IMPROVING PARALLEL DECODING OF BLOCK DIFFUSION LANGUAGE MODELS WITH AUTOREGRESSIVE GENERATION ORDER

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion language models enable flexible arbitrary-order generation, but existing token sampling methods are mostly designed for early masked diffusion models (MDMs). In this work, we study token sampling for recent block diffusion language models (BDLMs). We show empirically and analytically that these models are naturally more aligned with left-to-right decoding than MDMs. Beyond their AR-pretrained initialization, we attribute this alignment to the block diffusion training objective, which more frequently exposes the model to the left-to-right context used during AR decoding. Based on this observation, we propose Parallel Autoregressive Decoding (PARD), a simple training-free sampling method that preserves left-to-right unmasking structure while allowing parallel token commitment. Experiments show that PARD consistently improves the speed-quality trade-off over existing parallel samplers. Across three recent BDLMs and six benchmarks, it achieves the best average generation quality among parallel samplers. Compared with pure AR decoding, PARD achieves substantial speedups with only a small quality gap.

1 INTRODUCTION

Recently, diffusion language models (DLMs) have emerged as a promising alternative to autoregressive language models (ARMs) for text generation (Li et al., 2025). In particular, masked diffusion models (MDMs) have attracted growing attention for their simple denoising formulation and strong empirical performance (Sahoo et al., 2024; Shi et al., 2024). MDMs iteratively reconstruct a sequence from a partially masked state, enabling tokens to be generated in arbitrary orders and in parallel. To further improve decoding quality and efficiency, this arbitrary-order generation flexibility has motivated a growing line of token sampling methods that seek to decide which masked positions should be unmasked at each denoising step (Ye et al., 2025; Kim et al., 2025a). More recent methods further accelerate decoding through parallel sampling, committing multiple positions per step based on criteria designed to balance generation speed and quality (Wu et al., 2025; Kim et al., 2025b).

Early MDMs, such as LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025), apply masked diffusion over the entire sequence. In contrast, Block diffusion language models (BDLMs) are trained to factorize generation autoregressively across blocks, while applying masked diffusion within each block conditioned on the preceding clean blocks (Arriola et al., 2025). This setting is increasingly important, as block diffusion enables exact KV cache reuse across completed blocks and variable-length generation. Many recent high-performing DLMs adopt block diffusion, often by continuing training from pretrained ARMs with a block diffusion objective (Wu et al., 2026b; Cheng et al., 2025; Bie et al., 2025; 2026; Wu et al., 2026a; Fu et al., 2026). However, existing token sampling methods have mostly been developed and benchmarked on early MDMs (Kim et al., 2025b; Wu et al., 2025; Ben-Hamu et al., 2025), leaving their behavior on these recent BDLMs underexplored.

To investigate this gap, we compare the decoding orders of recent BDLMs and MDMs, and their performance under pure AR sampling. We find that BDLMs exhibit a stronger AR bias in their decoding orders and benefit more consistently from pure AR sampling, suggesting that they are more naturally aligned with AR decoding. Besides the AR-pretrained initialization, we further explain this behavior by comparing the training-time input contexts of MDM and block diffusion. We show that MDM training almost never exposes the model to the left-to-right context used by AR decoding. In

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

contrast, block diffusion training observes such contexts with larger and non-negligible probability, and it biases the overall training context toward the left-to-right context. This training-time bias makes BDLMs naturally more aligned with AR decoding.

These findings suggest that token sampling in BDLM should preserve the AR structure. However, pure AR decoding generates only one token per model forward pass, making it slower than parallel token samplers. To address this problem, we propose *Parallel Autoregressive Decoding (PARD)*, a simple training-free sampling strategy that adapts existing arbitrary-order parallel token samplers to left-to-right decoding. PARD unmasks the longest prefix of positions accepted by a given criterion, preserving AR structure while retaining parallelism. Experiments on three recent BDLMs show that PARD consistently outperforms existing parallel token samplers, achieving stronger generation quality while preserving substantial decoding speedups.

2 Related Work

2.1 Diffusion Language Models

Diffusion language models (DLMs) have become popular for text generation (Campbell et al., 2022; Austin et al., 2021a; Lou et al., 2023; Ou et al., 2025). Masked Diffusion Models (MDMs) are a particularly representative class among them (Sahoo et al., 2024; Shi et al., 2024). Recent work has scaled MDMs to billions of parameters (Nie et al., 2025; Zhu et al., 2025), showing that DLMs can achieve generation quality competitive with ARMs while offering the potential for faster inference (Song et al., 2025; Khanna et al., 2025). MDM treats the whole sequence as a single diffusion state, so each denoising step processes all token positions. At inference time, MDMs can use semi-AR decoding, which partitions the preallocated output sequence into blocks and restricts unmasking to positions within the current block. Block diffusion language models (BDLMs) instead factorize generation autoregressively over blocks, applying a masked diffusion objective within each block conditioned on the clean preceding blocks (Arriola et al., 2025). This enables exact KV cache reuse for finalized blocks, avoiding the full-sequence forward passes required by MDMs, and naturally supports variable-length generation. These advantages have motivated many recent foundation BDLMs (Wu et al., 2026b; Cheng et al., 2025; Bie et al., 2025; 2026; Tian et al., 2025; Wu et al., 2026a; Fu et al., 2026). They are often adapted from strong ARMs through continued pretraining or fine-tuning with the block diffusion objective, and achieve stronger generation quality and inference efficiency than early MDMs.

2.2 Token Sampling Methods for DLMs

Early MDMs use uniform sampling, which selects the next unmasked position uniformly from the remaining masked positions (Austin et al., 2021a). While aligned with the MDM training distribution, this strategy is often suboptimal for generation quality, motivating heuristic samplers based on model-dependent scoring rules. Confidence selects the position with the highest predicted probability (Chang et al., 2022). Margin selects the position with the largest gap between the top-1 and top-2 predicted probabilities (Kim et al., 2025a). Entropy selects the position with the lowest predictive entropy (Ye et al., 2025). Uncode (Huang et al., 2026) augments confidence with a position-aware prior and an informativeness term that down-weights generic high-frequency tokens.

Recent works have also developed more advanced parallel sampling methods that trade off generation quality and inference speed. Representative approaches include confidence-thresholded unmasking (Wu et al., 2025; Yu et al., 2025), KL-guided stability-based selection (Kim et al., 2025b), entropy-bounded subset selection (Ben-Hamu et al., 2025), and tree-structured hierarchical decoding (Qi et al., 2026). APD (Israel et al., 2025) instead uses a small auxiliary autoregressive model to verify parallel DLM proposals.

3 Understanding the AR Bias of BDLMs

In Section 3.1, we first empirically show that BDLMs are more strongly aligned with AR decoding than MDMs, based on their decoding orders and performance under pure AR sampling. Section 3.2 then analyzes how their training contribute to this stronger alignment.

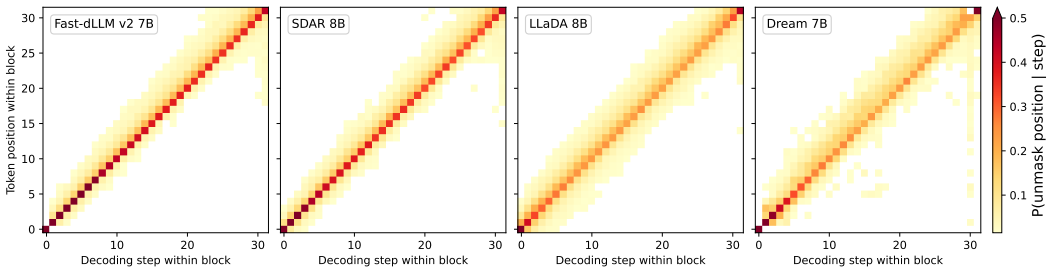

Figure 1: Distributions of unmasking positions at various denoising steps for Fast-dLLM v2 7B, SDAR 8B, LLaDA 8B, and Dream 7B on *HumanEval*.

3.1 AR Bias in BDLM Decoding

We examine the decoding orders induced by the arbitrary-order sampler based on Confidence. We compare two recent BDLMs, Fast-dLLM v2 7B (Wu et al., 2026b) and SDAR 8B (Cheng et al., 2025), with two representative MDMs, LLaDA 8B (Nie et al., 2025) and Dream 7B (Ye et al., 2025). All models are evaluated on *HumanEval*. For the MDMs, we employ the semi-AR decoding strategy. We use 32 denoising steps with block size 32 for all models, so one token is unmasked at each step.

Figure 1 shows the distribution of unmasking positions at each denoising step. A strictly autoregressive order would place all probability mass on the diagonal. As can be seen, all four models exhibit a concentration around the diagonal, indicating a tendency toward left-to-right decoding. However, this concentration is stronger for the two BDLMs. The average probability mass within ± 2 positions of the diagonal is 0.77 and 0.75 for Fast-dLLM v2, and SDAR, respectively, versus 0.70 and 0.69 for LLaDA and Dream. The contrast is more pronounced on the diagonal, with average masses of 0.45 and 0.39 for Fast-dLLM v2 and SDAR, compared with 0.27 for LLaDA and 0.30 for Dream. These results indicate a stronger AR bias in the decoding orders of the BDLMs.

To further quantify the decoding order's closeness to AR, we compute the two AR-ness@k metrics proposed in Gong et al. (2026): (i) local AR-ness@k, which measures the fraction of steps that extend a k-long consecutive run; and (ii) global AR-ness@k, which measures the fraction of steps where the selected position is among the k earliest remaining masked positions. Detailed definitions of the metrics are in Appendix A. Since blocks are generated autoregressively, we compute the average AR-ness metrics over all blocks. As shown in Figure 2, Fast-dLLM v2 and SDAR exhibit consistently higher local and global AR-ness across different k's, again confirming that BDLM is more AR-like than MDM on decoding.

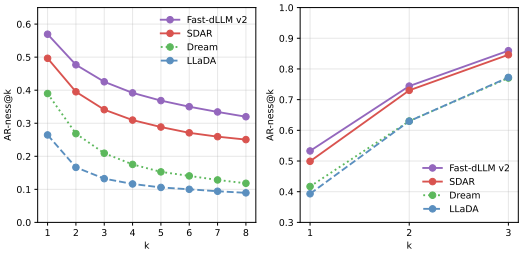

Figure 2: Local (left) and global (right) AR-ness@k of decoding on *HumanEval*.

Recent works (Gong et al., 2026; Li et al., 2026) have also observed that decoding in MDMs often exhibits strong left-to-right structure despite their arbitrary-order flexibility. Ni et al. (2026) further show that enforcing AR order during RL rollouts can improve the reasoning performance of LLaDA. However, it remains unclear whether explicit AR sampling at inference time is useful for MDM and BDLM. We therefore compare three representative arbitrary-order samplers (Confidence, Margin, and Entropy) with a pure AR sampler (which always selects the leftmost masked position at each step). Table 1 shows the pass@1 values on *HumanEval* for the four models with different samplers. Pure AR achieves the highest pass@1 on both BDLMs. In contrast, the two MDMs show smaller or negative performance changes under AR sampling. On LLaDA, pure AR performs worse than all arbitrary-order samplers, falling 4.2 points below the best. On Dream, pure AR achieves the highest

Table 1: *HumanEval* base pass@1 (%) under arbitrary-order sampling and pure AR sampling.

