025) as an example, we formalize the decoding process.

Setup. Let $\mathcal{V}$ denote the vocabulary, which includes a special mask token $[\texttt{MASK}] \in \mathcal{V}$. Given a prompt $\mathbf{q} = (q_0, \dots q_{L_p-1}) \in \mathcal{V}^{L_p}$ and a generation budget L, define the index set $\mathcal{J} \triangleq \{0, 1, \dots, L_p{+}L{-}1\}$. Let T be the total number of denoise–sample iterations. At step $t \in \{T, T{-}1, \dots, 0\}$, the sequence state is $\mathbf{y}^{\,t} = (y_i^t)_{i \in \mathcal{J}} \in \mathcal{V}^{L_p+L}$. The initial state of the sequence is $\mathbf{y}^{\,T} = (q_0, \dots, q_{L_p-1}, \underbrace{[\texttt{MASK}], \dots, [\texttt{MASK}]}_{L \text{ times}})$.

Denoise. A mask predictor p_θ predicts a sequence $\hat{\mathbf{y}}^t \in (\mathcal{V} \setminus [\texttt{MASK}])^{L_p+L}$ using greedy decoding:

$$\hat{y}_i^t \;=\; \arg\max_{v \in \mathcal{V}} p_\theta\big(v \mid \mathbf{y}^{\,t}, i\big)\,, \qquad i \in \mathcal{J}. \tag{1}$$

Sample. Define the masked-position set:

$$\mathcal{M}_t \triangleq \{\, i \in \mathcal{J} : y_i^t = [\texttt{MASK}] \,\}. \tag{2}$$

A sampler selects $S_t \subseteq \mathcal{M}_t$ to unmask. For all $i \in \mathcal{J}$, update

$$y_i^{t-1} = \begin{cases} \hat{y}_i^{\,t}, & i \in S_t \quad \text{(unmask)}, \\ \texttt{[MASK]}, & i \in \mathcal{M}_t \setminus S_t \quad \text{(stay masked)}, \\ y_i^{t}, & i \notin \mathcal{M}_t \quad \text{(already unmasked; keep)}. \end{cases} \tag{3}$$

Then, the sequence state becomes

$$\mathbf{y}^{\,t-1} = (y_i^{t-1})_{i \in \mathcal{J}} \in \mathcal{V}^{L_p+L}. \tag{4}$$

Repeat for $t = T, T-1, \dots, 1$; terminate when $\mathcal{M}_t = \varnothing$.

Sampling Methods. LLaDA (Nie et al., 2025) proposes a linear-schedule sampler that unmasks a fixed number of tokens L/T at each step, either at random or by confidence. LaViDa (Li et al., 2025b) applies a shifting schedule that unmasks a variable number of [MASK] positions in each step. Fast-dLLM (Wu et al., 2025) and Dimple (Yu et al., 2025) adopt dynamic sampling by introducing a confidence threshold τ. At each step t, the model unmask all positions with confidence $c_i^t \triangleq p_\theta(\hat{y}_i^{\,t} \mid \mathbf{y}^{\,t}, i)$ that $c_i^t \geq \tau$. Compared to the linear-schedule or the shifting schedule, dynamic sampling adapts to token-level uncertainty and improves the accuracy–throughput balance.

4 METHODOLOGY

Semi-AR decoding has been widely adopted in dLLMs, including LLaDA (Nie et al., 2025) and MMaDA (Yang et al., 2025). This decoding paradigm naturally supports a block-level KV cache, addressing a key bottleneck in dLLM inference. Additionally, combining semi-AR decoding with dynamic sampling (Wu et al., 2025) enables multi-token prediction within a single denoise—sample cycle (Wu et al., 2025; Wang et al., 2025c). This paradigm introduces two hyperparameters: **confidence threshold** τ and **block size** B. The confidence threshold maintains a balance between sampling speed and quality, whereas the effect of block size has not been systematically explored.

In Section 4.1, we analyze the confidence dynamics that govern dynamic sampling. Using these patterns, we characterize the inaccuracies and inefficiencies caused by the misalignment between a fixed block size and the model's inductive decoding preferences in Section 4.2. This motivates an adaptive block-size scheduler that minimizes this mismatch.

4.1 ANALYSIS OF CONFIDENCE DYNAMICS

Confidence Scores. Confidence scores are a key metric in dynamic sampling of dLLM. A high confidence score indicates that the model is more certain about the prediction (Wei et al., 2025). The decoding process consists of iterative denoise–sample cycles (Section 3). At each denoise step, the model predicts the tokens in all masked positions via greedy decoding. Afterwards, tokens with confidence $c_i \geq \tau$ are unmasked, where τ is a hyperparameter specifying the confidence threshold.

Confidence Dynamics and Global Autoregressiveness. Figure 3 visualizes confidence dynamics at early, middle, and late decoding stages of LLaDA-8B-Base inference on GSM8K Benchmark. Based on these statistical analyses, we summarize the following observations and patterns:

- For all stages, a high-confidence region emerges near the decoded (or prompt) tokens. This indicates that masked positions adjacent to decoded (high-confidence) tokens are more likely to attain high confidence and thus to be decoded. We attribute this to **confidence locality**: dLLMs exhibit higher confidence in regions where local semantic meaning is complete.
- Although the model may generate in arbitrary order, its decoding trace shows a **global autoregressive tendency**. This pattern suggests a chain-like progression, which arises from semantic causality where subsequent predictions depend on prior semantics.
- As the decoding process unfolds, confidence for decoded tokens remains high, and the high-confidence region extends toward adjacent positions. In contrast, positions outside this region maintain consistently low confidence, forming what we term the **low-confidence floor**.

Volatility Band and Local Stochasticity. Building on these observations, we partition the position-wise confidence landscape into three regimes: *i)* a high-confidence plateau with consistently large scores; *ii)* a volatility band (VB) characterized by unstable, variable scores; and *iii)* a low-confidence floor with persistently small scores. Among these regimes, the VB is the key region where the current decoding steps take place. As illustrated in Figure 4, scores within the VB fluctuate over decoding steps (time) and across positions (space), yet remain clearly separated from the low-confidence floor. Additionally, the width of the VB varies from case to case. In contrast to the left-to-right expansion of the high-confidence plateau that is driven by global autoregressive causality, the decoding order within the VB is locally stochastic: the positional preference diminishes, and the sampling choice depends more heavily on the immediate semantic context.

