# Diffusion with Truncated Blocks: Fast and High-Quality Text Generation using Truncated Block Generation

**Yuyan Zhou, Weiyu Chen*, James Kwok**
Department of Computer Science and Engineering
Hong Kong University of Science and Technology

## Abstract

Diffusion-based Large Language Models (dLLMs) are emerging as a powerful alternative to traditional autoregressive models. These models learn to generate text by iteratively denoising masked sequences. In this work, we identify a critical problem in dLLMs: the model's attention is wastefully expended on uninformative mask tokens, diluting its focus on meaningful context. We term this phenomenon "attention dilution". We further show that this artifact is amplified by token-level noising, whereas models employing sequence-level noise exhibit a reduced effect. To resolve this problem, we introduce Truncated Block Generation, a novel sampling algorithm that not only mitigates attention dilution but also enables faster inference and flexible-length sequence generation. Extensive experiments validate our analysis and demonstrate the marked effectiveness of our proposed method in enhancing both the performance and efficiency of dLLMs.

## 1 Introduction

Diffusion large language models (dLLMs) (Nie et al., 2025; Ye et al., 2025; Zhu et al., 2025; Khanna et al., 2025) have recently emerged as an alternative promising paradigm for language modeling. While autoregressive models (ARMs) (Achiam et al., 2023; Liu et al., 2024a; Dubey et al., 2024; Wang et al., 2025a; Zhou et al., 2024) generate text token-by-token in a strict left-to-right manner, dLLMs operate on a sequence of masked tokens, iteratively refining the entire sequence in parallel. This non-autoregressive, denoising approach enables bidirectional attention, parallel generation and more flexible generation patterns, directly addressing some of the inherent limitations of ARMs.

The standard dLLM employs a uniform, sequence-level noise, where all masked tokens in a sequence have equal importance (Nie et al., 2025). As illustrated in Figure 1(a), as the same loss weight is assigned to every masked position, its surrounding context is ignored. This sentence-level training leads to suboptimal learning, particularly for tokens with varying degrees of contextual informativeness (Ye et al., 2025) A more nuanced approach is proposed in Dream (Ye et al., 2025), which uses token-level noise and dynamically reweights the loss for each token based on its contextual informativeness. For example, as illustrated in Figure 1(b), a masked token surrounded by unmasked neighbors is considered more informative, and is thus assigned a higher loss weight. This encourages the model to prioritize predicting tokens with rich contextual support during inference, leading to a more structured generation process.

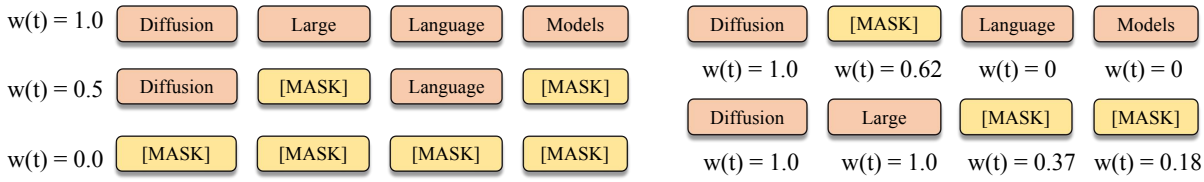
However, through empirical and theoretical evidence, we identify a critical but previously overlooked drawback of current dLLMs, especially for models trained with the token-level noise. As the number of masked tokens in the generation context grows, informativeness of each individual mask token decreases significantly. Attending to a long sequence of these less informative mask tokens dilutes the model's attention, preventing it from focusing on the truly informative tokens in the prompt and the partially informative mask tokens. This "attention dilution" degrades generation quality, particularly for long sequences. Moreover, our analysis reveals this issue is reduced for models trained with sequence-level noise, where all mask tokens are trained to carry useful information and contribute meaningfully to the self-attention mechanism.

Inspired by this core insight, we propose Truncated Block Generation, a sampling method designed specifically to mitigate attention dilution in dLLMs. Instead of appending a single, long sequence of masks to the prompt like semi-AR (Nie et al., 2025), we divide the generation process into

* Corresponding author



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 4335–4348




(a) Sequence-Level Noise. (b) Token-Level Noise.

Figure 1: The noise level of masked diffusion language models: Different from LLaDA, which uses sequence-level noise, Dream uses token-level noise to re-weight the loss of different tokens.

sequential rounds. In each round, a shorter block of masks is appended and denoised, and we use the truncated unmasked block as the new context. This strategy ensures that the model always attends to a context with a high density of informative tokens, alleviating the dilution problem. As a result, Truncated Block Generation accelerates sampling speed, supports flexible-length generation, and significantly improves output quality.

Our main contributions are summarized below.

1. We identify the "attention dilution" problem in dLLMs and demonstrate that an excessive number of mask tokens degrades the model performance by diverting the model's attention away from informative context.

2. We provide theoretical analysis explaining the underlying mechanism of attention dilution. Furthermore, we clarify why this issue is reduced in models trained with sequence-level noise, highlighting a critical difference in their behavior.

3. Based on the above analysis, we propose Truncated Block Generation, a simple yet effective sampling strategy that directly mitigates attention dilution. Extensive experiments validate that it leads to substantial improvements in generation quality, faster inference, and robust support for flexible-length outputs.

## 2 Background

### 2.1 Diffusion Large Language Models

Let $\boldsymbol{x}_0 \in \mathbb{R}^{1\times N}$ denote a sequence of discrete random variables. Discrete diffusion models (Sohl-Dickstein et al., 2015; Meng et al., 2022; Austin et al., 2021a; Nie et al., 2025) are a class of latent variable models of the form:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T})\, \mathrm{d}\boldsymbol{x}_{1:T}.$$

They are characterized by a forward noising process and a learned reverse denoising process. The forward process progressively corrupts the original data $\boldsymbol{x}_0$ into sequences of increasingly noisy masked tokens $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ (where each $\boldsymbol{x}_t \in \mathbb{R}^{1\times N}$) that are of the same dimensionality as the data $\boldsymbol{x}_0 \sim p_{\text{data}}$: $q(\boldsymbol{x}_{1:T}||\boldsymbol{x}_0) = \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$. Here, $q(\boldsymbol{x}_t|\boldsymbol{x}_s) = \text{Cat}(\boldsymbol{x}_t; \mathbf{Q}_t\boldsymbol{x}_s)$, and $\mathbf{Q}_t$ is the transition matrix. The backward process learns to gradually denoise the masked sequence back to the original data distribution by iteratively predicting masked tokens as $t$ moves from $T$ to 0:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{\boldsymbol{x}_{1:T}} p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t).$$