Method	Fast-dLLM v2 7B	SDAR 8B	LLaDA 8B	Dream 7B
Confidence	65.9	75.0	**45.7**	51.8
Entropy	65.9	77.4	**45.7**	50.6
Margin	62.8	71.3	43.3	45.7
AR	**72.0**	**79.3**	41.5	**52.4**

pass@1, suggesting greater AR alignment than LLaDA, consistent with Dream's AR initialization. However, its gap over the best arbitrary-order sampler is only 0.6 points, compared with 6.1 points for Fast-dLLM v2 and 1.9 points for SDAR. These results suggest that BDLMs benefit more from a strict AR order than MDMs, including the AR-initialized MDM Dream. In the following, we explore why BDLMs are better aligned with AR sampling.

3.2 Why Are BDLMs More Aligned With AR?

Recent BDLMs are obtained by fine-tuning or continual pretraining from AR-pretrained LLMs (Wu et al., 2026b; Cheng et al., 2025). Compared with the AR-pretraining stage, which optimizes next-token prediction over trillions of tokens, block diffusion adaptation typically uses a much smaller token budget. For example, Qwen2.5 7B is pretrained on 18T tokens (Yang et al., 2024), while Fast-dLLM v2 adapts it to a BDLM with only 1B tokens of fine-tuning. This large training-budget gap suggests that block diffusion adaptation does not fully overwrite the left-to-right conditional behavior learned during AR pretraining.

Beyond initialization, the block diffusion training objective also gives rise to input contexts better matched to AR decoding. During block diffusion training, when the model predicts tokens in block b, it conditions on the clean prefix from all previous blocks and only the current block is partially masked. Future blocks are not included in the input. In contrast, MDM training masks positions across the whole sequence, which can leave the model with a discontinuous left context while also exposing tokens to the right. To formally quantify this training-context difference, we first introduce the notation and training objectives for MDM and block diffusion.

Notation. Let V be the vocabulary, `[MASK]` be the absorbing mask token, and $\bar{V} = V \cup \{\texttt{[MASK]}\}$. A clean sequence $x_0 \in V^L$ is drawn from the data distribution p_{data}. The sequence is partitioned into B contiguous blocks, each of size L'. We denote each block as $x_0^b \in V^{L'}$ ($b \in \{1, ..., B\}$), and write $x_0^{<b} = (x_0^1, \ldots, x_0^{b-1})$ for the prefix before block b. We denote $p_\theta(x_0^\ell \mid x_t)$ as the denoising model's predictive distribution at position ℓ, conditioned on the noised input sequence x_t.

MDM. During training, the forward process first samples a noise level $t \sim U[0, 1]$, then masks each position ℓ independently to obtain the noised sequence $x_t \in \bar{V}^L$ according to

$$q_t(x_t^\ell \mid x_0^\ell) = \text{Cat}\Big(x_t^\ell; \alpha_t \mathbf{e}_{x_0^\ell} + (1 - \alpha_t)\mathbf{e}_{\texttt{[MASK]}}\Big), \tag{1}$$

where α_t is a non-increasing noise schedule, and $\mathbf{e}_v$ is the one-hot vector for token $v \in \bar{V}$. Let $\mathcal{M}(x_t) = \{j : x_t^j = \texttt{[MASK]}\}$ be the masked positions in x_t. With the standard linear schedule $\alpha_t = 1 - t$, MDM is trained to predict the original tokens at masked positions (Sahoo et al., 2024):

$$\mathcal{L}_{\text{MDM}} = -\mathop{\mathbb{E}}_{t\sim U[0,1],\, x_t\sim q_t(\cdot|x_0)} \frac{1}{t} \sum_{j\in\mathcal{M}(x_t)} \log p_\theta(x_0^j \mid x_t).$$

Block Diffusion. Block diffusion factorizes the sequence likelihood autoregressively over blocks: $p_\theta(x_0) = \prod_{b=1}^B p_\theta(x_0^b \mid x_0^{<b})$ (Arriola et al., 2025). For each block b, the forward noising process first samples a noise level $t \sim U[0, 1]$, then obtain x_t^b by position-independent masking as (1). With the linear schedule $\alpha_t = 1 - t$, the block diffusion training objective can be written as

$$\mathcal{L}_{\text{Block}} = -\sum_{b=1}^{B} \mathbb{E}_{t\sim U[0,1],\, x_t^b\sim q_t(\cdot|x_0^b)} \left[\frac{1}{t} \sum_{j\in\mathcal{M}(x_t^b)} \log p_\theta(x_0^{b,j} \mid x_0^{<b}, x_t^b)\right],$$

where $x_0^{b,j}$ is the j-th token in block b.

We now compare the training contexts induced by these two objectives. Consider a position ℓ. Let N denote the length of an input context, with $N = L$ for MDM and $N = bL'$ for block diffusion when position ℓ lies in block b. We define the AR mask pattern $\mathbf{m}_\ell^{\text{AR}} \in \{0, 1\}^N$ as

$$(\mathbf{m}_\ell^{\text{AR}})_j = \begin{cases} 0, & j < \ell, \\ 1, & \ell \le j \le N, \end{cases}$$

i.e., the left prefix has already been unmasked and the positions from ℓ onward are not visible. The following Proposition compares how often MDM and BDLM see this AR mask pattern during training. The proof is in Appendix B.1.

Proposition 1 (Probability of seeing AR mask pattern)**.** *Assume linear noise schedule $\alpha_t = 1 - t$ with $t \sim U[0,1]$. For MDM, $\Pr\big(\mathbf{m}_\ell^{\mathrm{AR}}$ observed$\big) = \frac{1}{(L+1)\binom{L}{\ell-1}}$; and for block diffusion, $\Pr\big(\mathbf{m}_\ell^{\mathrm{AR}}$ observed$\big) = \frac{1}{(L'+1)\binom{L'}{k}}$, where $k = \ell - 1 - (b-1)L'$ is the number of positions preceding ℓ within block b.*

For example, for MDM with sequence length $L = 1024$, the probability is approximately 10^{-310} when $\ell = 512$, and approximately 10^{-6} when $\ell = 1024$ (at the edge). Hence, AR contexts are almost absent during MDM training. In contrast, for block diffusion, this probability is always larger with the same ℓ. For example, with block size $L' = 32$, it is about 10^{-11} at $k = 15$; about 0.03 at $k = 0$, and about 0.001 at the second ($k = 1$) and last positions ($k = 31$) in each block. Hence, during training, BDLMs receive more frequent supervision under contexts that match those queried by AR decoding. AR decoding thus creates a smaller training-inference shift for BDLMs than for MDMs.

Besides studying the probability that the AR mask pattern is exactly observed, we can also measure how a training mask pattern $\mathbf{m}$ (with $\mathbf{m}_j = 1$ if position j is masked, and 0 otherwise) aligns with $\mathbf{m}_\ell^{\mathrm{AR}}$ at position ℓ. For MDM, $\mathbf{m}^{\mathrm{MDM}} \in \{0,1\}^L$, and the AR pattern alignment can be defined as

$$\rho_\ell(\mathbf{m}^{\mathrm{MDM}}) = \frac{1}{L-1}\left[\sum_{j=1}^{\ell-1}(1 - \mathbf{m}_j^{\mathrm{MDM}}) + \sum_{j=\ell+1}^{L}\mathbf{m}_j^{\mathrm{MDM}}\right];$$

whereas for block diffusion (with position ℓ contained in block b), $\mathbf{m}^{\mathrm{BDLM}} \in \{0,1\}^{bL'}$ and the AR pattern alignment is

$$\rho_\ell(\mathbf{m}^{\mathrm{BDLM}}) = \frac{1}{bL'-1}\left[\sum_{j=1}^{\ell-1}(1 - \mathbf{m}_j^{\mathrm{BDLM}}) + \sum_{j=\ell+1}^{bL'}\mathbf{m}_j^{\mathrm{BDLM}}\right].$$

This alignment measures the fraction of positions where the training mask agrees with the AR pattern at position ℓ. The following Proposition shows that block diffusion has a higher expected AR pattern alignment than MDM. The proof is in Appendix B.2.

Proposition 2 (Expected AR pattern alignment)**.** *Assume the linear noise schedule $\alpha_t = 1 - t$ with $t \sim U[0,1]$. For MDM, the expected AR pattern alignment is $\mathbb{E}[\rho_\ell(\mathbf{m}^{\mathrm{MDM}})] = 1/2$. For block diffusion,*

$$\mathbb{E}[\rho_\ell(\mathbf{m}^{\mathrm{BDLM}})] = \frac{1}{2} + \frac{(b-1)L'}{2(bL'-1)} \geq \mathbb{E}[\rho_\ell(\mathbf{m}^{\mathrm{MDM}})],$$

where expectations are over the mask patterns.

The alignment increases with the block position b. For the typical block size L'=32, the expected alignment is 0.5 in the first block, 0.75 in the second block, and 0.95 in the tenth block.

4 Parallel Autoregressive Decoding

The above observation and analysis suggest that recent BDLMs are better matched to left-to-right decoding contexts. This motivates a token sampler that preserves the AR prefix structure during generation. While a pure AR sampler obviously is the perfect match, it is inherently sequential and therefore sacrifices the parallel decoding advantage of DLMs.

To alleviate this problem, we introduce Parallel Autoregressive Decoding (PARD). PARD retains the AR bias while still allowing multiple tokens to be unmasked per step, and can be applied to existing arbitrary-order samplers.

Consider generation within block b, conditioned on the clean prefix $x_0^{<b}$. At sampling step s, let x_s^b be the current partially masked block, and denote by $\mathcal{M}_s^b$ the set of masked positions in this block. For each masked position $j \in \mathcal{M}_s^b$, let

$$\pi_j(v) = p_\theta(x_0^{b,j} = v \mid x_0^{<b}, x_s^b)$$

denote the model's predictive distribution over tokens. Let $p_j^{(1)}$ and $p_j^{(2)}$ be the largest and second-largest probabilities under π_j, respectively. Recall that the parallel samplers (Confidence, Margin, and Entropy) select the sets of unmasking positions as:

$$\begin{aligned} \textsf{Confidence:} \quad & U_s^{\mathrm{C}} = \{j \in \mathcal{M}_s^b : c_j > \tau_c\}, \\ \textsf{Margin:} \quad & U_s^{\mathrm{M}} = \{j \in \mathcal{M}_s^b : m_j > \tau_m\}, \\ \textsf{Entropy:} \quad & U_s^{\mathrm{H}} = \{j \in \mathcal{M}_s^b : h_j < \tau_h\}, \end{aligned}$$

where $c_j = p_j^{(1)}$, $m_j = p_j^{(1)} - p_j^{(2)}$, $h_j = -\sum_{v \in V} \pi_j(v) \log \pi_j(v)$, and τ_c, τ_m, τ_h are given thresholds. If the selected set is empty, the sampler falls back to unmasking the top candidate position under the same criterion, e.g., $\arg\max_j c_j$ for Confidence.