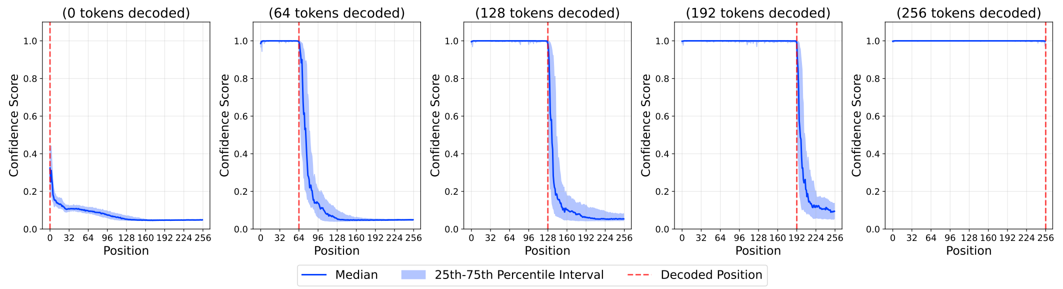


Figure 3: Confidence scores across sequence positions for LLaDA-8B-Base, evaluated on 100 samples from the GSM8K benchmark. The high confidence plateau expands as decoding progresses, while positions beyond the decoded prefix exhibit high variance.

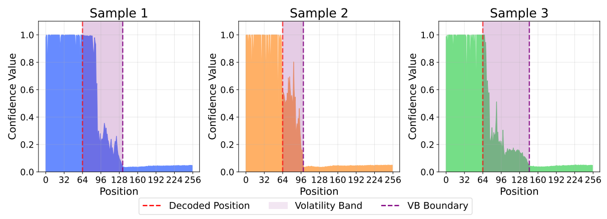


Figure 4: Illustration of the high confidence plateau, the volatility band (VB), and the low confidence floor across three samples. Within VB, the distribution of confidence scores and the width of the band vary across samples.

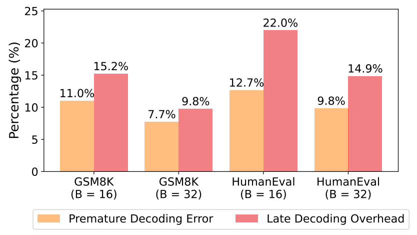


Figure 5: Proportion of sampling steps affected by late decoding overhead and premature decoding error on GSM8K and HumanEval for fixed block sizes.

4.2 Motivation for Adaptive Block Size.

Despite the local stochasticity exhibited during decoding, prevailing semi-AR decoding strategies use manually set, fixed block sizes that often fail to capture this stochasticity. This misalignment leads to **Late Decoding Overhead** and **Premature Decoding Error**, as illustrated in Figure 1, thereby reducing accuracy and efficiency. This motivates incorporating an adaptive block size.

Late Decoding Overhead. A fixed block size poses a hard constraint on number of tokens that can be sampled in a single step. By fixing the block size, the sampler must exclude nearby higher-confidence positions outside the block, especially when a small block size is used (Wang et al., 2025d; Cheng et al., 2025). Deferred high-confidence positions undergo additional denoising in later iterations, incurring unnecessary computation overhead and resulting in degraded throughput.

Premature Decoding Error. With a fixed block size, the sampler (Algorithm 3) must decode all masked positions in the current block before advancing. This constraint forces the sampler to decode low-confidence positions inside the block, instead of the high-confidence positions outside the block. The result is systematic inaccuracy: decoding premature tokens increases token-level error rates, propagates mismatches to subsequent steps via block-level autoregressive dependencies, and biases the hypothesis toward poorly calibrated regions of the confidence landscape. This usually contributes to degraded accuracy.

4.3 Semantic-Aware Adaptive Block-Size Scheduler

Challenge for predicting block size. The VB characterizes the region of active decoding: positions before the VB are typically decoded and exhibit consistently dynamic confidence. As shown in Figure 3, the VB boundary encodes semantic structure due to the confidence locality (Section 4.1). In contrast, tokens in the low-confidence floor are repeatedly assigned placeholders or formatting symbols (e.g., `<EOS>`, spaces).

Although the VB region contains semantic information, it is often too wide. As shown in Figure 7, the token "GB" repeatedly appears with confidence scores mostly between 0.1 and 0.3, lying within the VB. Although such tokens are contextually related, their significance to the current context is weak, providing little guidance for the current sampling step. Consequently, VB regions often provide limited actionable signals for local decisions, requiring a fine-grained segmentation to identify tokens that relate closely to the current decoding step.

Align Block Size With Semantic Steps. To obtain a fine-grained segmentation that reflects the context of the current decoding step, we partition the sequence into **semantic steps**, which are contiguous spans whose provisional tokens exhibit local semantic coherence. We then couple the scheduler to the semantic step, setting the block size guided by the current semantic step length. This allows the sampler to finalize higher-confidence positions within the step while deferring lower-confidence positions until the semantic step is ready to close. Across semantic steps, dependencies are enforced by the semi-AR paradigm, since each downstream step conditions on completed predecessors. This alignment curbs premature commitments outside the active step and prevents splitting a step across iterations, thereby reducing both error propagation and computational overhead.

Algorithm 1 Semantic-Aware Block Size Determination

```
    Inputs: predicted sequence $\hat{\mathbf{y}}$; confidences $\mathbf{c}$; generation budget $L$; default block size $B_0$;
    delimiter set $\mathcal{D}$; delimiter threshold $\tau_{\mathcal{D}}$; current position $g$.
    Output: block size $B$
 1: function ComputeBlockLength($\hat{\mathbf{y}}, \mathbf{c}, L, B_0, \mathcal{D}, \tau_{\mathcal{D}}, g$)
 2:     ▷ Sampling window boundary
 3:     $start,\ remaining \leftarrow g,\ L - g$
 4:     $w \leftarrow \min\big(\max(1, \lfloor 0.25 \cdot g \rfloor),\ remaining\big)$
 5:     $W \leftarrow \{\, start, \ldots, start + w - 1 \,\}$                ▷ window token indices
 6:     ▷ Find highest-confidence delimiter
 7:     $\mathcal{I} \leftarrow \{\, i \in W \mid \hat{y}_i \in \mathcal{D} \,\}$
 8:     if $\mathcal{I} \neq \emptyset$ then
 9:         $pos \leftarrow \arg\max_{i \in \mathcal{I}} c_i$        ▷ Select position with max delimiter token confidence
10:         $c_{\max} \leftarrow c_{pos}$
11:     else
12:         $c_{\max} \leftarrow -\infty$
13:     end if
14:     ▷ Determine block size
15:     if $c_{\max} \geq \tau_{\mathcal{D}}$ then
16:         $B \leftarrow (pos - start + 1)$                ▷ inclusive length up to the delimiter
17:     else
18:         $B \leftarrow \min(B_0,\ remaining)$
19:     end if
20:     return $B$
21: end function
```