The model parameter $\boldsymbol{\theta}$ is optimized by minimizing the negative log-likelihood of the clean data $\boldsymbol{x}_0$. When the absorbing kernel is used, learning the denoising process leads to the minimization of the following Evidence Lower Bound (ELBO) of the log likelihood (Nie et al., 2025):

$$\mathcal{L}_{\text{LLaDA}}(\theta) = -\mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x}),\epsilon\sim\mathcal{N}(0,1),t\sim\mathcal{U}(1,T)} w(t) \times \sum_{n=1}^{N} \mathbf{1}_{[x_t^n=\texttt{MASK}]} \log p_{\boldsymbol{\theta}}(x_0^n|\boldsymbol{x}_t),$$

where $\mathbf{1}_{[\boldsymbol{x}_t^n=\texttt{MASK}]}$ is the indicator function that ensures that the loss is computed only on the masked tokens, and $w(t) \in (0, 1]$ is a time-dependent reweighting term. To enable the model to decode easier tokens earlier, Ye et al. (2025) propose to use contextual token-level weighting for the token loss:

$$\mathcal{L}_{\text{Dream}}(\theta) = -\mathbb{E}_{\boldsymbol{x}_0\sim q(\boldsymbol{x}_0),\epsilon\sim\mathcal{N}(0,1),t\sim\mathcal{U}(1,T)} \sum_{n=1}^{N} w(\boldsymbol{x}_t, n)\mathbf{1}_{[x_t^n=\texttt{MASK}]} \log p_{\boldsymbol{\theta}}(x_0^{n-1}|\boldsymbol{x}_t),$$

where $N$ is the length of sentence and $w(\boldsymbol{x}_t, n)$ uses a mixture of geometric distributions to quan-

tify the information contribution of each clean token relative to the noised token.

$$w(\boldsymbol{x}_t, n) = \frac{1}{2}\sum_{i=1}^{N} \mathbf{1}_{[x_t^i \neq \texttt{MASK}]} \text{Geo}(p, |n-i|-1), \tag{1}$$

where $\text{Geo}(p,k) = (1-p)^k p$.

### 2.2 Bi-directional attention of Diffusion Language Models

For an input sentence $X \in \mathbb{R}^{1\times l}$, the dLLM first projects each token to its embedding $X_0 \in \mathbb{R}^{1\times l\times d}$, where $d$ is the dimensionality of the hidden layer. Unlike AR transformers which add an unidirectional causal attention mask to the attention score and generate text sequentially in a left-to-right manner, MDM's transformer uses bi-directional attention and attends to both the previous and future tokens.

$$Q_0 = X_0 W_Q, \quad K_0 = X_0 W_K, \quad A^0 = (\frac{Q_0 K_0^\top}{\sqrt{d}}),$$
$$\text{Attn}(X_0) = \text{Softmax}(A^0).$$

Thus, on inference, the MDM (i) attends to the previous prompt tokens to understand the question, and (ii) attends to the following [MASK] tokens for generating and organizing the content.

## 3 Problem of Attention Dilution

MDMs generate sequences by appending $m_1$ mask tokens $M_1 \in \mathbb{R}^{1\times m_1}$ to the end of a prompt $P_0 \in \mathbb{R}^{1\times c}$. The combined sequence is projected to an embedding space, as:

$$X_1 = \texttt{wte}\left(\begin{bmatrix} P_0 \\ M_1 \end{bmatrix}\right) \in \mathbb{R}^{1\times(c+m_1)\times d}, \tag{2}$$

where wte denotes the word token embedding layer. For two mask tokens $x_j$ and $x_{j+1}$ in $X_1$, we find that with the increase of the token'spositions, its contextual information in (1) decreases exponentially:

$$\frac{w(X_1, j+1)}{w(X_1, j)} = \frac{\sum_{i=0}^{c+m_1-1}(1-p)^{|n-i+1|}}{\sum_{i=1}^{c+m_1}(1-p)^{|n-i|}} = 1-p.$$

If the appended masked sequence is long enough, we have the theorem below:

**Theorem 1.** *For every $\epsilon > 0$, there exists $k > 0$ such that $w(\boldsymbol{x}_u, u) < \epsilon,\ \forall u > k$.*

Tokens with contextual information less than $\epsilon$ are called non-informative. The model attending on these tokens can only get positional information (i.e., length of the remaining free space for generation), but this is not helpful for understanding and solving the problem.

This introduces a critical challenge, which we named *attention dilution*. We first define the attention on tokens whose contextual information is greater than $\epsilon$ as contextual attention:

**Definition 1.** *Let the ith mask token of $X_1$ be $\boldsymbol{m}_i^1$. Define its contextual attention as its attention on tokens whose contextual information is greater than $\epsilon$:*

$$CA(\boldsymbol{m}_i^1) = \sum_{j=1}^{c+m_1} \mathbf{1}_{[w(X_1,j)>\epsilon]} Attn_{ij}(X_1).$$

Next, we prove that the excessive redundant mask tokens dilutes the contextual attention.

**Theorem 2.** *For the $X_1$ defined above, we append $m_2 - m_1$ mask tokens to its end and denote it as $X_2$:*

$$X_1 = \texttt{wte}\left(\begin{bmatrix} P_0 \\ M_1 \end{bmatrix}\right) \in \mathbb{R}^{1\times(c+m_1)\times d}, \tag{3}$$

$$X_2 = \texttt{wte}\left(\begin{bmatrix} P_0 \\ M_2 \end{bmatrix}\right) \in \mathbb{R}^{1\times(c+m_2)\times d}. \tag{4}$$

*We denote the ith mask token of $X_1$ as $\boldsymbol{m}_i^1$ and the ith mask token of $X_2$ as $\boldsymbol{m}_i^2$. The contextual attention of $\boldsymbol{m}_i^1$ is strictly greater than the contextual attention of $\boldsymbol{m}_i^2$:*

$$CA(\boldsymbol{m}_i^1) = \sum_{j=1}^{c+m_1} \frac{\exp(A_{ij}^1)}{\sum_{k=1}^{c+m_1}\exp(A_{ik}^1)}$$
$$> \sum_{j=1}^{c+m_1} \frac{\exp(A_{ij}^2)}{\sum_{k=1}^{c+m_2}\exp(A_{ik}^2)} = CA(\boldsymbol{m}_i^2).$$