PARD applies the same criteria, but restricts unmasking to the leftmost accepted prefix. It scans the masked positions from left to right and unmasks consecutive positions until the first one that fails the criterion. Specifically, it selects the sets of unmasking positions as:

$$\begin{aligned} U_s^{\text{PARD-C}} &= \{j \in \mathcal{M}_s^b : c_i > \tau_c \; \forall i \in \mathcal{M}_s^b,\, i \le j\}, \\ U_s^{\text{PARD-M}} &= \{j \in \mathcal{M}_s^b : m_i > \tau_m \; \forall i \in \mathcal{M}_s^b,\, i \le j\}, \\ U_s^{\text{PARD-H}} &= \{j \in \mathcal{M}_s^b : h_i < \tau_h \; \forall i \in \mathcal{M}_s^b,\, i \le j\}. \end{aligned}$$

If this prefix is empty, it falls back to unmasking the leftmost masked position, i.e., $U_s = \{\min \mathcal{M}_s^b\}$.

Let $\hat{x}_s^{b,j} = \arg\max_{v \in V} p_\theta(x_0^{b,j} = v \mid x_0^{<b}, x_s^b)$. The block is then updated as

$$x_{s+1}^{b,j} = \begin{cases} \hat{x}_s^{b,j} & j \in U_s, \\ x_s^{b,j} & \text{otherwise.} \end{cases}$$

where U_s is chosen from the corresponding arbitrary-order or PARD's unmasking sets defined above. The update is repeated until all positions in the current block are unmasked. Generation then proceeds block by block until an `[EOS]` token is generated or the maximum generation length is reached. This design of PARD preserves the left-to-right context that BDLMs are better aligned with, while retaining parallel token commitment during decoding.

5 Experiments

In Section 5.1 we first evaluate the speed-quality trade-off by varying the parallel decoding thresholds. In Section 5.2, we then focus on the generation qualities and evaluate under fixed threshold settings across various tasks. We evaluate three recent BDLMs: Fast-dLLM v2 7B (Wu et al., 2026b), SDAR 8B (Cheng et al., 2025), and LLaDA2.1-Mini 16B (Bie et al., 2026). All models use a block size of 32, matching their default configurations. More details about the models are in Appendix C.1.

5.1 Speed-Quality Trade-Off

We first evaluate PARD on how it improves the trade-off between generation quality and decoding efficiency. We compare Confidence, Margin, and Entropy parallel sampling with their PARD variants, sweeping the corresponding thresholds τ_c, τ_m, and τ_h. Details on the threshold values are in Appendix C.2. We conduct the threshold sweeps on Fast-dLLM v2 and SDAR. LLaDA2.1-Mini is more computationally demanding and additionally introduces a token-editing threshold, making the speed–quality sweep less straightforward. Evaluation is conducted on the *HumanEval* Base, *MBPP* Base, and *GSM8K*, covering both code generation and mathematical reasoning. Because of the lack of space, results are only presented for *HumanEval*, while those on *MBPP* and *GSM8K* (with similar trends) are reported in Appendix D.1.

Figure 3 shows the pass@1-throughput curves. Results on the pass@1-Number of Function Evaluations (NFE) curves are shown in Figure 5 of Appendix D.1, where similar trends are observed. As can be seen, applying the AR prefix constraint consistently improves the trade-off, shifting the curves toward higher pass@1 at comparable throughput. Margin shows the largest gap between the original sampler and its PARD variant. This suggests that a token-level margin criterion alone is insufficient for parallel sampling, but becomes much more effective when paired with a left-to-right

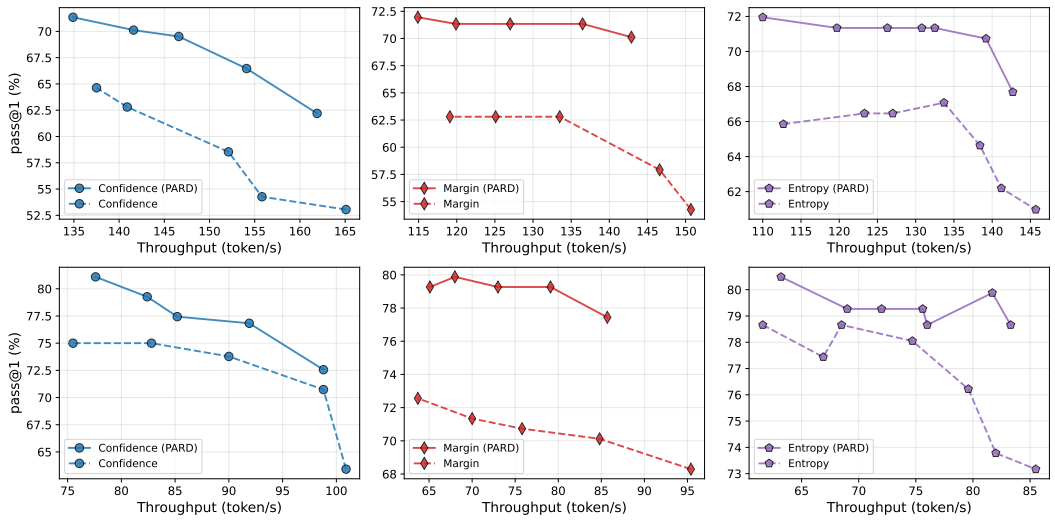

Figure 3: *HumanEval* Base pass@1 vs. decoding throughput (token/sec) for Confidence (left), Margin (middle), and Entropy (right) parallel sampling and their PARD variants on Fast-dLLM v2 7B (top row) and SDAR 8B (bottom row).

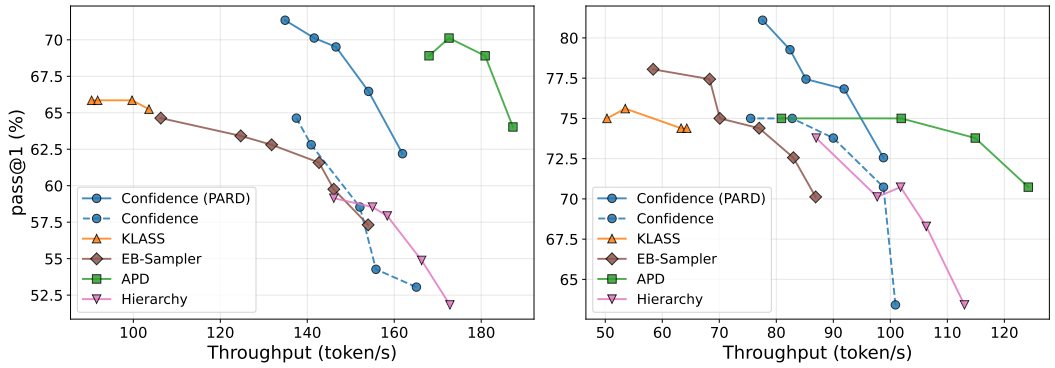

Figure 4: *HumanEval* Base pass@1 vs. decoding throughput (token/sec) for confidence-based PARD, EB-Sampler, KLASS, APD, and Hierarchy on Fast-dLLM v2 7B (left) and SDAR 8B (right).

prefix constraint. The gains are also clear for Confidence and Entropy, indicating that PARD is advantageous for various criteria.

Next, we further compare confidence-based PARD with advanced parallel samplers, KLASS (Kim et al., 2025b), EB-Sampler (Ben-Hamu et al., 2025), APD (Israel et al., 2025), and Hierarchy decoding (Qi et al., 2026). APD is particularly related to PARD because it also unmasks a longest agreed prefix, where the prefix is verified by an external small AR model.

Figure 4 shows the pass@1-throughput curves. The corresponding pass@1-NFE curves (with similar trends) are in Figure 6 of Appendix D.1. Across both models, PARD better preserves pass@1 while improving throughput. KLASS has the lowest throughput, and its throughput changes only slightly across KL thresholds. This is because its KL-stability criterion requires high-precision full-vocabulary softmax operations for numerical stability. APD achieves higher throughput than PARD on Fast-dLLM v2, but its best pass@1 remains lower than that of PARD on both models. Overall, these results suggest that preserving the AR prefix structure is more efficient and effective for BDLM decoding than using more complex arbitrary-order subset-selection rules.

5.2 Generation Quality and Efficiency

Instead of sweeping the thresholds, in this section we focus on the commonly used threshold settings and evaluate benchmark performance across a broader set of tasks and models. We use six benchmarks:

Table 2: Performance across six benchmarks on three recent BDLMs. Methods are grouped into sequential and parallel samplers. For each benchmark, the best result among parallel samplers is **bolded**, and the best result among sequential samplers is underlined.

Type	Method	HumanEval		MBPP		GSM8K	MATH	IFEval	BBH	Avg
		Base	Plus	Base	Plus					
					Fast-dLLM v2 7B					
Sequential	Confidence	65.9	61.0	63.5	53.4	83.6	60.7	60.1	51.9	62.5
	Margin	62.8	57.9	60.3	50.8	82.9	60.0	57.9	50.7	60.4
	Entropy	65.9	61.0	65.3	56.1	82.6	62.4	60.8	53.0	63.4
	Uncode	68.9	64.6	67.2	56.4	84.1	63.4	62.3	54.0	65.1
	AR	72.0	67.7	69.3	58.7	84.3	65.0	62.5	54.6	66.8
Parallel	KLASS	65.9	61.0	63.5	53.4	83.8	61.0	60.3	52.1	62.6
	EB-Sampler	62.8	59.1	63.8	54.5	82.1	58.7	61.0	51.7	61.7
	Confidence Parallel	64.6	59.8	63.5	52.9	82.4	58.0	**62.1**	51.4	61.8
	APD	64.0	59.8	64.0	54.8	81.4	58.8	58.4	49.8	61.4
	Hierarchy	59.1	54.9	61.1	51.3	81.0	58.1	59.7	50.2	59.4
	PARD	**71.3**	**67.1**	**69.6**	**58.7**	**83.9**	**63.5**	**62.1**	**54.3**	**66.3**
					SDAR 8B					
Sequential	Confidence	75.0	70.1	75.1	64.0	89.8	71.4	58.8	62.2	70.8
	Margin	71.3	65.2	73.3	61.4	89.5	70.3	56.9	61.8	68.7
	Entropy	77.4	71.3	77.0	65.6	91.0	72.1	59.7	65.5	72.5
	Uncode	78.0	72.6	76.2	65.1	91.2	73.3	60.1	66.2	72.8
	AR	79.3	73.2	78.0	66.4	91.1	73.3	61.0	66.5	73.6
Parallel	KLASS	75.0	70.1	75.1	64.3	90.5	70.2	57.9	61.5	70.6
	EB-Sampler	75.0	69.5	**78.0**	**66.9**	90.6	69.6	59.5	64.7	71.7
	Confidence Parallel	75.0	70.1	74.9	63.8	89.8	67.7	58.8	61.9	70.2
	APD	70.7	66.5	72.5	61.4	89.2	67.8	59.3	61.1	68.6
	Hierarchy	73.8	68.3	73.5	62.7	87.7	65.8	58.2	61.6	69.0
	PARD	**81.1**	**76.2**	77.0	65.3	**91.4**	**71.6**	**60.4**	**65.8**	**73.6**
					LLaDA2.1-Mini 16B					
Sequential	Confidence	75.0	72.6	82.3	69.8	91.7	81.8	83.9	86.3	80.4
	Margin	78.1	74.4	81.5	68.8	91.4	82.1	83.7	86.8	80.8
	Entropy	79.3	75.6	82.8	69.3	91.4	81.9	84.3	87.1	81.5
	Uncode	76.2	72.0	83.6	71.2	92.0	82.3	84.7	86.7	81.1
	AR	87.8	84.8	85.7	72.2	92.3	82.2	84.3	86.9	84.5
Parallel	KLASS	78.7	76.2	80.7	68.2	91.3	**82.3**	**84.1**	85.8	80.9
	EB-Sampler	80.5	77.4	81.5	69.6	91.4	**82.3**	83.0	86.0	81.5
	Confidence Parallel	84.8	82.3	78.8	66.7	90.5	81.6	81.9	85.2	81.5
	Hierarchy	81.1	76.8	79.4	67.2	91.6	81.8	80.4	85.6	80.5
	PARD	**86.6**	**82.9**	**84.9**	**71.4**	**92.0**	**82.3**	**84.1**	**86.4**	**83.9**