Semantic-Aware Block Size Scheduling With *AdaBlock-dLLM*. To facilitate the dynamic adjustment of block size B, we insert an additional block-size determination procedure between denoising and sampling at the start of each block (Algorithm 1). Given the current predicted sequence $\hat{\mathbf{y}}$ and confidence scores $\{c_i\}$, we collect indices whose predicted tokens fall in the delimiter set $\mathcal{D}$ (line 7). Among these, we choose the delimiter $\hat{y}_{\max}$ with the highest confidence $c_{\max}$. If $c_{\max} \geq \tau_D$, we set B to the position of $\hat{y}_{\max}$ (lines 15–16), indicating that the model has a reliable preference for the end of the current semantic step. If no delimiters appear in $\hat{\mathbf{y}}$ within W_t

(lines 11–12), or if all delimiter predictions are low-confidence ($c_{\max} < \tau_D$, lines 17–18), we fall back to the default block size. Additionally, we apply an index window W that masks distant positions to avoid decoding the `<EOS>` token in the early stage, a cause for severe performance drop (Nie et al., 2025). This procedure yields step-aware blocks when evidence is strong, while remaining conservative in ambiguous regions.

5 EXPERIMENT

5.1 EXPERIMENTAL SETUP

Implementation Details. We evaluate *AdaBlock-dLLM* on representative diffusion LLMs (dLLMs): LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2025), and Dream-v0-Base-7B (Ye et al., 2025). Unless otherwise noted, all experiments run on NVIDIA H100 GPUs.

Hyperparameters Settings. We use the generation budget $L = 512$ for all benchmarks. For dynamic sampling, we use a confidence threshold $\tau = 0.9$. We sweep default block sizes $B \in \{16, 32, 64\}$. *AdaBlock-dLLM* introduces two hyperparameters: the delimiter set $\mathcal{D}$ and the delimiter confidence threshold τ_D. We set $\mathcal{D} = \{\backslash\mathtt{n}\}$, since newline tokens commonly mark the end of reasoning steps in test-time search (Snell et al., 2024) style prompting. We use $\tau_D = 0.3$ for LLaDA-series models and $\tau_D = 0.5$ in Dream-series models. This is tuned on a small subset of the GSM8K benchmark, and the selection is discussed in Section 5.3.

Benchmarks and Metrics. We evaluate *AdaBlock-dLLM* on standard LLM benchmarks. For math reasoning, we use GSM8K (5-shot) (Cobbe et al., 2021) and MATH (4-shot) (Hendrycks et al., 2021). For code generation, we use HumanEval (0-shot) (Chen et al., 2021) and MBPP (3-shot) (Austin et al., 2021b). Generation quality is measured by accuracy: pass@1 for code generation and answer accuracy for math reasoning. All accuracy values are reported as percentages. We evaluate accuracy across five sampling methods in total, including three baselines based on Fast-dLLM (Wu et al., 2025): **Vanilla** (top-1 confidence sampling), **Dynamic** (dynamic sampling), and **+Cache** (dynamic sampling with DualCache); and two variants enhanced with *AdaBlock-dLLM*: **+Ada** (dynamic sampling with *AdaBlock-dLLM*) and **+Ada+Cache** (dynamic sampling with DualCache and *AdaBlock-dLLM*).

5.2 MAIN RESULTS

Generation Quality Across Models. Table 1 reports accuracy to quantify generation quality. *AdaBlock-dLLM* achieves accuracy gains across most model and dataset pairs. Notably, on GSM8K with LLaDA-Instruct, accuracy is improved by 3.0% without caching and 5.3% with caching.

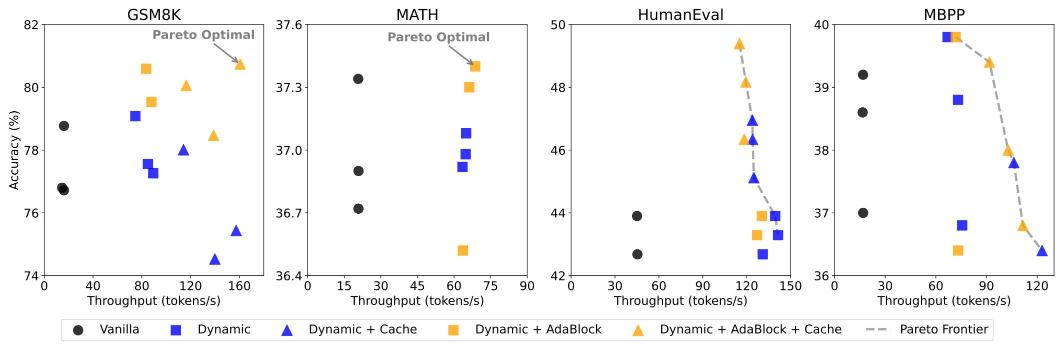


Figure 6: Accuracy and throughput of different sampling methods evaluated on LLaDA-Instruct. Integrating *AdaBlock-dLLM* into Fast-dLLM (Wu et al., 2025) yields accuracy gains across all benchmarks while maintaining little throughput overhead. Notably, Fast-dLLM with *AdaBlock-dLLM* is Pareto-optimal on the GSM8K and MATH datasets.

Table 1: Accuracy (%) across sampling methods, evaluated on LLaDA-1.5, LLaDA-Instruct, and Dream-Base under default block sizes $B_0 \in \{16, 32, 64\}$. Differences are shown in gray. Comparisons are reported relative to **Dynamic** and **+Cache** (Wu et al., 2025).