We empirically validate this phenomenon in Figure 3. As can be seen from Figure 3(a) and Figure 3(b), given the prompt "You are an expert python coder", when the number of appended [MASK] is 64, the attention of the first mask token to the [You] token is $2.1 \times 10^{-5}$. However, if the number of appended [MASK] is 1024, the attention of the first mask token to the [You] token will decrease to $5.3\times10^{-6}$. The same phenomenon can be observed for attention on all of the prompt tokens. Moreover, we also visualize how the contextual attention affected by the number of mask tokens

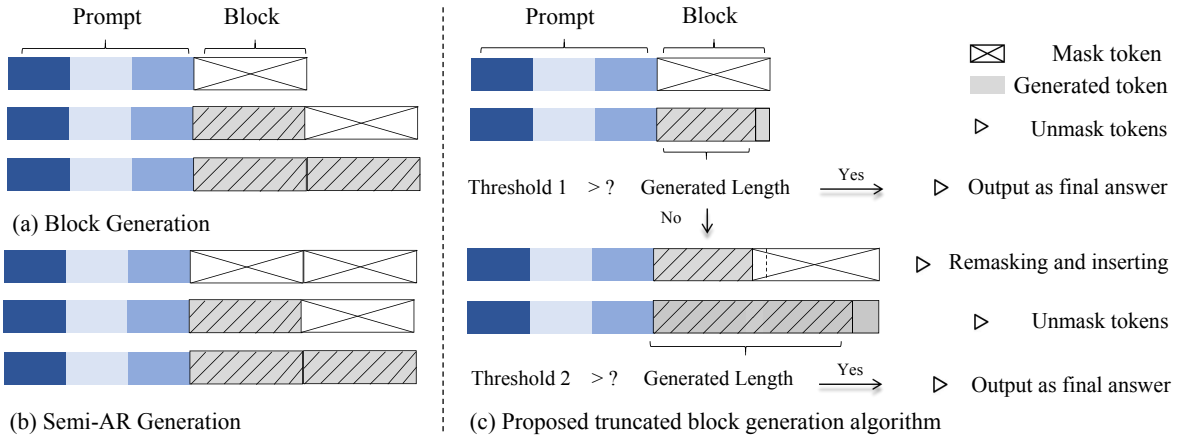
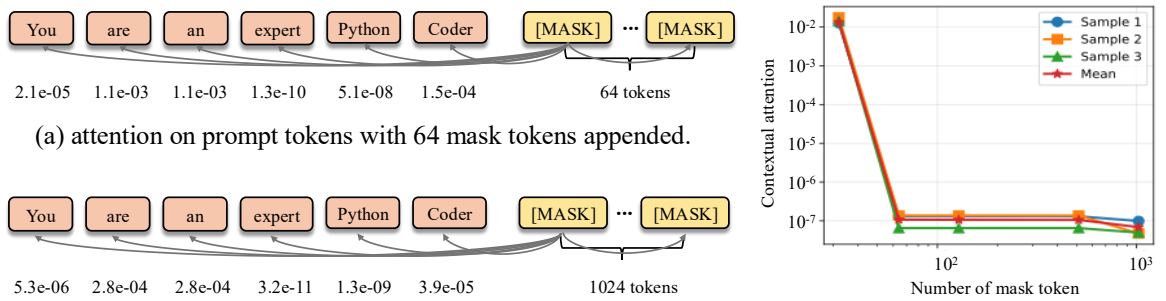



Figure 2: Comparison of block generation, semi-AR generation and our truncated block generation.




(a) attention on prompt tokens with 64 mask tokens appended.

(b) attention on prompt tokens with 1024 mask tokens appended.

(c) Contextual attention with different number of mask token.

Figure 3: The attention of Dream will be diluted by excessive masks.

and show the result in Figure 3(c). We draw the curve of three samples and the mean on the MBPP dataset, and we can see that the contextual attention is decreasing with the increase of the number of mask tokens. This demonstrates that an excess of masks diverts focus from the crucial context provided by the prompt.

The attention dilution problem is significant in MDMs trained with token-level noise but is reduced in models trained with sequence-level noise. Recall that in the attention layer, $q^\top k$ measures the similarity of different tokens, so mask tokens will attend more to mask tokens rather than prompt tokens. However, prompt tokens are more important for generation. We observe that for both Dream (Ye et al., 2025) and LLaDA (Nie et al., 2025), they will copy a previous token as query for self-attention in the following layers. As can be seen from Figure 4(a) and Figure 4(b), for the mask tokens, their attention will be paid to one token in the prompt, we denote the index of the prompt token as $k$ and the mask token as $\boldsymbol{m}$:

$$\text{Attn}_{\boldsymbol{m}}(X_0) = \boldsymbol{e}_k, \quad \text{Attn}_{\boldsymbol{m}}(X_0)\boldsymbol{v} = X_0^k W_V,$$

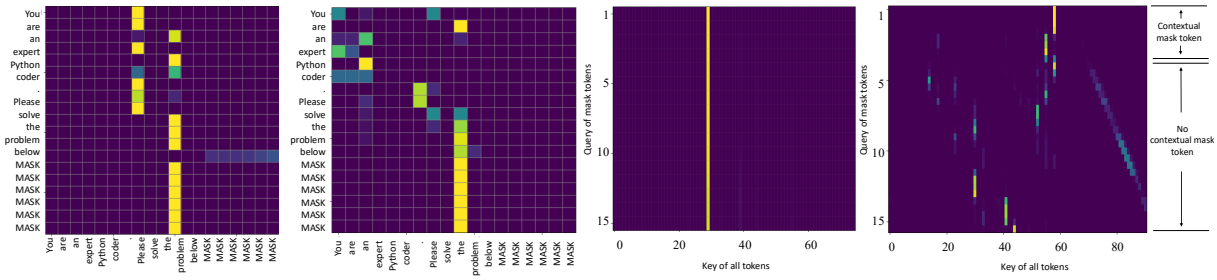
where $\boldsymbol{e}_k$ denotes the unit vector with a 1 in the $k$th position and zeros elsewhere. The attention layer will output the linear transformed token $X_0^k W_V$ in all the mask tokens' positions and copy it to these positions using residual connection.

$$\boldsymbol{m} + \text{Layer}(X_0) = \boldsymbol{m} + X_0^k W_v. \tag{5}$$

This mechanism allows mask tokens to carry forward contextual information from the prompt. However, its application across the sequence differs between training methods.

For models trained with token-level noise (e.g., Dream), only the informative masks successfully learn this copying behavior. As shown in Figure 4(d), non-informative masks, being too distant, fail to learn a meaningful attention pattern and instead exhibit random behavior. These semantically empty tokens are the source of attention dilution.