HumanEval (Chen et al., 2021) and *MBPP* (Austin et al., 2021b) for code generation, *GSM8K* (Cobbe et al., 2021) and *MATH* (Hendrycks et al., 2021) for mathematical reasoning, *IFEval* (Zhou et al., 2023) for instruction following, and *BigBenchHard (BBH)* (Suzgun et al., 2022) for general reasoning. We report pass@1 on both Base and Plus variants of HumanEval and MBPP. For LLaDA2.1-Mini, we evaluate *BBH* on an 8-task subset due to computational constraints. Details are in Appendix C.4.

For models, in addition to Fast-dLLM v2 7B and SDAR 8B, we include LLaDA2.1-Mini 16B (Bie et al., 2026). Since these BDLMs use Confidence Parallel (Wu et al., 2025) as their default decoding strategy, we compare against it and use confidence-based PARD in this benchmark evaluation. Following their default configurations, we use τ_c=0.9 for Fast-dLLM v2 and SDAR, and τ_c=0.7 for LLaDA2.1-Mini. For LLaDA2.1-Mini, we use the default editing threshold τ_{edit}=0.5.

As additional baselines, we include sequential strategies (which unmasks one token per step) based on Confidence, Margin and Entropy; and the recent strong sequential sampler Uncode (Huang et al., 2026). We also compare with pure AR sampling, which always unmasks the leftmost masked position. For the parallel baselines KLASS, EB-Sampler, APD, and Hierarchy decoding, we follow their default settings. Details are given in Appendix C.2. We report APD only for Fast-dLLM v2 and SDAR, where a small AR verifier sharing the same tokenizer as the target DLM is available.

Table 2 shows the results of various sampling methods. Among the parallel samplers, PARD achieves the best average accuracy on every model, outperforming Confidence Parallel by 4.5, 3.4, and 2.4 points on Fast-dLLM v2, SDAR, and LLaDA2.1-Mini, respectively. The improvements are more

Table 3: Decoding throughput (tokens/sec) of AR and confidence-based PARD on six benchmarks. Speedup over AR is shown in parentheses.

Model	Method	HumanEval	MBPP	GSM8K	MATH	IFEval	BBH
Fast-dLLM v2 7B	AR	85.8	82.7	96.1	69.6	64.8	77.6
	PARD	134.9 (1.57×)	112.5 (1.36×)	179.9 (1.87×)	152.0 (2.18×)	111.2 (1.72×)	113.5 (1.46×)
SDAR 8B	AR	46.5	42.4	46.2	40.7	32.0	40.1
	PARD	77.6 (1.67×)	72.6 (1.71×)	108.2 (2.34×)	124.9 (3.07×)	59.4 (1.86×)	60.5 (1.51×)
LLaDA2.1-Mini	AR	12.1	10.6	10.0	10.9	9.1	11.6
	PARD	44.1 (3.64×)	36.8 (3.47×)	29.4 (2.94×)	28.9 (2.65×)	11.7 (1.29×)	25.2 (2.17×)

pronounced on code generation tasks. This may be because arbitrary-order samplers tend to bypass uncertain earlier positions and commit later tokens first, even when their confidence is only slightly higher. This can prematurely constrain the program structure. We illustrate this failure mode with a qualitative example in Appendix D.3. Taking the sequential samplers also into account, pure AR achieves the best accuracy on all three models. PARD performs strongly, matching AR on SDAR and trailing AR by only 0.5 and 0.6 points on Fast-dLLM v2 and LLaDA2.1-Mini, respectively. We additionally evaluate with stochastic top-p sampling in Appendix D.4, where PARD retains the highest mean performance among the competitive parallel samplers.

Although AR achieves the strongest generation quality, it is sequential. We measure the decoding throughput of AR and confidence-based PARD across benchmarks. Table 3 reports throughput in generated tokens per second, with speedup over AR shown in parentheses. PARD is consistently faster than AR, with speedup varying across models and tasks. This variation closely follows the number of tokens committed per step. For example, on SDAR, PARD commits an average of 2.02 tokens per step on MBPP and 3.14 tokens per step on GSM8K, while the corresponding speedup increases from $1.71\times$ to $2.34\times$. This suggests that PARD is particularly effective when it can confidently commit longer prefixes. We further compare the number of tokens unmasked per step with Confidence Parallel in Appendix D.2. PARD unmasks a similar number of tokens per step, indicating that the AR prefix constraint does not hurt parallelism and largely preserves efficiency.

5.3 Ablation

In this experiment, we ablate the effect of adaptive prefix selection by comparing confidence-based PARD with a static leftmost variant, which always unmasks a fixed number (k) of leftmost masked tokens per step. Evaluation is performed on *MBPP*. PARD uses the same thresholds as in Section 5.2.

As shown in Table 4, the static variants suffer large performance drops across all three models. Even with only $k = 2$, pass@1 drops by 20.1 points on Fast-dLLM v2 7B, 19.3 points on SDAR 8B, and 8.4 points on LLaDA2.1-Mini. The degradation becomes even more severe as k increases, with $k = 4$ reducing pass@1 by 46.8, 53.5, and 23.3 points, respectively. These results show that simply committing multiple leftmost tokens is too aggressive. Effective parallel left-to-right decoding must also decide when the leftmost prefix is sufficiently reliable.

Table 4: *MBPP* Base pass@1 for PARD and static leftmost variants on Fast-dLLM v2 7B, SDAR 8B, and LLaDA2.1-Mini.

Method	Fast-dLLM v2 7B	SDAR 8B	LLaDA2.1-Mini
PARD	69.6	77.0	84.9
Leftmost (k=2)	49.5	57.7	76.5
Leftmost (k=3)	34.4	34.7	69.0
Leftmost (k=4)	22.8	23.5	61.6

6 Conclusion

We revisited sampling for recent BDLMs in this work. Our empirical and training-context analyses show that BDLMs are more naturally aligned with left-to-right decoding than with fully arbitrary-order sampling. Motivated by this observation, we introduced PARD, a training-free sampler that preserves the AR prefix structure while retaining parallel token commitment. Across three recent BDLMs and six benchmarks, PARD substantially improves throughput over pure AR while incurring only a small quality drop, and consistently outperforms existing parallel samplers in generation quality.

AI USE STATEMENT

Generative AI tools were used to assist with the code edits and experiments in this work. In particular, Claude Code was used for code implementation, and ChatGPT was used for language polishing and improving the readability of the manuscript. The authors reviewed all AI-assisted work. The central idea of AR bias analysis of DLMs and the PARD framework were developed by the authors.

REPRODUCIBILITY STATEMENT

We provide detailed experimental settings in Appendix C, including model checkpoints, hyperparameter values, implementation details, and benchmark configurations. All model checkpoints and datasets used in our experiments are publicly available. Code will be released upon publication.

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.

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

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

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas Nolte, and Brian Karrer. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *arXiv preprint arXiv:2505.24857*, 2025.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, et al. LLaDA 2.0: Scaling up diffusion language models to 100B. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, et al. LLaDA 2.1: Speeding up text diffusion via token editing. *arXiv preprint arXiv:2602.08676*, 2026.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35:28266–28279, 2022.

Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11315–11325, 2022.

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.

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

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

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, et al. Nemotron-labs-diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding. *arXiv preprint arXiv:2607.05722*, 2026.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation. https://github.com/EleutherAI/lm-evaluation-harness, 2023.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=58NA3unZj5.

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. *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

Pengcheng Huang, Tianming Liu, Zhenghao Liu, Yukun Yan, Shuo Wang, Tong Xiao, Zulong Chen, and Maosong Sun. Empirical analysis of decoding biases in masked diffusion models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 52870–52888. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4c7d31b2da05a17f15101db5b37a1e24-Paper-Conference.pdf.

Woogyeol Jin, Taywon Min, Yongjin Yang, Dennis Wei, Yi Zhou, Swanand Ravindra Kadhe, Nathalie Baracaldo, and Kimin Lee. Entropy-aware on-policy distillation of language models. In *Forty-third International Conference on Machine Learning*, 2026.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv e-prints*, pp. arXiv–2506, 2025.

Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham M Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In *International Conference on Machine Learning*, pp. 30749–30768. PMLR, 2025a.

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. KLASS: KL-guided fast inference in masked diffusion models. *Advances in Neural Information Processing Systems*, 2025b.

Pengxiang Li, Dilxat Muhtar, Tianlong Chen, Lu Yin, and Shiwei Liu. Why diffusion language models struggle with truly parallel (non-autoregressive) decoding? *arXiv preprint arXiv:2602.23225*, 2026.

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

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023.

Junnan Liu, Hongwei Liu, Linchen Xiao, Ziyi Wang, Kuikun Liu, Songyang Gao, Wenwei Zhang, Songyang Zhang, and Kai Chen. Are your llms capable of stable reasoning? In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 17594–17632, 2025.

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

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, et al. The flexibility trap: Why arbitrary order limits reasoning potential in diffusion language models. *arXiv preprint arXiv:2601.15165*, 2026.

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.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *International Conference on Learning Representations*, volume 2025, pp. 64972–65009, 2025.

Xiaojing Qi, Lun Du, Xinyuan Zhang, Lanning Wei, Tao Jin, and Da Zheng. Hierarchy decoding: A training-free parallel decoding strategy for diffusion large language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

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

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37: 103131–103167, 2024.