Method	LLaDA-Instruct			LLaDA-1.5			Dream-Base		
	$B_0 = 16$	$B_0 = 32$	$B_0 = 64$	$B_0 = 16$	$B_0 = 32$	$B_0 = 64$	$B_0 = 16$	$B_0 = 32$	$B_0 = 64$
GSM8K									
Vanilla	78.8	76.7	76.8	82.3	82.3	80.4	76.3	76.4	75.1
Dynamic	79.1	77.6	77.3	82.6	82.2	80.7	75.5	75.5	75.6
+Ada	80.6 +1.5	80.6 +3.0	79.5 +2.2	83.0 +0.4	82.4 +0.2	80.3 -0.4	75.7 +0.2	75.7 +0.2	75.9 +0.3
+Cache	78.0	74.5	75.4	80.7	80.2	80.0	75.6	74.5	74.6
+Ada+Cache	80.0 +2.0	78.5 +4.0	80.7 +5.3	81.3 +0.6	81.7 +1.5	79.7 -0.3	76.5 +0.9	75.1 +0.6	74.6 +0.0
HumanEval									
Vanilla	43.9	43.9	42.7	39.0	36.6	38.4	53.7	52.4	54.3
Dynamic	42.7	43.9	43.3	36.6	37.8	36.6	53.0	51.2	52.4
+Ada	43.3 +0.6	43.3 -0.6	43.9 +0.6	37.8 +1.2	38.4 +0.6	38.4 +1.8	53.7 +0.7	51.2 +0.0	53.7 +1.3
+Cache	45.1	46.3	47.0	33.5	36.0	34.1	50.0	53.0	56.1
+Ada+Cache	49.4 +4.3	46.3 +0.0	48.2 +1.2	36.0 +2.5	39.0 +3.0	36.0 +1.9	52.4 +2.4	53.0 +0.0	57.3 +1.2
MATH									
Vanilla	36.7	36.9	37.3	36.3	37.0	34.4	39.8	40.2	40.1
Dynamic	37.0	36.9	37.1	36.3	36.7	34.4	39.7	39.9	39.9
+Ada	36.5 -0.5	37.3 +0.4	37.4 +0.3	36.8 +0.5	36.7 +0.0	34.1 -0.3	39.6 -0.1	39.9 +0.0	39.9 +0.0
+Cache	35.4	35.8	36.0	34.9	33.2	32.1	38.0	38.5	38.8
+Ada+Cache	35.8 +0.4	35.3 -0.5	35.6 -0.4	35.2 +0.3	33.9 +0.7	32.4 +0.3	37.8 -0.2	38.4 -0.1	38.4 -0.4
MBPP									
Vanilla	39.2	38.6	37.0	38.2	37.0	23.2	12.4	12.4	12.8
Dynamic	39.8	38.8	36.8	38.2	37.0	24.6	12.6	12.4	12.2
+Ada	40.2 +0.4	39.8 +1.0	36.4 -0.4	39.4 +1.2	37.6 +0.6	29.8 +5.2	12.8 +0.2	14.2 +1.8	12.4 +0.2
+Cache	35.6	37.8	36.4	38.0	34.8	19.8	12.8	11.6	9.6
+Ada+Cache	39.4 +3.8	38.0 +0.2	36.8 +0.4	36.6 -1.4	36.4 +1.6	36.6 +6.8	12.8 +0.0	11.6 +0.0	12.4 +2.8

Pronounced Accuracy Improvement With Cache. We observe that accuracy gains are particularly pronounced when KV caching is used. Unlike autoregressive decoding, where caching is effectively lossless, block-level KV caching in dLLMs is approximated because key and value tensors vary across time steps, and the decoding order within each block is non-sequential. Notably, accuracy degrades markedly at large block sizes.

Table 3 reports accuracy gains with both PrefixCache and DualCache (Wu et al., 2025). Improvements are twofold: *i)* For large default block sizes B_0, the resulting average block size $\bar{B}$ is smaller, reducing KV-cache approximation error. *ii)* By enhancing semantic locality within each block, inter-block dependencies are reduced, making decoding less sensitive to stale cache entries. The second effect is more impactful: on GSM8K, $\bar{B}$ for $B_0 = 64$ is 33.98, yet accuracy still exceeds that of $B_0 = 32$ without *AdaBlock-dLLM* by 1.90% (no cache) and 6.20% (DualCache). Given that block-level KV caching is a core advantage of semi-AR decoding, these results indicate that the method integrates seamlessly with existing techniques that improve inference efficiency.

Throughput Overhead. Table 2 reports the accuracy, throughput (measured in tokens per second, TPS), and the average number of function evaluations (NFE). The product of throughput and NFE remains stable across methods and block sizes, indicating an approximately inverse relationship between these metrics. This observation suggests that throughput can be improved primarily by reducing NFE, for example, by increasing the parallelism capacity of the denoiser (Wang et al., 2025c) or by improving the sampling efficiency of the sampler.

We observe that the Vanilla decoding, which fixes NFE to the generation budget, maintains almost identical throughput across block sizes. In contrast, dynamic sampling with either fixed or adaptive block sizes benefits from increasing B_0 from 4 to 64, but exhibits a drop in throughput when the block size is increased further. We attribute this initial positive correlation between block size and throughput to the mitigation of late-stage decoding overhead. The subsequent degradation in throughput is mainly due to the noisy output of the denoiser: when a large block size B is used, the semantic coherence within each block weakens, resulting in fewer tokens being decoded per step and thereby requiring more denoise–sample iterations. Given the approximate inverse relationship between NFE and throughput, this increase in NFE leads to reduced throughput. The noisy output is also reflected in the accuracy performance, with the large block size $B_0 = 128$ experiencing a significant performance degradation.

With *AdaBlock-dLLM*, throughput improves for small default block sizes ($B_0 \in \{4, 8\}$) and slightly decreases for larger defaults ($B_0 \in \{16, 32, 64, 128\}$). *AdaBlock-dLLM* attempts to align the block size with semantic steps, leading to $B > B_0$ for small defaults and $B < B_0$ for larger defaults. For small B_0, *AdaBlock-dLLM* effectively reduces late decoding overhead. As B_0 increases, applying $B < B_0$ strengthens the local semantic context and improves sampling quality, at the cost of a modest throughput decrease. Nevertheless, both **Dynamic** and **Dynamic+Ada** achieve substantial speedups over **Vanilla** (top-1) decoding. Figure 6 presents the trade-off between accuracy and throughput, demonstrating the improvements induced by our method.

Table 2: Performance comparison across default block sizes B_0 under a generation budget of $L = 512$. The product of throughput and NFE is nearly identical across methods and block sizes, indicating an approximately inverse relationship between these quantities when no block-level KV caching is applied. *AdaBlock-dLLM* yields throughput gains for $B_0 \in \{4, 8\}$. Boldface indicates superior performance; this convention applies to all tables unless noted otherwise.