For models trained with sequence-level noise (e.g., LLaDA), the model is incentivized to make all




(a) LLaDA attention heat map with 6 mask tokens

(b) Dream attention heat map with 6 mask tokens

(c) LLaDA attention heat map with 16 mask tokens

(d) Dream attention heat map with 16 mask tokens

Figure 4: Attention heat maps of LLaDA and Dream.

mask tokens useful for the final prediction. Consequently, the copying behavior propagates throughout the entire masked sequence, as seen in Figure 4(c). In this case, attending to mask token is beneficial, as they all try to carry relevant contextual information. Therefore, this issue will be reduced in models trained with sequence-level noise.

## 4 Truncated Block Generation

In this section, we introduce the proposed method, Truncated Block Generation (TBG). We first motivate the approach by outlining the trade-offs in generative sequence length. We then detail the algorithm and discuss the importance the truncation step.

As discussed in Section 3, MDMs face the "attention dilution" challenge. Appending a long sequence of [MASK] tokens to a prompt dilutes the model's attention on the informative context, which degrades generation quality. However, a short masked sequence provides insufficient context for complex tasks, such as generating complete code blocks or executing a full chain of thought for mathematical reasoning. This limitation also leads to poor performance. This raises a key question:

*How can we generate sequences with sufficient length without causing attention dilution?*

To address this challenge, we propose an iterative method called Truncated Block Generation. The core idea is to generate text in small-size blocks, allowing the model to extend the sequence as needed without being overwhelmed by an excessive number of [MASK] tokens at any single step. The pseudocode of the proposed algorithm is shown in Algorithm 1.

Specifically, the algorithm proceeds as follows: First, the model generates content within a masked block of a predefined length (line 2 to 20). Next, we measure the length of the valid generated output (i.e., the sequence before any <eos> token appears) and compare it against a continuation threshold, $\gamma$ (line 21). If the valid length is less than $\gamma$, we consider the generation complete. This indicates the model does not need the full block capacity to finish its response. If the valid length is greater than or equal to $\gamma$, we conclude the model needs more space. We then truncate the generated sequence to a shorter, predetermined length. This truncated output serves as the new context, to which a new block of [MASK] tokens is appended for the next round of generation. This block-wise approach mitigates attention dilution by ensuring that only a manageable number of uninformative [MASK] tokens are present in any generation step.

The truncation step is important to the success of this method applying to MDMs. Without it (i.e., a truncation length of zero), this algorithm would simplify to naive block generation (Arriola et al., 2025), a strategy with a significant flaw for current MDMs. Due to their bi-directional attention mechanism, MDMs are aware of sequence boundaries. As generation approaches the end of a block, the model is strongly biased toward producing an <eos> token to complete the output within the available space, as illustrated in Figure 6(a). This premature termination prevents the model from generating content that naturally extends beyond the block boundary. In contrast, our truncation method re-masks the tokens near the end of the generated block (Figure 6(b)). This action effectively removes the premature <eos> token and signals to the model that the sequence is incomplete (such as "#" token in GSM8K and "[DONE]" token in MBPP), prompting it to continue generating coherently into the subsequent block. This enables the flexible, arbitrary-length generation that our method is designed to achieve.

| | | length of generation | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 32 | 64 | 128 | 256 | 512 | 1024 | **Ours** |
| MBPP | Dream-7B | 43.8 | 58.0 | 57.2 | 58.6 | <u>59.6</u> | 59.2 | **60.4** |
| HumanEval | | 26.8 | 43.9 | <u>48.7</u> | 48.1 | 43.9 | 43.9 | **52.4** |
| MBPP | LLaDA-8B | 22.4 | 36.2 | 37.0 | 36.6 | 36.8 | 37.4 | **37.6** |
| HumanEval | | 11.2 | 32.9 | 37.8 | **40.8** | 36.6 | 37.2 | <u>40.2</u> |

Table 1: Test accuracies of Dream-7B and LLaDA-8B on the coding benchmarks. The best results are in bold, and the second-best are underlined.

## 5 Experiments

In this section, we show that the proposed method can improve the sampling quality and faster generation speed for Dream. Then we conduct an ablation study and compare with block generation.

**Setup.** The proposed truncated block generation is used with LLaDA-8B-Instruct (Nie et al., 2025) and Dream-7B-Instruct (Ye et al., 2025). Experiments are performed on the (i) math reasoning datasets: MBPP (Austin et al., 2021b) and HumanEval (Chen et al., 2021), and (ii) code generation datasets: GSM8K (Cobbe et al., 2021) and MATH (Saxton and Hill, 2019). Following LLaDA (Nie et al., 2025), we use 4-shot prompt for GSM8K and Math, 3-shot prompt for MBPP, and 0-shot prompt for HumanEval. All experiments are conducted on NVIDIA A6000 GPUs.

**Baselines.** We compare against the strongest sampling strategies of Dream and LLaDA. Specifically, LLaDA uses confidence-based remasking with semi-autoregressive decoding, while Dream adopts an entropy-based sampler.

### 5.1 Test Accuracy

The LLaDA and Dream baselines require a fixed pre-defined sequence length, and we set this to 32, 64, 128, 256, 512 and 1024. For truncated block generation, we set the block length to 64. and the maximum number of blocks to two.

Table 1 shows the coding results. On both MBPP and HumanEval, using the proposed method with Dream achieves higher accuracies across all fixed-length generations. As the proposed method uses 2 length-64 blocks, this improves the fixed length-128 results by 3.2% on MBPP and 3.7% on HumanEval. Results on the math problems are shown in Table 2. As can be seen, our method also improves the accuracy from 78.01% to 78.92% on GSM8K; and from 42.80% to 43.98% on Math.

Moreover, from Tables 1 and 2, we observe that the proposed algorithm outperforms the baseline on most benchmarks, while achieving comparable accuracies on LLaDA. As discussed in Section 3, this is because LLaDA applies the same penalty weight to all tokens during training and therefore does not suffer from attention dilution.

### 5.2 Inference Speed

We use the same setting of HumanEval in the section above. If we want to generate 128 tokens, fixed-length generation needs to append 128 tokens to the end of the prompt and forward the whole sentence. In contrast, our method uses two generation blocks, each containing 64 masked tokens. If generation terminates after the first block, the fixed-length baseline incurs twice as many forward computations, as it always generates the full 128 tokens. For example, on HumanEval, 36% of the samples complete generation in one stage. Furthermore, longer sequence lengths substantially increase the computational cost of each forward pass. If generation completes in the second round, the computational cost of the first 64 tokens is reduced due to the shorter effective sequence length, while the cost of generating the remaining 64 tokens remains unchanged.