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

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Yuchuan Tian, Yuchen Liang, Shuo Zhang, Yingte Shu, Guangwen Yang, et al. From next-token to next-block: A principled adaptation path for diffusion LLMs. *arXiv preprint arXiv:2512.06776*, 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.

Chengyue Wu, Shiyi Lan, Yonggan Fu, Sensen Gao, Jin Wang, Jincheng Yu, Jose M Alvarez, Pavlo Molchanov, Ping Luo, Song Han, et al. Fast-dvlm: Efficient block-diffusion vlm via direct conversion from autoregressive vlm. *arXiv preprint arXiv:2604.06832*, 2026a.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dLLM v2: Efficient block-diffusion LLM. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=1NZ3DHF9nT.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

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

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding. *arXiv preprint arXiv:2505.16990*, 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, et al. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

A Definitions of Local AR-ness@k and Global AR-ness@k

Let L' be the block size. For a generated block, let $M_{s-1} \subseteq \{1, \dots, L'\}$ denote the set of masked positions before step s, and let $p_s \in M_{s-1}$ be the position unmasked at step s. Following Gong et al. (2026), we define local AR-ness by the step-level indicator

$$I_{\mathrm{loc}}(s,k) = \begin{cases} 1, & \{p_{s-i}\}_{i=1}^{k} = \{p_s - i\}_{i=1}^{k}, \\ 0, & \text{otherwise.} \end{cases}$$

The block-level local AR-ness@k is computed as

$$\frac{1}{L'} \sum_{s=1}^{L'} I_{\mathrm{loc}}(s,k).$$

For global AR-ness, let $\mathrm{Left}_k(M_{s-1})$ denote the k leftmost positions in M_{s-1}. We define

$$I_{\mathrm{glob}}(s,k) = \begin{cases} 1, & p_s \in \mathrm{Left}_k(M_{s-1}), \\ 0, & \text{otherwise.} \end{cases}$$

The block-level global AR-ness@k is computed as

$$\frac{1}{L'} \sum_{s=1}^{L'} I_{\mathrm{glob}}(s,k).$$

B Proofs

B.1 Proof of Proposition 1

Proof. Given a clean training sequence x_0. Assume $\alpha_t = 1-t$. During training, we draw $t \sim U[0,1]$, then draw the noised sequence $x_t \sim q(\cdot \mid x_0)$. Each token position is then masked independently by probability $1 - \alpha_t = t$. Let $E = \{\mathbf{m}_\ell^{\mathrm{AR}}\ observed\}$ denote the event of interest.

For the MDM case, we have

$$\Pr(E \mid t) = \alpha_t^{\ell-1} \cdot (1-\alpha_t)^{L-\ell+1} \tag{2}$$

$$= (1-t)^{\ell-1} \cdot t^{L-\ell+1} \tag{3}$$

Then marginalize over $t \sim U[0,1]$,

$$\Pr(E) = \mathbb{E}_{t\sim U[0,1]} \Pr(E \mid t) \tag{4}$$

$$= \int_0^1 (1-t)^{\ell-1} \cdot t^{L-\ell+1} dt \tag{5}$$

$$= \frac{(\ell-1)!\,(L-\ell+1)!}{(L+1)!} \tag{6}$$

$$= \frac{1}{(L+1)\binom{L}{\ell-1}} \tag{7}$$

where (7) uses the definition of the Beta function.

For the block diffusion case, let $k = \ell - 1 - (b-1)L'$ be the number of positions before ℓ in this block b. Note that the positions in the conditioning prefix ($x_0^{<b}$) are deterministically unmasked. We have

$$\Pr(E \mid t) = \alpha_t^{k} \cdot (1-\alpha_t)^{L'-k} \tag{8}$$

$$= (1-t)^{k} \cdot t^{L'-k} \tag{9}$$

Then marginalize over $t \sim U[0,1]$,

$$\Pr(E) = \mathbb{E}_{t\sim U[0,1]} \Pr(E \mid t) \tag{10}$$

$$= \int_0^1 (1-t)^k \cdot t^{L'-k} dt \tag{11}$$

$$= \frac{k!(L'-k)!}{(L'+1)!} \tag{12}$$

$$= \frac{1}{(L'+1)\binom{L'}{k}} \tag{13}$$

□

B.2 Proof of Proposition 2

Proof. Recall the definition of *AR pattern alignment*: Under MDM, it is

$$\rho_\ell(\mathbf{m}^{\mathrm{MDM}}) = \frac{1}{L-1}\left[\sum_{j=1}^{\ell-1}(1-\mathbf{m}_j^{\mathrm{MDM}}) + \sum_{j=\ell+1}^{L}\mathbf{m}_j^{\mathrm{MDM}}\right].$$

Similarly, under block diffusion, it is

$$\rho_\ell(\mathbf{m}^{\mathrm{BDLM}}) = \frac{1}{bL'-1}\left[\sum_{j=1}^{\ell-1}(1-\mathbf{m}_j^{\mathrm{BDLM}}) + \sum_{j=\ell+1}^{bL'}\mathbf{m}_j^{\mathrm{BDLM}}\right].$$

For the MDM case, for all positions $j \in \{1,...,L\}$, $\mathbf{m}_j^{\mathrm{MDM}} \mid t \sim$ Bernoulli$(1-\alpha_t)$ iid, with $t \sim U[0,1]$. Assume the linear schedule $\alpha_t = 1-t$.

$$\begin{aligned}\mathbb{E}[\mathbf{m}_j^{\mathrm{MDM}}] &= \mathbb{E}_{t\sim U[0,1]}[\mathbb{E}[\mathbf{m}_j^{\mathrm{MDM}} \mid t]] \\ &= \mathbb{E}_{t\sim U[0,1]}[1-\alpha_t] \\ &= \int_0^1 t\,dt = \frac{1}{2}.\end{aligned}$$

Similarly,

$$\begin{aligned}\mathbb{E}[1-\mathbf{m}_j^{\mathrm{MDM}}] &= \mathbb{E}_{t\sim U[0,1]}[\mathbb{E}[1-\mathbf{m}_j^{\mathrm{MDM}} \mid t]] \\ &= \int_0^1 (1-t)\,dt = \frac{1}{2}.\end{aligned}$$

Therefore,

$$\mathbb{E}[\rho_\ell(\mathbf{m}^{\mathrm{MDM}})] = \frac{1}{L-1}\left[(\ell-1)\frac{1}{2} + (L-\ell)\frac{1}{2}\right] = \frac{1}{2}$$

For the block diffusion case, $\mathbf{m}_j^{\mathrm{BDLM}} = 0$ for $j \in \{1,...,(b-1)L'\}$ deterministically. For positions $j \in \{(b-1)L'+1,...,bL'\}$, $\mathbf{m}_j^{\mathrm{BDLM}} \mid t \sim$ Bernoulli$(1-\alpha_t)$ iid, with $t \sim U[0,1]$. Let $k = \ell - 1 - (b-1)L'$ be the number of positions before ℓ in this block b. Let $r = (b-1)L'$. We split the first sum at the block-b boundary:

$$\begin{aligned}\mathbb{E}\left[\sum_{j=1}^{\ell-1}(1-\mathbf{m}_j^{\mathrm{BDLM}})\right] &= \sum_{j=1}^{r}\mathbb{E}[1-\mathbf{m}_j^{\mathrm{BDLM}}] + \sum_{j=r+1}^{\ell-1}\mathbb{E}[1-\mathbf{m}_j^{\mathrm{BDLM}}] \\ &= (b-1)L' + \frac{k}{2}.\end{aligned}$$

For the second sum,

$$\mathbb{E}\left[\sum_{j=\ell+1}^{bL'}\mathbf{m}_j^{\mathrm{BDLM}}\right] = \frac{L'-k-1}{2}$$

Therefore, combining both expected sums,

$$\begin{aligned}
&\mathbb{E}[\rho_\ell(\mathbf{m}^{\mathrm{BDLM}})] \\
&= \frac{1}{bL'-1}\left[(b-1)L' + \frac{k}{2} + \frac{L'-k-1}{2}\right] \\
&= \frac{1}{2} + \frac{(b-1)L'}{2(bL'-1)}
\end{aligned}$$

□

C Experimental Details

C.1 Model Checkpoints

We use the following publicly available HuggingFace checkpoints in our experiments.

- **Fast-dLLM v2 7B.** The model is initialized from Qwen2.5-7B-Instruct (Yang et al., 2024) and adapted to block diffusion paradigm with $\sim$ 1B tokens of fine-tuning. We use `Efficient-Large-Model/Fast_dLLM_v2_7B`[1], the 7B checkpoint released by Wu et al. (2026b). We follow its default sub-block inference strategy with a sub-block size of 8, except in the experiment in Section 3.1, where we disable sub-block inference to avoid imposing additional left-to-right structure.
- **SDAR 8B.** The model is initialized from Qwen3-8B (Yang et al., 2025) and adapted to the block diffusion paradigm via continued pretraining on 50B tokens, followed by supervised fine-tuning on 4B instruction tokens. We use `JetLM/SDAR 8B-Chat-b32`[2], the block-size-32 version 8B checkpoint released by Cheng et al. (2025).
- **LLaDA 8B.** This model is trained from scratch with MDM objective. We use `GSAI-ML/LLaDA 8B-Instruct`[3], released by Nie et al. (2025). This model is used in the comparison experiment in Section 3.1 to contrast the decoding behavior of an MDM with that of a BDLM. We use it with `DualCache` from Wu et al. (2025) for faster inference.
- **Dream 7B.** The model is initialized from Qwen2.5-7B (Yang et al., 2024) and adapted to the masked diffusion paradigm with 580B tokens of continual pretraining. We use `Dream-org/Dream-v0-Instruct-7B`[4], released by Ye et al. (2025). We use it with `DualCache` from Wu et al. (2025) for faster inference.
- **LLaDA2.1-Mini 16B.** A Mixture-of-Experts (MoE) block diffusion language model. It supports token editing, where previously unmasked tokens in the current block can be revised when a new prediction exceeds an editing threshold τ_{edit}. We use `inclusionAI/LLaDA2.1-mini`[5], released by Bie et al. (2026).

C.2 Hyperparameter Values and APD's Verifier Models

In the speed–quality trade-off experiments (Section 5.1), we use the same set of threshold values for both Fast-dLLM v2 and SDAR. Specifically, we use $\tau_c \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$ for Confidence, $\tau_m \in \{0.7, 0.8, 0.9, 0.95, 0.97\}$ for Margin, and $\tau_h \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$ for Entropy.