	Acc. (%)	TPS	Avg. NFE	TPS×NFE (10^3)	Acc. (%)	TPS	Avg. NFE	TPS×NFE (10^3)	Acc. (%)	TPS	Avg. NFE	TPS×NFE (10^3)
Method	$B_0 = 4$				$B_0 = 8$				$B_0 = 16$			
Vanilla	80.9	16.1	512.0	8.2	80.5	16.1	512.0	8.2	78.8	16.1	512.0	8.2
Dynamic	81.2	43.0	189.4	8.2	80.6	60.0	135.5	8.1	79.1	**74.7**	**109.2**	8.2
+Ada	**81.6**	**51.3**	159.8	8.2	**81.8**	**63.9**	128.3	8.2	**80.6**	73.9	102.4	8.2
Method	$B_0 = 32$				$B_0 = 64$				$B_0 = 128$			
Vanilla	76.8	16.1	512.0	8.2	76.8	16.1	512.0	8.2	71.0	16.0	512.0	8.2
Dynamic	77.6	**85.0**	94.9	8.1	77.3	**89.4**	91.6	8.2	70.7	**81.2**	101.2	8.2
+Ada	**80.6**	83.5	98.5	8.2	**79.5**	87.9	93.4	8.2	**72.8**	80.6	101.6	8.2

Table 3: Accuracy (%) on GSM8K for LLaDA-Instruct across caching methods with $L = 512$.

Method	$B_0 = 16$	$B_0 = 32$	$B_0 = 64$
+PrefixCache	78.2	76.9	75.0
+Ada+PrefixCache	**81.4**	**79.8**	**77.6**
+DualCache	78.0	74.5	75.4
+Ada+DualCache	**80.0**	**78.5**	**80.7**

Table 4: Accuracy (%) on GSM8K for LLaDA-Instruct under different generation budgets.

Method	$L = 256$	$L = 512$	$L = 1024$
Dynamic	78.1	77.6	77.4
+Ada	**78.5**	**80.6**	**79.3**
+Cache	77.4	74.5	75.8
+Ada+Cache	**79.2**	**78.5**	**78.1**

Table 5: Accuracy (%) on GSM8K for LLaDA and Dream across delimiter thresholds $\tau_D \in \{0.3, 0.5, 0.7\}$ with $B_0 = 32$. A smaller τ_D provides sufficient semantic guidance for dLLMs trained from scratch (e.g., LLaDA).

Model	$\tau_D = 0.3$	$\tau_D = 0.5$	$\tau_D = 0.7$
LLaDA-Instruct	**80.59**	79.08	77.94
Dream-Base	75.66	**75.74**	**75.74**

Table 6: Accuracy (%) on IFEval for LLaDA-1.5 across sampling methods. *AdaBlock-dLLM* also improves performance.

Method	$B_0 = 16$	$B_0 = 32$	$B_0 = 64$
Vanilla	69.1	66.7	61.3
Dynamic	69.0	66.7	61.2
+Ada	68.4 -0.6	67.5 +0.8	64.4 +3.2
+Cache	67.5	64.6	59.4
+Ada+Cache	68.9 +1.4	66.4 +1.8	62.7 +3.3

Table 7: Accuracy (%) on GSM8K across eight different delimiter sets with $B_0 = 32$. Results show that using the newline token (`\n`) as the delimiter accounts for most of the accuracy gains, while additionally including the comma and period further improves performance.

Delimiter Set	**Acc. (%)**
None (+Cache)	74.5
`{[\n]}`	**78.5**
`{[,]}`	75.1
`{[.]}`	74.5
`{[,], [.]}`	75.1
`{[\n], [,]}`	**78.5**
`{[\n], [.]}`	**78.3**
`{[\n], [,], [.]}`	**78.7**

Discussion on Difference Between Models. The gains from *AdaBlock-dLLM* vary across models. In particular, *AdaBlock-dLLM* yields larger improvements on the LLaDA family of models. By setting block size according to local semantics, *AdaBlock-dLLM* groups tokens that constitute a semantic step into the same block and focuses on refining the local context. This effect is strongest for dLLMs trained from scratch, which exhibit greater local stochasticity. In contrast, dLLMs adapted from AR models display global autoregressive order and a high degree of local autoregressiveness (Gong et al., 2025). In such a scenario, the improvements from *AdaBlock-dLLM* are correspondingly smaller. Additionally, the fundamental generation quality is limited by the model's denoising quality; our work focuses on mitigating the performance loss due to semi-AR decoding.

5.3 Ablation Studies

Performance Across Different Generation Budgets. We evaluate *AdaBlock-dLLM* under three generation budgets: $L \in \{256, 512, 1024\}$, as shown in Table 4. Across all generation budgets L and decoding settings, *AdaBlock-dLLM* improves generation quality. These results motivate a semantics-aware block-size scheduling design for dLLMs.

Effect on Delimiter Threshold τ_D. We evaluate three delimiter thresholds for each model family, as shown in Table 5. We observe that $\tau_D = 0.3$ yields the best performance in most cases for LLaDA, whereas a higher $\tau_D = 0.5$ is optimal for Dream. We attribute this difference to the distinct confidence distributions of the two models. LLaDA is trained purely from scratch and exhibits lower variance within the volatility band, whereas Dream is adapted from autoregressive models and shows substantially higher variance (Figure 8). Consequently, different thresholds are required to track the boundary of a semantic step. Additionally, overly high thresholds (e.g., $\tau_D = 0.9$) often cause the scheduler to revert to its default behavior, reducing its effectiveness.

Selection of Delimiter Set $\mathcal{D}$. We apply more delimiter sets $\mathcal{D}$ to include additional tokens (comma and period), which often mark the termination of local semantic context. Table 7 shows that although accuracy improvements vary, the inclusion of the comma and period achieves higher accuracy than the dynamic sampling baseline. These results highlight the importance of aligning block size with semantic steps in semi-AR decoding. We provide further analysis in A.4.

Performance on Non-Reasoning Benchmarks. We further evaluate the performance of *AdaBlock-dLLM* on IFEval (Zhou et al., 2023), a benchmark that examines the instruction-following capability of LLMs. As shown in Table 6, with the delimiter set $\mathcal{D} = \{$`[\n]`, `[,]`, `[.]`$\}$, *AdaBlock-dLLM* yields accuracy improvements, especially when integrated with block-level KV caching. These results suggest that aligning block sizes with semantic steps effectively enhances sampling quality, leading to improved performance on tasks other than math reasoning or coding.

Limitations. *AdaBlock-dLLM* effectively enhances the existing semi-AR decoding paradigm by aligning block sizes with semantic steps. However, the proposed block scheduler may be less effective when the generation budget is small (e.g., multiple-choice questions), where semi-AR decoding is not particularly beneficial. Additionally, the decoding process of dLLMs includes both denoising and sampling. *AdaBlock-dLLM* primarily improves sampling quality; when the denoising outputs are poor, the benefit of *AdaBlock-dLLM* diminishes.