Inference speed is reported as tokens per second (TPS). Table 3 shows that truncated block generation yields 1.9× and 1.6× speedups on HumanEval and MBPP, respectively, with consistent accuracy improvements. Moreover, the proposed method is compatible with existing acceleration techniques, including parallel decoding and block KV cache (Wu et al., 2025). On HumanEval, the Dream baseline generates 13 tokens per second with an accuracy of 48.7%. In contrast, our method nearly doubles the generation speed while improving accuracy to 52.4%. Fast-dLLM (Wu et al., 2025) accelerates decoding via block KV cache and parallel decoding, achieving a 3.5× speedup but at

| | length of generation | GSM8K | | Math | |
|---|---|---|---|---|---|
| | | Flexible-Match | Strict-Match | Exact-Match | Math-Verify |
| Dream-7B | 512 | 75.51 | 75.43 | <u>42.80</u> | 37.38 |
| | 256 | <u>78.01</u> | <u>77.10</u> | 42.70 | <u>37.96</u> |
| | 128 | 66.64 | 58.68 | 35.48 | 35.26 |
| | 64 | 34.87 | 21.98 | 13.34 | 22.10 |
| | 32 | 6.52 | 3.03 | 00.32 | 5.02 |
| | **Ours** | **78.92** | **77.71** | **43.98** | **38.42** |
| LLaDA-8B | 512 | <u>80.21</u> | <u>73.2</u> | 26.76 | <u>29.94</u> |
| | 256 | 79.75 | 54.73 | 28.70 | **31.18** |
| | 128 | 74.22 | 59.96 | **30.86** | 29.52 |
| | 64 | 63.53 | 37.68 | 24.70 | 22.64 |
| | 32 | 21.22 | 11.75 | 4.16 | 18.8 |
| | **Ours** | **80.43** | **73.38** | <u>30.14</u> | 28.92 |

Table 2: Test accuracies of Dream-7B and LLaDA-8B on the math benchmarks. The best results are in bold, and the second-best are underlined.

| | HumanEval | | MBPP | |
|---|---|---|---|---|
| Method | TPS | Acc(%) | TPS | Acc(%) |
| Dream-7B | 13 (1.0×) | 48.7 | 1.58 (1.0×) | 57.2 |
| + Fast-dLLM | 46 (3.5×) | 40.2 | 32.8 (20×) | 55.2 |
| + Ours | 25 (1.9×) | **52.4** | 2.48 (1.6×) | **60.4** |
| + Fast-dLLM + Ours | **57** (4.3×) | 46.3 | **42.2** (26×) | 55.8 |

Table 3: Inference speed and accuracy comparison with the baseline and Fast-dLLM on HumanEval and MBPP.

the cost of a substantial accuracy drop to 40.2%. Combining Fast-dLLM with truncated block generation further improves efficiency while partially recovering accuracy, reaching a 4.3× speedup with 46.3% accuracy.

### 5.3 Multi-Block Generation

To evaluate whether the proposed method remains effective and stable in long-context scenarios that require multiple iterative block extensions, we build on the experiment in Table 1 and extend Dream-7B's generation to up to four blocks (256 tokens). Table 4 shows the test accuracies on the coding benchmarks. As can be seen, when the number of blocks increases from one to two, test accuracies on both MBPP and HumanEval improve significantly from 43.9% and 58.0% to 52.4% and 60.4%, respectively. Performance then saturates at three blocks, as the model has already generated the full content and additional blocks do not provide further gains. This experiment demonstrates that TBG scales effectively with increased iterative block generation, maintaining stable performance without degradation.

### 5.4 Ablation Studies

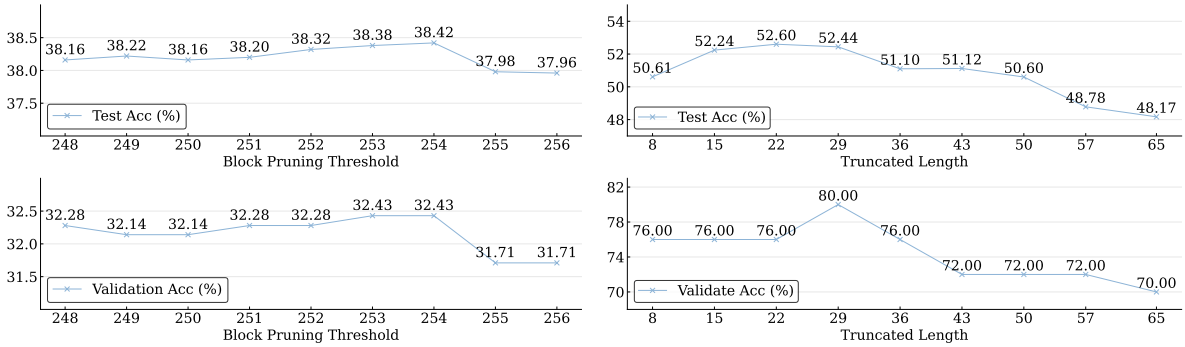
In this section, we perform ablation studies on key hyper-parameters in the proposed method.

#### 5.4.1 Threshold

Figure 5a reports exact-match accuracy of Dream 7B on the Math dataset. We fix the initial generation block length to 256 tokens. The results indicate that performance peaks at an accuracy of 38.42 when the threshold is set to 254, while remaining insensitive to the threshold choice.

### 5.5 Truncated Length

When the truncated length is zero, our algorithm degrades to traditional block generation (Arriola et al., 2025). We further evaluate the effect of different truncated lengths on model performance. Specifically, we conduct experiments on the HumanEval




(a) Ablation study on block pruning threshold

(b) Ablation study on truncated length

Figure 5: Ablation study on threshold and truncated length.

| Task | Model | TBG 1 block | TBG 2 blocks | TBG 3 blocks | TBG 4 blocks |
|---|---|---|---|---|---|
| MBPP | Dream-7B | 58.0 | 60.4 | 60.9 | 60.9 |
| HumanEval | Dream-7B | 43.9 | 52.4 | 53.1 | 53.1 |

Table 4: Performance of Dream-7B under different TBG block settings.

| Dataset | Block | Ours |
|---|---|---|
| HumanEval | 48.1 | 52.4 |
| MBPP | 58.0 | 60.4 |
| Math | 43.6 | 44.0 |
| GSM8K | 66.6 | 78.9 |

Table 5: Accuracy comparison between block generation and the proposed method across four datasets.

dataset and visualize the results in Figure 5b. On the validation set, accuracy initially increases with truncated length and then decreases. Both the validation and test sets achieve the highest score when the truncated length is set to 30.