For the comparison with advanced parallel samplers, we use $\tau_c \in \{0.7, 0.75, 0.8, 0.85, 0.9\}$ for confidence-based PARD, entropy-bound thresholds in $\{0.01, 0.05, 0.1, 0.2, 0.3, 0.4\}$ for EB-Sampler, KL-divergence thresholds in $\{0.01, 0.1, 1.0, 10.0\}$ for KLASS, and mixture weights in $\{0.01, 0.1, 0.3, 0.5\}$ for APD. For Hierarchy, which uses high- and low-confidence thresholds as well as a remasking step, we sweep the high threshold τ_{high} over the same grid as τ_c, since both thresholds play a similar role in controlling confidence-based token acceptance. We set the low threshold to

[1] `https://huggingface.co/Efficient-Large-Model/Fast_dLLM_v2_7B`
[2] `https://huggingface.co/JetLM/SDAR-8B-Chat-b32`
[3] `https://huggingface.co/GSAI-ML/LLaDA-8B-Instruct`
[4] `https://huggingface.co/Dream-org/Dream-v0-Instruct-7B`
[5] `https://huggingface.co/inclusionAI/LLaDA2.1-mini`

$\tau_{\text{low}} = 0.5$, as suggested by the authors (Qi et al., 2026), and disable remasking, as we empirically found it to degrade performance.

In the fixed threshold experiments (Section 5.2), following their default settings, we use a KL-divergence threshold of 0.01 for KLASS, an entropy-bound threshold of 0.1 for EB-Sampler, a mixture weight of 0.5 for APD. For Hierarchy decoding, we use the default low confidence threshold of 0.5, while setting the high confidence threshold to the same value as τ_c, as they serve the same role.

APD requires an external small AR verifier that shares the same tokenizer as the target DLM. Following the original APD setup (Israel et al., 2025), we use verifiers at the 0.5–0.6B scale: Qwen2.5-0.5B for Fast-dLLM v2 and Qwen3-0.6B for SDAR, matching the model family of each BDLM. This is important as they share the same tokenizer. We omit APD for LLaDA2.1-Mini due to the lack of a suitable small AR verifier with a compatible tokenizer.

C.3 Implementation Details

For all benchmarks and all samplers, we use greedy decoding (temperature $= 0$, top-p disabled) in the main experiments. We apply each model's default chat template to all question prompts. Generation stops at the end-of-sequence token or when the per-task `max_new_tokens` budget is reached. All experiments run on a single NVIDIA A6000 GPU.

Table 5: Per-benchmark evaluation setup. "#-shot" counts in-context demonstrations, we use zero-shot for all benchmarks in our evaluation.

Benchmark	#-shot	Max new tokens
HumanEval	0	768
MBPP	0	768
GSM8K	0	2048
MATH	0	2048
IFEval	0	2048
BBH	0	1024

C.4 Benchmarks

We use `evalplus` (Liu et al., 2023) for *HumanEval* and *MBPP*, and the `lm-evaluation-harness` (Gao et al., 2023) for the remaining tasks. Table 5 summarizes the per-benchmark configuration, including the number of in-context demonstrations and the maximum new tokens allowed for generation.

For *HumanEval* and *MBPP*, we report both the Base and extended Plus pass@1, where a problem is counted as solved under Plus only if it passes both the base and extended test suites. Base and Plus are evaluated on the same model generations and therefore share the same generation-speed measurements (e.g., wall-clock throughput). For *IFEval* we report the prompt-level strict accuracy.

For LLaDA2.1 on *BBH*, due to computational constraints, we evaluate on a balanced 8-task subset, with 4 tasks drawn from each of the two task groupings introduced by Suzgun et al. (2022):

- **Algorithmic and multi-step arithmetic reasoning**: `boolean_expressions`, `navigate`, `object_counting`, `web_of_lies`.
- **Natural language understanding**: `causal_judgement`, `date_understanding`, `logical_deduction_seven_objects`, `temporal_sequences`.

We report the macro-mean over the eight tasks, following the convention of Suzgun et al. (2022).

D Additional Results

D.1 Speed-Quality Trade-off

Figure 5 compares Confidence, Margin, and Entropy with their PARD variants, and Figure 6 compares confidence-based PARD with EB-Sampler, KLASS, APD, and Hierarchy. Both of the figures show the pass@1-NFE trade-offs on *HumanEval*. PARD consistently achieves strong trade-offs in terms of NFE on *HumanEval*.

Figures 7 and 8 show the corresponding pass@1–throughput and pass@1–NFE trade-offs on *MBPP*. PARD generally improves upon the underlying sampling criteria, with particularly clear gains for Confidence and Margin. Figures 9 and 10 compare confidence-based PARD with the other parallel samplers. On Fast-dLLM v2 7B, PARD achieves the highest pass@1 and favorable trade-offs. On SDAR 8B, EB-Sampler achieves higher peak pass@1, while PARD remains competitive over their overlapping throughput range.

Figures 11-14 extend these comparisons to *GSM8K*. Across both Fast-dLLM v2 and SDAR, PARD consistently improve the accuracy–throughput and accuracy–NFE trade-offs over their corresponding parallel samplers. Against the advanced parallel samplers, confidence-based PARD achieves strong throughput trade-offs in the higher-accuracy region and strong NFE trade-offs on both models. These results further demonstrate PARD's ability to preserve generation quality while enabling efficient parallel decoding.

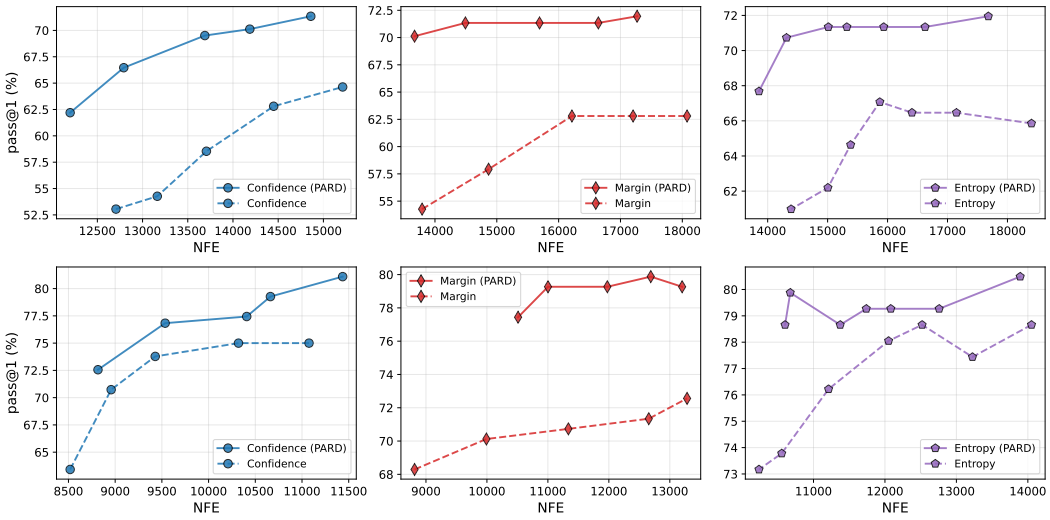

Figure 5: *HumanEval* pass@1 vs. Number of Function Evaluations (NFE) for Confidence (left), Margin (middle), and Entropy (right) parallel sampling and their PARD variants on Fast-dLLM v2 7B (top row) and SDAR 8B (bottom row).

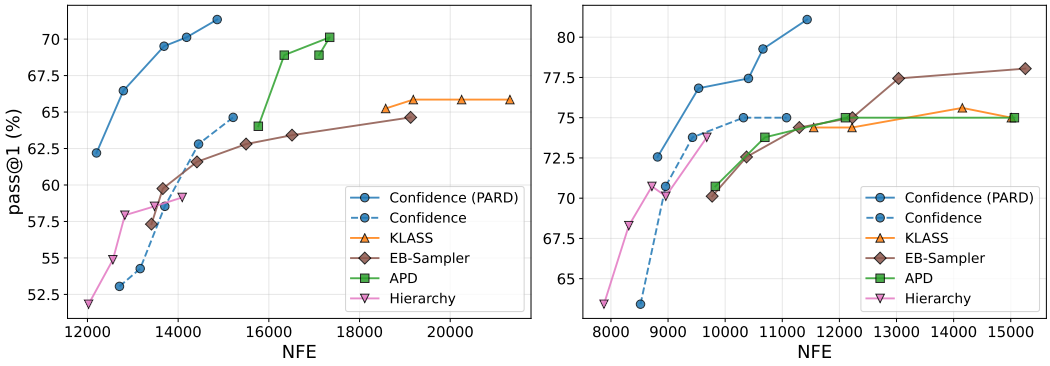

Figure 6: *HumanEval* Base pass@1 vs. Number of Function Evaluations (NFE) for confidence-based PARD, EB-Sampler, KLASS, APD, and Hierarchy on Fast-dLLM v2 7B (left) and SDAR 8B (right).

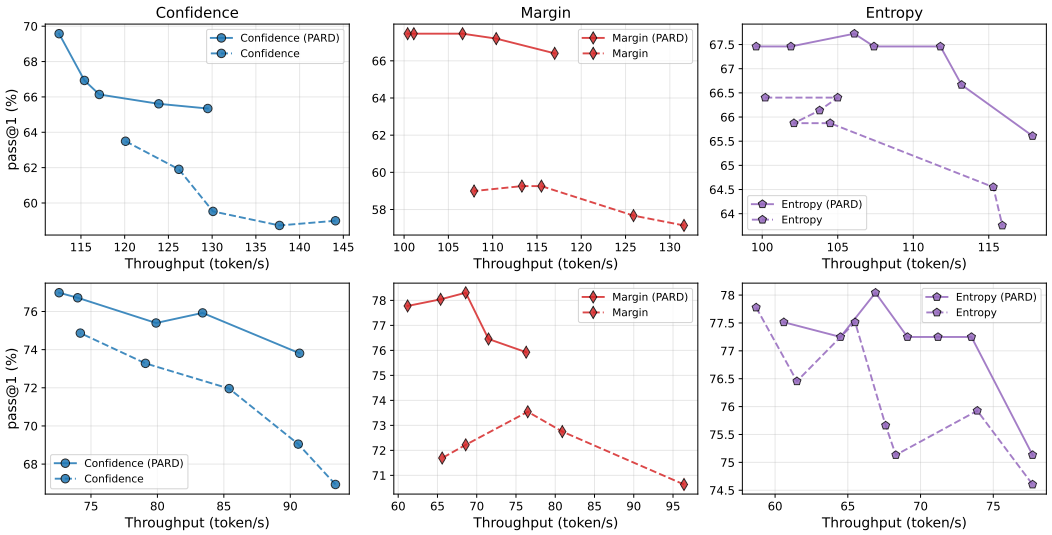

Figure 7: *MBPP* pass@1 vs. decoding throughput for Confidence (left), Margin (middle), and Entropy (right) parallel sampling and their PARD variants on Fast-dLLM v2 7B (top row) and SDAR 8B (bottom row).