6 Conclusion

This work proposes *AdaBlock-dLLM*, a training-free, plug-and-play scheduler that enhances the existing semi-autoregressive decoding paradigm. We identify two fundamental limitations of conventional semi-AR decoding (late decoding overhead and premature decoding errors) that motivate adaptive block-size scheduling. Building on an analysis of confidence dynamics, *AdaBlock-dLLM* adaptively adjusts the block size at runtime, aligning block sizes with semantic steps. Extensive experiments across benchmarks demonstrate improvements in generation quality of up to 5.3% under a comparable speed budget. We hope that our semantics-aware adaptive approach and statistical analysis will inspire future training and inference strategies for dLLMs.

Ethics Statement

This work adheres to the ICLR Code of Ethics. In this study, no research involving human subjects or animals was conducted. All datasets used, including GSM8K, MATH, HumanEval, and MBPP, were obtained in compliance with relevant usage guidelines, ensuring no violations of privacy. We took care to avoid the bias and discriminatory outcomes throughout our research process. No personally identifiable information was used, and no experiments were conducted that could raise privacy or security concerns. We are committed to maintaining transparency and integrity throughout the research process.

Reproducibility Statement

We ensure that all reported results are reproducible. The code repository is provided in the supplementary material to facilitate replication and verification. The experimental setup, including model configurations and hardware details, is described in Section 5.1. We also provide a full description of our methodology, including detailed pseudocode in Algorithm 1, to support understanding and reproduction. Additionally, the open-source models used in this work (e.g., LLaDA and Dream) are publicly available, enabling consistent and reproducible evaluation.

References

Marianne Arriola, Aaron Gokaslan, Justin T Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2503.09573.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, 2021a. URL https://arxiv.org/abs/2107.03006.

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.

Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai ”Hellen” Li, and Yiran Chen. Dpad: Efficient diffusion language models with suffix dropout, 2025. URL https://arxiv.org/abs/2508.14148.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Wenghai Wang, Qipeng Guo, Kai Chen, Biqing Qi*, and Bowen Zhou. Sdar: A synergistic diffusion–autoregression paradigm for scalable sequence generation, 2025. URL https://github.com/JetAstra/SDAR.

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.

Gemini Diffusion. Gemini diffusion: A state-of-the-art, experimental text diffusion model. https://deepmind.google/models/gemini-diffusion/, 2025. Accessed: 2025-09-22.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

Haoyu He, Katrin Renz, Yong Cao, and Andreas Geiger. Mdpo: Overcoming the training-inference divide of masked diffusion language models. *arXiv preprint arXiv:2508.13148*, 2025.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *34th Conference on Neural Information Processi*, 2020. URL `https://arxiv.org/abs/2006.11239`.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in neural information processing systems*, 35:8633–8646, 2022.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv preprint arXiv:2507.18578*, 2025.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *arXiv preprint arXiv:2102.05379*, 2021. URL `https://arxiv.org/abs/2102.05379`.

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

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

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

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

Shufan Li, Konstantinos Kallidromitis, Hritik Bansal, Akash Gokul, Yusuke Kato, Kazuki Kozuka, Jason Kuen, Zhe Lin, Kai-Wei Chang, and Aditya Grover. Lavida: A large diffusion language model for multimodal understanding. *arXiv preprint arXiv:2505.16839*, 2025b.

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

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution, 2024. URL `https://arxiv.org/abs/2310.16834`.

Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed–dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025.

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

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 4195–4205, 2023.

Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion beats autoregressive in data-constrained settings. *arXiv preprint arXiv:2507.15857*, 2025.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015. URL `https://arxiv.org/abs/1503.03585`.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

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. *arXiv preprint arXiv:2508.02558*, 2025a.

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*, 2025b.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling. In *ICLR 2025 Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*, 2025a.

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

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025c.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models, 2025d. URL `https://arxiv.org/abs/2509.06949`.

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

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

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal large diffusion language models. *arXiv preprint arXiv:2505.15809*, 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.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding, 2025. URL `https://arxiv.org/abs/2505.16990`.

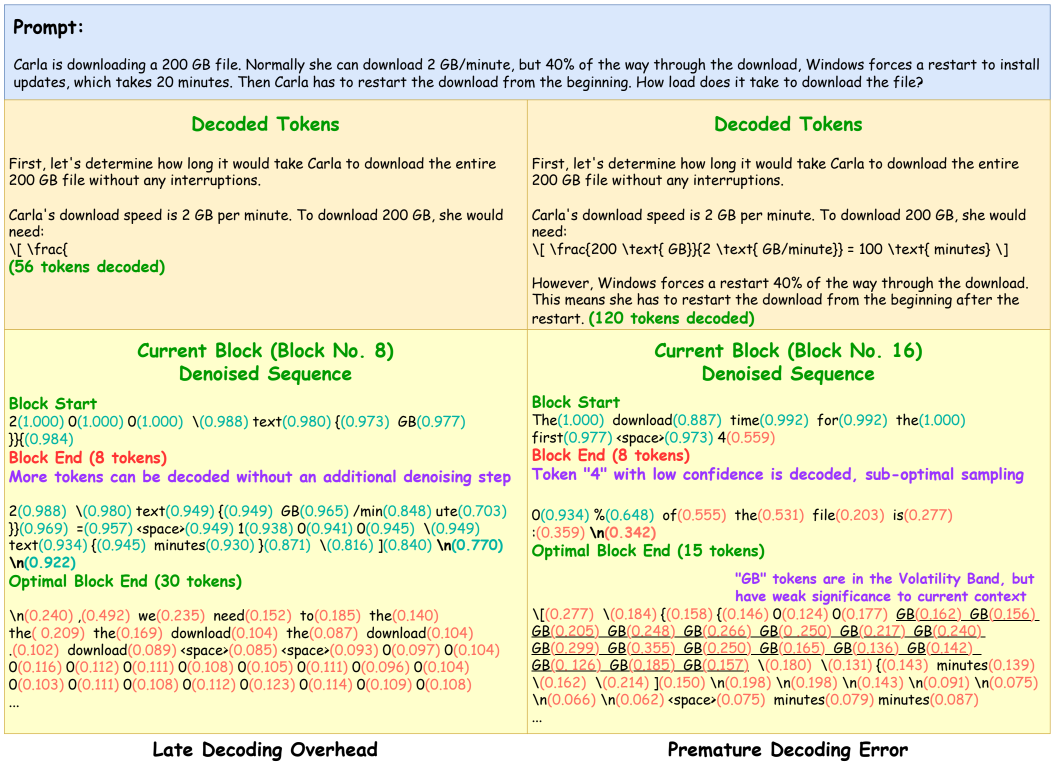
Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, et al. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv preprint arXiv:2508.08712*, 2025.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

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

A Appendix

A.1 Case Study For The Two Fundamental Issues In Semi-Autoregressive Decoding


Prompt:

Carla is downloading a 200 GB file. Normally she can download 2 GB/minute, but 40% of the way through the download, Windows forces a restart to install updates, which takes 20 minutes. Then Carla has to restart the download from the beginning. How load does it take to download the file?