## 6 Related Works

**Diffusion Language Models:** Diffusion models have achieved great success in the continuous domain (Sohl-Dickstein et al., 2015; Ho et al., 2020; Karras et al., 2022). A simple approach is to map tokens into continuous embeddings and perform diffusion process in the continuous space (Li et al., 2022; Han et al., 2022; Mahabadi et al., 2023). Alternatively, some methods directly train a discrete diffusion models on the discrete vocabulary space (Sohl-Dickstein et al., 2015; Austin et al., 2021a; Lou et al., 2023; Nie et al., 2025; Ye et al., 2025). In this formulation, the diffusion model's forward steps progressively map original tokens to [MASK] tokens or random tokens, which corresponds to absorbing diffusion kernel and uniform diffusion kernel, and the reverse process reconstructs the original text from these noised sequences. Consequently, many recent works scale masked diffusion models to billion-parameter scale (Nie et al., 2025; Ye et al., 2025; Khanna et al., 2025; Zhu et al., 2025). All of these dLLMs adopt the absorbing diffusion kernel, which maps original tokens to [MASK] tokens in the forward process. LLaDA series (Nie et al., 2025; Zhu et al., 2025) trained diffusion models from scratch using direct mask prediction and sentence level noise loss reweighting. The Dream series (Xie et al., 2025; Ye et al., 2025) use ARMs for model initialization and train diffusion models using shift mask prediction and token level noise loss reweighting.

**Inference Remasking Strategies for dLLMs.** Diffusion large language models inference are based on low-confidence remasking (Zhu et al., 2025; Nie et al., 2025; Ye et al., 2025). Specifically, similarly to (Chang et al., 2022), they remask $t/s$ of the predicted tokens with lowest confidence based on the predictions, called low-confidence remasking. Instead of the low-confidence remasking strategy, Zhou et al. (2026) and Kim et al. (2025) propose to use the top-attention and top-probability margin unmasking strategy, respectively, and obtain increased performance on several benchmarks.

**Block Generation in dLLMs.** Block generation or semi-AR generation are widely used in current diffusion language models (Arriola et al., 2025; Nie et al., 2025; Zhu et al., 2025; Wu et al., 2025). Arriola et al. (2025) proposed a block-wise extension of the D3PM framework (Austin et al., 2021a) to generate arbitrary-length sequences. LLaDA (Zhu et al., 2025; Nie et al., 2025) also adopts this strategy in their models. Wu et al. (2025) introduces KV-cache in their block-wise decoding.

**Attention patterns:** Attention patterns such as "attention drift" (Wang et al., 2025b) and "lost in the middle" (Liu et al., 2024b) indicate that language models may struggle to attend to the correct tokens in long sequences. Attention drift describes the tendency of LLMs to gradually shift their focus toward newly generated tokens, at the expense of the initial input context. Lost in the middle describes the phenomenon that long-context language models use information least effectively when it is positioned in the middle of the input, while showing better performance when the relevant content appears near the beginning or the end. While attention drift and lost in the middle primarily describe the attention patterns that ARMs attend to (NON-MASK) tokens in long sequences, "attention dilution" identifies a distinct challenge unique to diffusion LLMs. Specifically, attention dilution refers to performance degradation caused by an excessive number of [MASK] tokens. In this scenario, the uninformative [MASK] tokens dilute the attention scores, making it difficult for the model to extract a clear signal from the informative context.

## 7 Conclusion

In this paper, we provide empirical and theoretical evidence that excessive redundant mask tokens will dilute the contextual attention of dLLMs and degrade its performance. We also show that both the contextual mask tokens of Dream and all the mask tokens of LLaDA will copy a token from the prompt as query at predict the token. Inspired by this observation, we propose truncated block generation for diffusion language model sampling, which leads to faster generation speed and high quality generation, and supports flexible generation. We conduct extensive experiments to visualize our observation and validate effectiveness of the algorithm.

## 8 Limitation

Similar to existing semi-autoregressive decoding methods, our approach relies on several hyperparameters. Automatically tuning the generation threshold and truncation length may further improve performance. We leave this exploration to future work.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *Techreport, arXiv:2303.08774*.

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

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. *NeurIPS*.

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

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. 2022. Maskgit: Masked generative image transformer. In *CVPR*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *Techreport*, pages arXiv–2407.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. 2022. Ssd-lm: Semi-autoregressive simplex-based

diffusion language model for text generation and modular control. *Techreport, arXiv:2210.17432*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *NeurIPS*.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. 2022. Elucidating the design space of diffusion-based generative models. *NeurIPS*.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, and 1 others. 2025. Mercury: Ultra-fast language models based on diffusion. *Techreport, arXiv:2506.17298*.

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

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. 2022. Diffusion-lm improves controllable text generation. *NeurIPS*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *Techreport, arXiv:2412.19437*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024b. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

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

Rabeeh Karimi Mahabadi, Hamish Ivison, Jaesung Tae, James Henderson, Iz Beltagy, Matthew E Peters, and Arman Cohan. 2023. Tess: Text-to-text self-conditioned simplex diffusion. *Techreport, arXiv:2305.08379*.

Chenlin Meng, Kristy Choi, Jiaming Song, and Stefano Ermon. 2022. Concrete score matching: Generalized score matching for discrete data. *NeurIPS*.

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

Grefenstette Saxton and Kohli Hill. 2019. Analysing mathematical reasoning abilities of neural models. *arXiv:1904.01557*.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*.

Bingning Wang, Haizhou Zhao, Huozhi Zhou, Liang Song, Mingyu Xu, Wei Cheng, Xiangrong Zeng, Yupeng Zhang, Yuqi Huo, Zecheng Wang, and 1 others. 2025a. Baichuan-m1: Pushing the medical capability of large language models. *Techreport, arXiv:2502.12671*.

Pinzheng Wang, Zecheng Tang, Keyan Zhou, Juntao Li, Qiaoming Zhu, and Min Zhang. 2025b. Revealing and mitigating over-attention in knowledge editing. *Techreport, arXiv:2502.14838*.

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

Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, and 1 others. 2025. Dreamcoder 7b: An open diffusion language model for code. *Techreport, arXiv:2509.01142*.

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

Yuyan Zhou, Kai Syun Hou, Weiyu Chen, and James Kwok. 2026. Attention-based sampler for diffusion language models. *Techreport, arXiv:2604.08564*.

Yuyan Zhou, Liang Song, Bingning Wang, and Weipeng Chen. 2024. Metagpt: Merging large language models using model exclusive task arithmetic. In *EMNLP*.