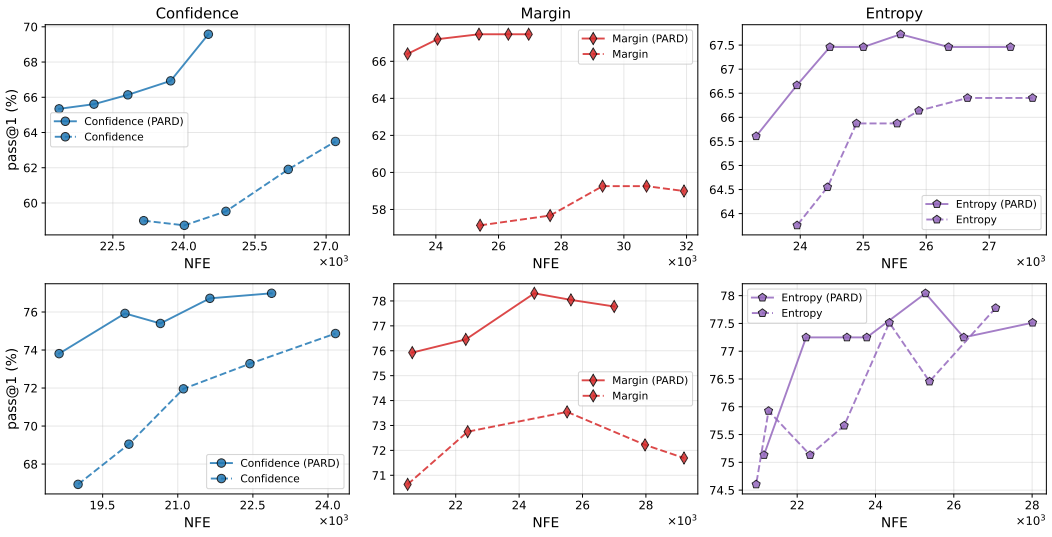

Figure 8: *MBPP* pass@1 vs. NFE for Confidence (left), Margin (middle), and Entropy (right) parallel sampling and their PARD variants on Fast-dLLM v2 7B (top row) and SDAR 8B (bottom row).

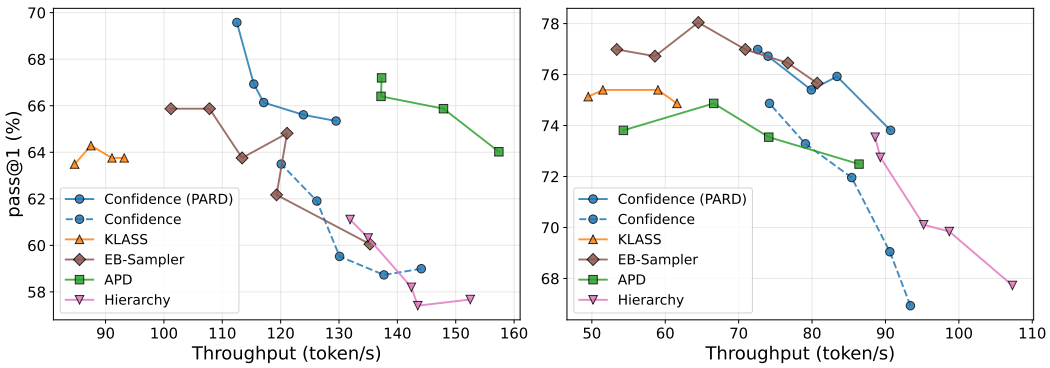

Figure 9: *MBPP* Base pass@1 vs. decoding throughput for confidence-based PARD, EB-Sampler, KLASS, APD, and Hierarchy on Fast-dLLM v2 7B (left) and SDAR 8B (right).

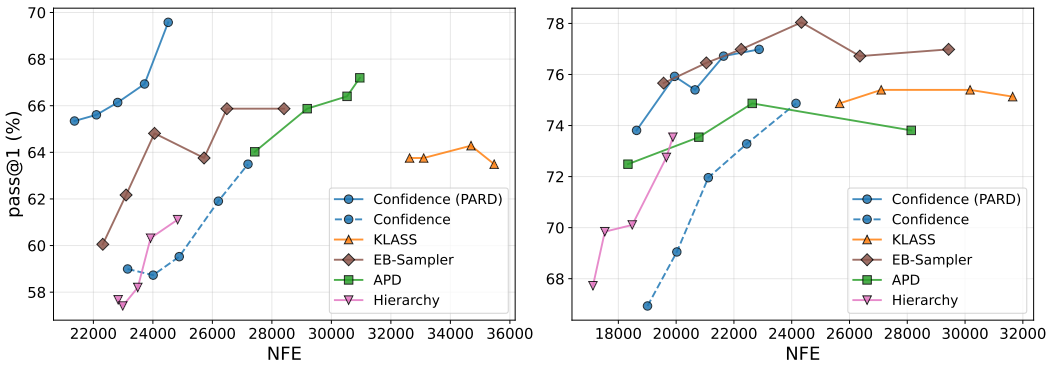

Figure 10: *MBPP* Base pass@1 vs. NFE for confidence-based PARD, EB-Sampler, KLASS, APD, and Hierarchy on Fast-dLLM v2 7B (left) and SDAR 8B (right).

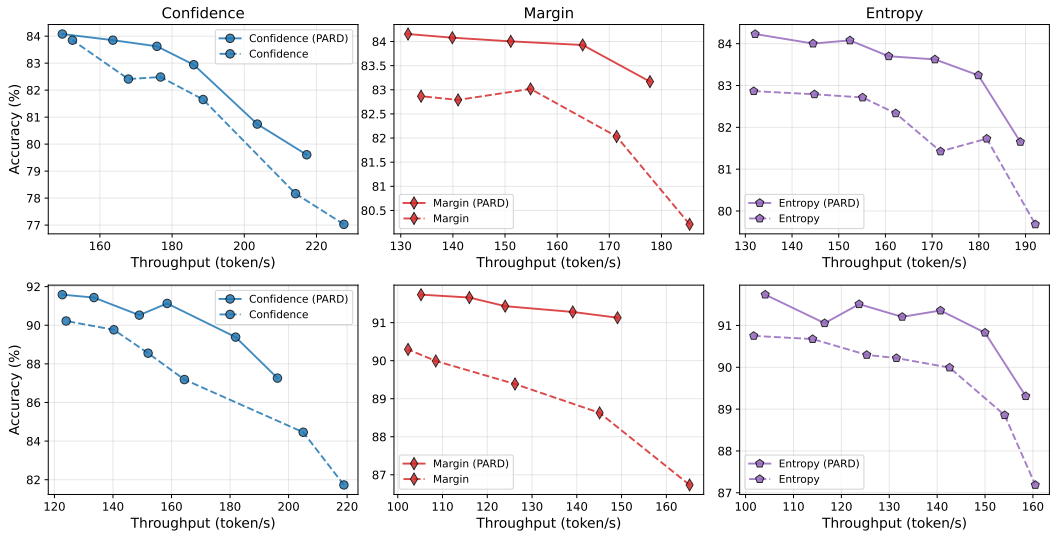

Figure 11: *GSM8K* accuracy vs. decoding throughput for Confidence (left), Margin (middle), and Entropy (right) parallel sampling and their PARD variants on Fast-dLLM v2 7B (top row) and SDAR 8B (bottom row).

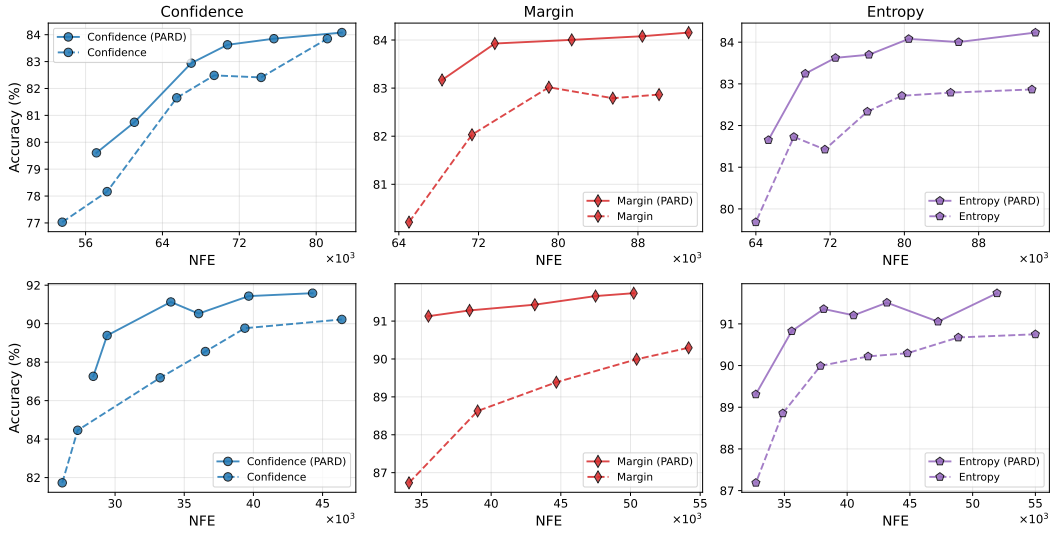

Figure 12: *GSM8K* accuracy vs. NFE for Confidence (left), Margin (middle), and Entropy (right) parallel sampling and their PARD variants on Fast-dLLM v2 7B (top row) and SDAR 8B (bottom row).

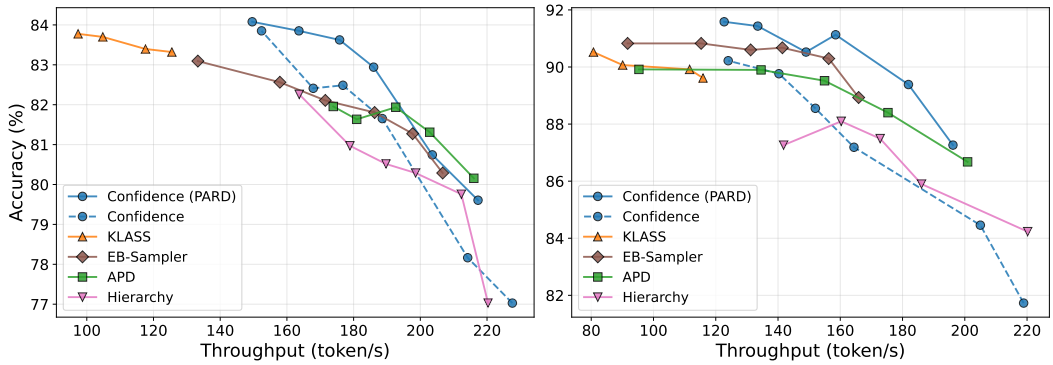

Figure 13: *GSM8K* accuracy vs. decoding throughput for confidence-based PARD, EB-Sampler, KLASS, APD, and Hierarchy on Fast-dLLM v2 7B (left) and SDAR 8B (right).