Decoded Tokens

First, let's determine how long it would take Carla to download the entire 200 GB file without any interruptions.

Carla's download speed is 2 GB per minute. To download 200 GB, she would need:
\[\frac{
(56 tokens decoded)

Current Block (Block No. 8)
Denoised Sequence

Block Start
2(1.000) 0(1.000) 0(1.000) \(0.988) text(0.980) {(0.973) GB(0.977) }}{(0.984)
Block End (8 tokens)
More tokens can be decoded without an additional denoising step

2(0.988) \(0.980) text(0.949) {(0.949) GB(0.965) /min(0.848) ute(0.703) }}(0.969) =(0.957) <space>(0.949) 1(0.938) 0(0.941) 0(0.945) \(0.949) text(0.934) {(0.945) minutes(0.930) }(0.871) \(0.816)](0.840) **\n(0.770)** **\n(0.922)**
Optimal Block End (30 tokens)

\n(0.240) ,(0.492) we(0.235) need(0.152) to(0.185) the(0.140) the(0.209) the(0.169) download(0.104) the(0.087) download(0.104) .(0.102) download(0.089) <space>(0.085) <space>(0.093) 0(0.097) 0(0.104) 0(0.116) 0(0.112) 0(0.111) 0(0.108) 0(0.105) 0(0.111) 0(0.096) 0(0.104) 0(0.103) 0(0.111) 0(0.108) 0(0.112) 0(0.123) 0(0.114) 0(0.109) 0(0.108)
...

Late Decoding Overhead

Decoded Tokens

First, let's determine how long it would take Carla to download the entire 200 GB file without any interruptions.

Carla's download speed is 2 GB per minute. To download 200 GB, she would need:
\[\frac{200 \text{ GB}}{2 \text{ GB/minute}} = 100 \text{ minutes} \]

However, Windows forces a restart 40% of the way through the download. This means she has to restart the download from the beginning after the restart. **(120 tokens decoded)**

Current Block (Block No. 16)
Denoised Sequence

Block Start
The(1.000) download(0.887) time(0.992) for(0.992) the(1.000) first(0.977) <space>(0.973) 4(0.559)
Block End (8 tokens)
Token "4" with low confidence is decoded, sub-optimal sampling

0(0.934) %(0.648) of(0.555) the(0.531) file(0.203) is(0.277) :(0.359) **\n(0.342)**
Optimal Block End (15 tokens)

"GB" tokens are in the Volatility Band, but have weak significance to current context
\[(0.277) \(0.184) {(0.158) {(0.146) 0(0.124) 0(0.177) GB(0.162) GB(0.156) GB(0.205) GB(0.248) GB(0.266) GB(0 .250) GB(0.217) GB(0.240) GB(0.299) GB(0.355) GB(0.250) GB(0.165) GB(0.136) GB(0.142) GB(0. 126) GB(0.185) GB(0.157) \(0.180) \(0.131) {(0.143) minutes(0.139) \(0.162) \(0.214)](0.150) \n(0.198) \n(0.198) \n(0.143) \n(0.091) \n(0.075) \n(0.066) \n(0.062) <space>(0.075) minutes(0.079) minutes(0.087)
...

Premature Decoding Error

Figure 7: A case study of the two fundamental issues (Late Decoding Overhead and Premature Decoding Error). The configuration uses dynamic sampling with a generation budget of $L = 512$ and a block size of $B = 8$.

A.2 Semi-AR Decoding Algorithm With Adaptive Block Size

Algorithm 2 Auxiliary: Denoiser

```
    Inputs: mask predictor $p_\theta$; vocabulary $\mathcal{V}$; index set $\mathcal{J}$; current sequence $\mathbf{y}$.
    Outputs: predicted sequence $\hat{\mathbf{y}}$; confidences $\mathbf{c}$
1:  function DENOISER($p_\theta$, $\mathcal{V}$, $\mathcal{J}$, $\mathbf{y}$)
2:      for $i \in \mathcal{J}$ do
3:          $\hat{y}_i \leftarrow \arg\max_{v \in \mathcal{V}} \; p_\theta(v \mid \mathbf{y}, i)$
4:          $c_i \leftarrow \max_{v \in \mathcal{V}} \; p_\theta(v \mid \mathbf{y}, i)$
5:      end for
6:      return $\hat{\mathbf{y}}$, $\mathbf{c}$
7:  end function
```

Algorithm 3 Auxiliary: Threshold-Based Dynamic Sampling

```
Inputs: current sequence $\mathbf{y}$; predicted sequence $\hat{\mathbf{y}}$; confidences $\mathbf{c}$; unmasking threshold $\tau$; index set $\mathcal{J}$.
Output: sampled index set $\mathcal{S}$
 1: function THRESHOLD-SAMPLE($\mathbf{y}$, $\hat{\mathbf{y}}$, $\mathbf{c}$, $\tau$, $\mathcal{J}$)
 2:     $\mathcal{S} \leftarrow \emptyset$
 3:     $\mathcal{J}_{\text{mask}} \leftarrow \{ i \in \mathcal{J} \mid y_i = \texttt{[MASK]} \}$    ▷ Select all masked positions in the block
 4:     Select index $i_{\text{top}}$ with highest confidence $c_i$
 5:     $\mathcal{S} \cup i_{\text{top}}$
 6:     for $i \in \mathcal{J}$ do
 7:         if $i \in \mathcal{J}_{\text{mask}} \wedge c_i \geq \tau$ then
 8:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{i\}$
 9:         end if
10:     end for
11:     return $\mathcal{S}$
12: end function
```