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

# Appendix

## A Statistics For Data

Table 6 shows the details of the datasets we use in our paper.

## B Proof of Theorem 1

Since $\frac{w(X_1,j+1)}{w(X_1,j)} = p$, we have $\frac{w(X_1,,j+k)}{w(X_1,j,n)} = p^k$. We denote $w(X_1,,c+1) = C$. Then, for all $\epsilon > 0$, there exists $k = \log_p \frac{\epsilon}{C}$, such that for all $u > k$, $w(X_1,,u) < w(X_1,k,n) = Cp^k \leqslant \epsilon$.

## C Proof of Theorem 2

For the $X_1$ defined in Section 3, we further append $m_2 - m_1$ mask tokens to its end and denote it as $X_2$:

$$X_1 = \begin{bmatrix} P_0 \\ M_1 \end{bmatrix} \in \mathbb{R}^{1\times(c+n_1)\times d} \tag{6}$$

$$X_2 = \begin{bmatrix} P_0 \\ M_1 \\ M_2 \end{bmatrix} \in \mathbb{R}^{1\times(c+n_1+n_2)\times d}, \tag{7}$$

where $n_1 = m_1$ and $n_2 = m_2 - m_1$. The query of $X_1$ and $X_2$ can be calculated as:

$$Q_1 = X_1 W_Q = \begin{bmatrix} P_0 W_Q \\ M_1 W_Q \end{bmatrix} \tag{8}$$

$$Q_2 = X_2 W_Q = \begin{bmatrix} P_0 W_Q \\ M_1 W_Q \\ M_2 W_Q \end{bmatrix} \tag{9}$$

The key of $X_1$ and $X_2$ can be calculated as:

$$K_1 = X_1 W_K = \begin{bmatrix} P_0 W_K \\ M_1 W_K \end{bmatrix} \tag{10}$$

$$K_2 = X_2 W_K = \begin{bmatrix} P_0 W_K \\ M_1 W_K \\ M_2 W_K \end{bmatrix} \tag{11}$$

By multiplying the key and value, we have:

$$A^1 = \begin{bmatrix} P_0 W_Q \\ M_1 W_Q \end{bmatrix} \left( \begin{bmatrix} P_0 W_K \\ M_1 W_K \end{bmatrix} \right)^\top \tag{12}$$

$$= \begin{bmatrix} P_0 W_Q W_K^\top P_0 & P_0 W_Q W_K^\top M_1 \\ M_1 W_Q W_K^\top P_0 & M_1 W_Q W_K^\top M_1 \end{bmatrix} \tag{13}$$

$$A^2 = \begin{bmatrix} P_0 W_Q \\ M_1 W_Q \\ M_2 W_Q \end{bmatrix} \left( \begin{bmatrix} P_0 W_K \\ M_1 W_K \\ M_2 W_K \end{bmatrix} \right)^\top \tag{14}$$

$$= \begin{bmatrix} P_0 W_Q W_K^\top P_0 & P_0 W_Q W_K^\top M_1 & P_0 W_Q W_K^\top M_2 \\ M_1 W_Q W_K^\top P_0 & M_1 W_Q W_K^\top M_1 & M_1 W_Q W_K^\top M_2 \\ M_2 W_Q W_K^\top P_0 & M_2 W_Q W_K^\top M_2 & M_1 W_Q W_K^\top M_2 \end{bmatrix} \tag{15}$$

After softmax, we can get the attention score:

$$\text{Softmax}(A^1)_{kz} = \frac{\exp(A^1_{kc})}{\sum_{i=1}^{c+n_1} A^1_{ki}} \tag{16}$$

$$\text{Softmax}(A^2)_{kz} = \frac{\exp(A^2_{kz})}{\sum_{i=1}^{c+n_1+n_2} \exp(A^2_{ki})} \tag{17}$$

where $z \in [0, c]$, $k \in [c, n_1]$It is easy to show that

$$\text{Softmax}(A^2)_{kz} \tag{18}$$

$$= \frac{\exp(A^2_{kz})}{\sum_{i=1}^{c+n_1+n_2} \exp(A^2_{ki})} \tag{19}$$

$$= \frac{\exp(A^1_{kz})}{\sum_{i=1}^{c+n_1} \exp(A^2_{ki}) + \sum_{i=c+n_1}^{n_2} \exp(A^2_{ki})} \tag{20}$$

$$= \frac{\exp(A^1_{kz})}{\sum_{i=1}^{c+n_1} \exp(A^1_{ki}) + \sum_{i=c+n_1}^{n_2} \exp(A^1_{ki})} \tag{21}$$

$$\frac{\exp(A^1_{k1})}{\sum_{i=1}^{c+n_1} \exp(A^1_{ki})} \tag{22}$$

$$= \text{Softmax}(A^1_{kz}) \tag{23}$$

Thus, we have:

$$\text{CA}(\boldsymbol{m}_i^1) \tag{24}$$

$$= \sum_{j=1}^{c+m_1} \frac{\exp(A^1_{kc})}{\sum_{i=1}^{c+n_1} \exp(A^1_{ki})} \tag{25}$$

$$> \sum_{j=1}^{c+m_1} \frac{\exp(A^1_{k1})}{\sum_{i=1}^{c+n_1} \exp(A^1_{ki})} \tag{26}$$

$$= \text{CA}(\boldsymbol{m}_i^2) \tag{27}$$

## D Detailed experiment setting

For Dream model, when sampling, we set dtype as “bfloat16", temperature as 0.1,top_p as 0.9 and alg as "entropy". For Fast-dLLM, we set the block of KV-cache as 32.

For our truncated block generation, as shown in Table 7, for MBPP dataset, we set the block length as 64, truncated length as 32, and threshold as 55. For HumanEval dataset, we set the block length as 64, truncated length as 32, and threshold as 55. For GSM8K dataset, we set the block length as 128, truncated length as 64, and threshold as 127. For Math dataset, we set the block length as 256, truncated length as 128, and threshold as 255.

| Dataset | Number of Training Examples | Number of Validation Examples | Number of Testing Examples |
|---|---|---|---|
| GSM8k | 7,473 | N/A | 1,319 |
| Math | 7,500 | N/A | 1,500 |
| MBPP | 374 | 30 | 500 |
| HumanEval | N/A | N/A | 164 |

Table 6: Details of datasets we used for our evaluation.

| | MBPP | HumanEval | GSM8K | Math |
|---|---|---|---|---|
| Block length | 64 | 64 | 128 | 256 |
| Truncated length | 32 | 32 | 64 | 128 |
| Threshold | 55 | 55 | 127 | 255 |

Table 7: Detailed hyper-parameters setting.