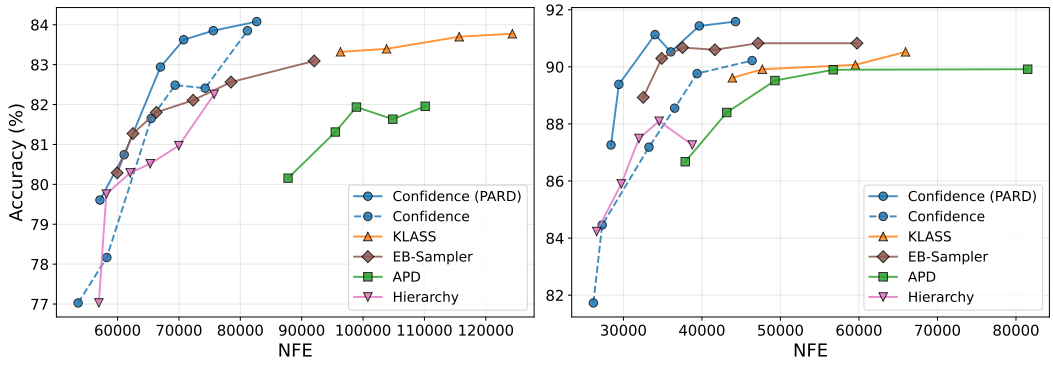

Figure 14: *GSM8K* accuracy vs. NFE for confidence-based PARD, EB-Sampler, KLASS, APD, and Hierarchy on Fast-dLLM v2 7B (left) and SDAR 8B (right).

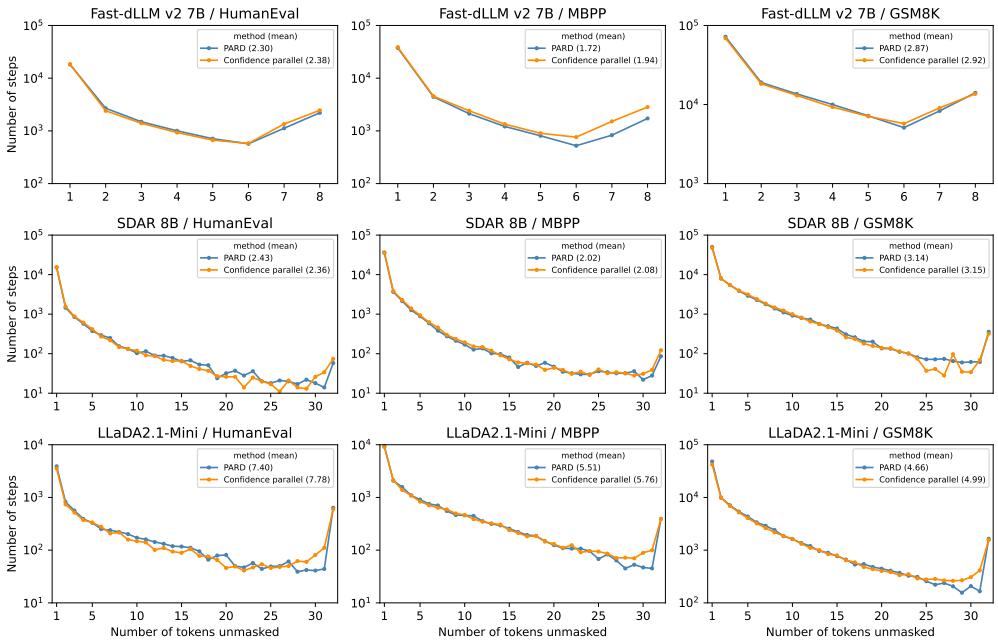

Figure 15: Number of tokens unmasked per step by confidence-based PARD and Confidence Parallel on *HumanEval*, *MBPP*, and *GSM8K*.

D.2 Number of Tokens Unmasked per Step

Figure 15 shows the distributions of the number of tokens unmasked per step by confidence-based PARD and Confidence Parallel on *HumanEval*, *MBPP*, and *GSM8K*. Under the default threshold configurations as in Section 5.2, Fast-dLLM v2 and SDAR use $\tau_c = 0.9$, while LLaDA2.1 uses a more permissive threshold of $\tau_c = 0.7$. We observe similar distributions and mean number of token unmasked for PARD and Confidence Parallel, suggesting that PARD preserves most of the parallelism while enforcing a left-to-right prefix constraint. For Fast-dLLM v2, the number of tokens unmasked in a single step is upper-bounded by 8 because we follow its default sub-block inference strategy with sub-block size 8. Accordingly, Fast-dLLM v2 and SDAR generally unmask about 2-3 tokens per step on average, whereas LLaDA2.1 shows substantially higher parallelism, with average unmasking counts around 5-8 tokens per step.

We also observe a mild upward tail when the number of unmasked tokens approaches the whole-block size, i.e., 8 for Fast-dLLM v2 (sub-block) and 32 for SDAR and LLaDA2.1. We did not identify a clear structural pattern in the decoded contents that fully explains this effect. One explanation is that some blocks occasionally become "easy", so that all masked tokens in the block exceed the confidence threshold and are unmasked in a single step.

D.3 Qualitative Case Study

We study how arbitrary-order decoding can fail with a concrete example from *HumanEval*. Figure 16 shows the generations produced by Confidence Parallel and confidence-based PARD on `HumanEval/111`, using SDAR 8B and the same default confidence threshold τ_c. The two outputs are nearly identical and differ only in the empty-input guard `if not test: return {}`. PARD generates this guard, whereas Confidence Parallel omits this part. Without the guard, `max(counts.values())` is evaluated on an empty dictionary when the input is the empty string and raises a `ValueError`, so Confidence Parallel fails the `histogram("")` test while PARD passes.

The omission stems from the decoding order, shown in Figure 17. Both samplers produce the function signature identically through step 0 and step 1. At step 2, the guard token `"if"`, at the slot immediately after the signature, has confidence 0.20, while `"="`, one slot later (starting `letters = test.split()`), has the highest confidence 0.28 (only 0.08 larger). Confidence Parallel selects the

Sampling Example — HumanEval/111 (`histogram`)

Prompt:

```
def histogram(test):
    """Given a string representing a space separated lowercase letters, return a
    dictionary of the letter with the most repetition and containing the
    ↪  corresponding
    count. If several letters have the same occurrence, return all of them.

    Example:
    histogram('a b c') == {'a': 1, 'b': 1, 'c': 1}
    histogram('a b b a') == {'a': 2, 'b': 2}
    histogram('b b b b a') == {'b': 4}
    histogram('') == {}
    """
```

Confidence Parallel (Fail):

```
def histogram(test):
    letters = test.split()
    counts = {}
    for letter in letters:
        if letter in counts:
            counts[letter] += 1
        else:
            counts[letter] = 1
    max_count = max(counts.values())
    return {letter: count for letter, count in counts.items() if count == max_count}
```

PARD (Pass):

```
def histogram(test):
    if not test:
        return {}
    letters = test.split()
    counts = {}
    for letter in letters:
        if letter in counts:
            counts[letter] += 1
        else:
            counts[letter] = 1
    max_count = max(counts.values())
    return {letter: count for letter, count in counts.items() if count == max_count}
```

Figure 16: Generations of Confidence Parallel and PARD on `HumanEval/111`. The empty-input guard is highlighted in green.

most confident tokens first, so it unmasks `"="`. This constrains the surrounding body and forces the earlier slot to become `letters`, leaving no room for the guard. PARD instead commits positions left to right, so it resolves the earlier slot first and unmasks `"if"` despite its lower confidence, keeping the guard.

Step	Confidence Parallel	PARD
0	`def`	`def`
1	`histogram(test):`	`histogram(test):`
2	`=` (0.28)	`if` (0.20)
⋮	`letters = test.split()` ✗	`if not test: return {}` ✓

Figure 17: Tokens unmasked at the first few decoding step by Confidence Parallel and PARD for the example in Figure 16. Values in parentheses are the predictive confidence in the token.

Table 6: Performance of Fast-dLLM v2 7B, SDAR 8B, and LLaDA2.1-Mini under top-p sampling with temperature 1.0 and $p = 0.8$ on *HumanEval*, *MBPP*, and *GSM8K*. Results are reported as mean $\pm$ standard deviation over three runs. The best result for each model and benchmark is bolded.

Model	Method	HumanEval		MBPP		GSM8K
		Base	Plus	Base	Plus	
Fast-dLLM v2 7B	KLASS	64.0±2.6	58.9±2.0	56.9±0.6	48.1±1.3	82.0±0.6
	EB-Sampler	61.4±2.5	55.5±2.6	58.8±0.4	49.8±1.0	81.3±0.3
	Confidence Parallel	58.1±3.7	53.9±3.7	57.5±1.5	49.1±2.2	81.9±0.4
	PARD	**67.1±1.8**	**62.6±1.2**	**63.4±1.6**	**53.5±1.5**	**82.2±0.9**
SDAR 8B	KLASS	67.3±2.7	63.4±2.2	69.8±1.3	59.1±1.2	90.3±0.6
	EB-Sampler	71.5±2.4	64.8±1.0	72.7±0.8	61.1±1.6	90.0±0.4
	Confidence Parallel	70.9±1.7	66.9±2.5	72.6±1.9	61.3±1.3	89.8±0.3
	PARD	**73.8±2.8**	**67.9±2.4**	**72.9±0.7**	**61.8±1.1**	**90.8±0.5**
LLaDA2.1-Mini	KLASS	73.6±2.9	70.3±2.7	78.4±0.5	65.5±1.5	90.2±0.3
	EB-Sampler	76.6±1.3	74.0±1.0	79.6±0.2	67.7±0.6	90.8±0.4
	Confidence Parallel	79.9±2.5	76.8±2.3	78.2±0.7	66.5±0.2	90.6±0.4
	PARD	**85.8±1.0**	**81.9±0.3**	**82.6±0.3**	**70.6±0.2**	**91.1±0.3**

D.4 TOP-p SAMPLING EXPERIMENTS

The main experiments in Section 5 use greedy decoding (temperature = 0.0). To assess whether PARD's advantage remains robust under stochastic nucleus decoding, we additionally evaluate top-p sampling on *HumanEval*, *MBPP*, and *GSM8K* using temperature = 1.0 and p = 0.8, a configuration adopted in recent LLM reasoning studies (Liu et al., 2025; Jin et al., 2026). We compare PARD against the three most competitive parallel baselines used in our main experiments: Confidence Parallel, KLASS, and EB-Sampler. We repeat the experiment three times and report the mean and standard deviation of pass@1.

Table 6 shows the results on Fast-dLLM v2, SDAR, and LLaDA2.1-Mini. We have two major observations. First, under top-p sampling, almost all samplers perform worse than their greedy counterparts in Table 2. This is consistent with the default greedy decoding adopted by recent BDLMs such as Fast-dLLM v2 and LLaDA2.1, while also avoiding an additional hyperparameter search. Second, PARD achieves the highest mean performance among all parallel samplers, showing that its advantage remains robust under stochastic top-p sampling. We conduct paired t-tests over all models and benchmarks for PARD vs each baseline. PARD significantly outperforms Confidence Parallel, EB-Sampler, and KLASS, with p-values of 3.5×10^{-4}, 3.4×10^{-4}, and 1.7×10^{-4}, respectively. Overall, PARD retains its relative advantage under top-p sampling.