Algorithm 4 Adaptive Semi-AR Decoding with Semantic-Aware Block Size

```
Inputs: mask predictor $p_\theta$; vocabulary $\mathcal{V}$; initial sequence $\mathbf{y}^{(T)}$ with index set $\mathcal{J}$; generation budget $L$; default block size $B_0$; delimiter set $\mathcal{D}$; delimiter threshold $\tau_{\mathcal{D}}$; unmasking threshold $\tau$.
Output: decoded sequence $\mathbf{y}$.
 1: generated length $g \leftarrow 0$,   timestep $t \leftarrow T$
 2: while $g < L \wedge t \geq 1$ do
 3:
 4:     ▷ First denoising to obtain predicted sequence and confidence at step t
 5:     $(\hat{\mathbf{y}}^{(t)}, \mathbf{c}^{(t)}) \leftarrow$ DENOISER($p_\theta$, $\mathcal{V}$, $\mathcal{J}$, $\mathbf{y}^{(t)}$)
 6:
 7:     ▷ Compute block size
 8:     $B \leftarrow$ COMPUTEBLOCKLENGTH($\hat{\mathbf{y}}^{(t)}$, $\mathbf{c}^{(t)}$, $L$, $B_0$, $\mathcal{D}$, $\tau_{\mathcal{D}}$, $g$)
 9:     $\mathcal{J}_{\text{blk}} \leftarrow \{ g, g{+}1, \dots, g{+}B{-}1 \}$
10:
11:     ▷ First sample
12:     $\mathcal{S} \leftarrow$ THRESHOLD-SAMPLE($\mathbf{y}^{(t)}$, $\hat{\mathbf{y}}^{(t)}$, $\mathbf{c}^{(t)}$, $\tau$, $\mathcal{J}_{\text{blk}}$)
13:     for $i \in \mathcal{S}$ do
14:         $y_i^{t-1} \leftarrow \hat{y}_i^{t}$    ▷ sample tokens with high confidence
15:     end for
16:
17:     $y_j^{(t-1)} \leftarrow y_j^{(t)}\ \forall j \notin \mathcal{S}$,   $t \leftarrow t-1$    ▷ Copy other tokens
18:     $\mathcal{J}_{\text{mask}} \leftarrow \{ i \in \mathcal{J}_{\text{blk}} \mid y_i^{(t)} = \texttt{[MASK]} \}$
19:
20:     ▷ In-block denoise–sample cycles
21:     while $\mathcal{J}_{\text{mask}} \neq \emptyset \wedge t \geq 1$ do
22:         $(\hat{\mathbf{y}}^{(t)}, \mathbf{c}^{(t)}) \leftarrow$ DENOISER($p_\theta$, $\mathcal{V}$, $\mathcal{J}_{\text{mask}}$, $\mathbf{y}^{t}$)
23:         $\mathcal{S} \leftarrow$ THRESHOLD-SAMPLE($\mathbf{y}^{t}$, $\hat{\mathbf{y}}^{(t)}$, $\mathbf{c}^{(t)}$, $\tau$, $\mathcal{J}_{\text{mask}}$)
24:         for $i \in \mathcal{S}$ do
25:             $y_i^{(t-1)} \leftarrow \hat{y}_i^{(t)}$
26:         end for
27:         $y_j^{(t-1)} \leftarrow y_j^{(t)}\ \forall j \notin \mathcal{S}$,   $t \leftarrow t-1$
28:         $\mathcal{J}_{\text{mask}} \leftarrow \{ i \in \mathcal{J}_{\text{blk}} \mid y_i^{(t)} = \texttt{[MASK]} \}$
29:     end while
30:
31:     $g \leftarrow g + B$
32: end while
33:
34: return $\mathbf{y}^{t}$
```

A.3 Confidence Dynamics for Dream-Base

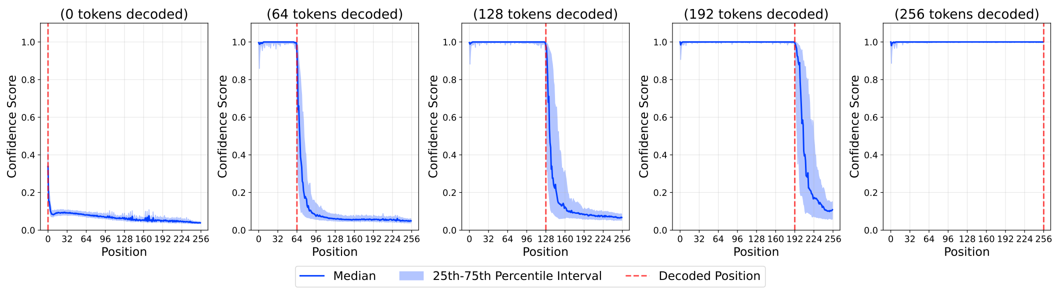


Figure 8: Confidence scores across sequence positions for Dream-v0-Base-7B, evaluated on 100 samples from the GSM8K benchmark. Adapted models such as Dream exhibit a similar degree of global autoregressiveness to LLaDA, but with higher variance in the volatility band. This increased variance motivates the use of a higher delimiter threshold τ_D to provide stronger semantic guidance.

A.4 Further Analysis of the Delimiter Set

The choice of $\backslash\mathrm{n}$ as a delimiter token is also supported by the statistics of confidence drops between consecutive tokens. Large confidence drops indicate sharp semantic boundaries within the volatility band and can thus be used to segment semantic steps. Evaluating 100 samples from the GSM8K dataset, we find that $\backslash\mathrm{n}$ frequently leads to large confidence drops, indicating semantic boundaries.

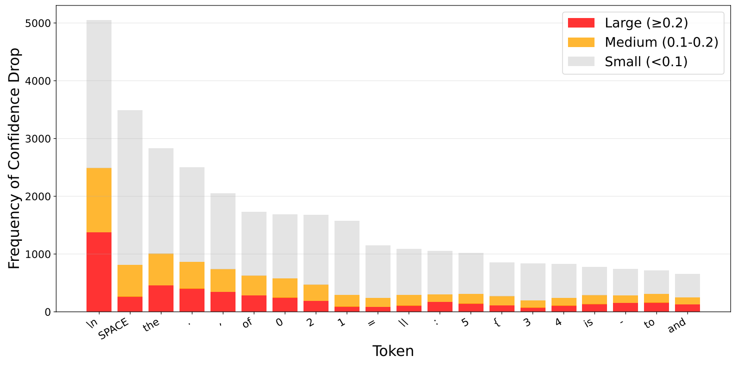


Figure 9: The frequency of confidence drops between consecutive tokens.

A.5 The Use of Large Language Models

LLMs were used for polishing the manuscript. Specifically, we used an LLM to assist in correcting grammar errors to improve readability. It is important to note that the LLM was not involved in the ideation, research methodology, or experimental design. All intellectual contributions and scientific ideas developed in this work originated from the authors. The contributions of the LLM were solely focused on improving the linguistic quality of the paper, with no involvement in the scientific content or data analysis. The authors take full responsibility for the content of the manuscript, including any text polished by the LLM. We have ensured that the LLM-polished text adheres to ethical guidelines and does not contribute to plagiarism or scientific misconduct.