## E Detailed comparison of different sampling methods

In this section, we provide a detailed comparison of different methods on LLaDA using the HumanEval dataset and present the results in Table 8. We compare our approach against confidence-based remasking with and without fixed-length semi-AR generation, as well as block-wise sampling (Arriola et al., 2025). With a fixed generation length of 128, LLaDA with confidence-based remasking alone achieves 8.5 accuracy, while adding semi-autoregressive sampling improves the accuracy to 37.0. Block diffusion sampling attains 17.1 accuracy. Our method using 64+64 blocks achieves 37.6 accuracy, outperforming all of the above baselines.

| Method | HumanEval |
|---|---|
| Block diffusion | 17.1 |
| confidence remasking w/o semi-AR | 8.5 |
| confidence remasking w/ semi-AR | 37.0 |
| Ours | 37.6 |

Table 8: Detailed comparison of different sampling methods

## F General task

For a general ability testing, We tested BBH (Suzgun et al., 2023) and present the results in Table 9, which consists of 23 particularly challenging BIG-Bench tasks spanning traditional NLP, mathematics, commonsense reasoning, and question answering. We can see that our method also lead to better performance.

## G Detailed Truncated Block Generation Algorithm

In this section, we present the detailed generation process of truncated block generation algorithm in Algorithm 1.

## H Case Study

In this section, we visualize the output of block generation and truncated block generation. As shown in Fig. 6, we can see that as generation approaches the end of a block, the model is strongly biased toward finishing the generation and output tokens such as "EOS", "DONE" and "return true". The truncation is used to remask the EOS and part of the previous tokens and enable the model to produce longer and more reasonable answer. Because the model retains knowledge of the previously generated text, it can reliably regenerate the masked tokens without losing global coherence.

## I The use of Large Language Models

We used LLMs for grammar checking and wording improvement, ensuring it did not alter the text's meaning or add references.

| Method | Semi-AR + Confidence based remasking | | Ours |
|---|---|---|---|
| Gen Len | 128 | 256 | 128 + 128 |
| Acc(%) | 50.68 | 56.60 | 57.12 |

Table 9: Accuracy of LLaDA-8B on BBH benchmarks.

```
Prompt: Write a python function to find
the count of rotations of a binary
string with odd value.
[BEGIN]

First round generation:
def odd_Equivalent(s, n):
    count = 0
    m = len(s)
    for i in range(n):
        rotated = s[i:] + s[:i]
        if rotated.count('1') % 2 == 1:
            count += 1
    return count
[DONE]<|endoftext|><|endoftext|><|endof
text|><|endoftext|>

Second round generation:
def odd_Equivalent(s, n):
    count = 0
    m = len(s)
    for i in range(n):
        rotated = s[i:] + s[:i]
        if rotated.count('1') % 2 == 1:
            count += 1
    return count
[DONE]<|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|>......
```

(a) block generation

```
Prompt: Write a python function to find
the count of rotations of a binary
string with odd value.
[BEGIN]

First round generation:
def odd_Equivalent(s, n):
    count = 0
    m = len(s)                          ---> Keep
    for i in range(n):
        rotated = s[i:] + s[:i]
        if rotated.count('1') % 2 == 1:
            count += 1                  ---> Delete
    return count
[DONE]<|endoftext|><|endoftext|><|endof
text|><|endoftext|>

Second round generation:
def odd_Equivalent(s, n):
    count = 0                           ---> Generated in
    m = len(s)                               first round
    for i in range(n):
        rotated = s[i:] + s[:i]
        if rotated[-1] == '1':
            count += 1
    return count                        ---> Generated in
                                             second round
# Test cases
assert odd_Equivalent("011001", 6) == 3
assert odd_Equivalent("11011", 5) == 4
assert odd_Equivalent("1010", 4) == 2
[DONE]<|endoftext|><|endoftext|><|endof
text|><|endoftext|><|endoftext|>......
```

(a) truncated block generation

Figure 6: Illustrative example comparing block generation with truncated block generation.

**Algorithm 1** Truncated Block Generation

**Require:** Prompt $c$, model $f_{\boldsymbol{\theta}}$, threshold $\gamma_1, \gamma_2, \cdots, \gamma_k$, block length $b_1, b_2 \cdots, b_k$, truncated length $l_1, l_2, \cdots, l_k$, sampling step $N$.

Set $r^1$ is a fully masked sequence of length $L$ at time step 1.
**for** $j \leftarrow 1$ to $k$ **do** ▷ Generate $j$ th round.
  **for** $t \leftarrow 1$ **down to** $\frac{1}{N}$ **step** $\frac{1}{N}$ **do** ▷ Iterate through all time steps.
    $s = t - \frac{1}{N}$ ▷ Calculate previous timestep: $s = t - 1/N$.
    **for** $i \leftarrow 1$ to $L$ **do** ▷ Iterate through each token $i$ in the sequence (1 to $L$).
      **if** $r_t^i \neq M$ **then** ▷ If token $i$ at timestep $t$ is not masked.
        $r_0^i = r_t^i, c^i = 1$ ▷ Keep the token unchanged and set confidence to 1.
      **else** ▷ If token $i$ is masked.
        $r_0^i = \arg\max_{r_0^i} p_\theta(r_0^i | p_0, r_t)$ ▷ Predict the most likely token for this token.
        $c^i = p_\theta(r_0^i | p_0, r_t)_{r_0^i}$ ▷ Record the score (conf. / entropy.) of this token.
      **end if**
    **end for**
    $n_{\text{un}} = \lfloor L(1-s) \rfloor$ ▷ The number of unmasked tokens is $n_{\text{un}}$ in timestep $s$
    **for** $i \leftarrow 1$ to $L$ **do**
      **if** $c^i \in \text{Lowest} - n_{\text{un}}(\{c^i\}_{i=1}^L)$ **then** ▷ If confidence of token $i$ is low.
        $r_0^i = M$ ▷ Remask this token and select it for remasking.
      **end if**
    **end for**
    $r_s = r_0$ ▷ Update the sequence state.
  **end for**
  **if** len$(r_s) \leqslant \gamma_k$ **then** ▷ If the valid length is less than a threshold.
    Break ▷ Break the loop and output.
  **else**
    $r_0 = \text{Cat}\left(r_0[0 : l_j], [\texttt{MASK}] \times b_j\right)$ ▷ Remasking and appending mask.
  **end if**
**end for**
**return** $r_0$