

PV-Decoding: Progressive Verification Decoding with Decode Gating for Diffusion Language Models

Renxi Liu^{1,*}, Yiming Li^{1,*}, Shengyin Sun², Chen Chen¹, Xian Wang¹, Shuai Yang¹,
Huiling Zhen¹, Xianzhi Yu¹, Yaoyuan Wang¹, Weizhe Lin^{1,†}, Xiaosong Li¹

¹Huawei Technologies

²City University of Hong Kong

*Equal contribution.

[†]Corresponding author. Email: linweizhe1@huawei.com

Abstract

Discrete diffusion language models (dLLMs) offer an attractive alternative to autoregressive generation by updating multiple tokens in parallel, but their inference efficiency is still limited by the cost of safely committing tentative predictions. In particular, speculative decoding for dLLMs is challenging because speculative tokens within the same diffusion block may depend on one another, making aggressive acceptance prone to error propagation and strict verification computationally expensive. We present **PV-Decoding**, a confidence-driven speculative decoding framework with progressive verification for block diffusion inference. PV-Decoding partitions masked positions into directly decoded tokens, speculative tokens, and unresolved tokens using two confidence thresholds, and then verifies speculative tokens with progressively stronger policies, including single-batch, dual-batch, and multi-batch verification. This design exposes a practical accuracy–efficiency trade-off: weaker verification minimizes overhead, whereas stronger verification improves acceptance precision and robustness. Experiments on three diffusion language models across GSM8K, MBPP, and HumanEval show that PV-Decoding reduces sequential model forward passes (MFPs) by 8%–37% relative to confidence-based decoding, while keeping the largest observed accuracy drop to 1.3%. On Dream-7B-Instruct/GSM8K, PV-Decoding (1-batch) and PV-Decoding (2-batch) improve throughput by 7.0%–13.6% under the same hardware budget, whereas PV-Decoding (multi-batch) is slower under the current runtime. These results suggest that confidence-guided speculation with structured verification is a practical path toward efficient and reliable dLLM inference.

1 Introduction

Discrete diffusion language models (dLLMs) iteratively denoise masked sequences. This generation

paradigm enables bidirectional conditioning and multi-token parallel updates within each inference step (Nie et al., 2025; Ye et al., 2025; Wu et al., 2025). Under block diffusion, one forward pass predicts an entire block of masked positions rather than a single next token. Such parallelism makes dLLMs attractive for low-latency generation, but it also changes the central inference problem: when many positions are updated simultaneously, the decoder must decide which tentative tokens can be safely committed without degrading the conditioning context for later denoising steps.

Speculative decoding is a natural way to further reduce the number of denoising steps, yet verification is substantially harder for dLLMs than for autoregressive models. A speculative token in a diffusion block is typically conditioned on other tentative tokens produced in the same round. As a result, aggressive acceptance may amplify early mistakes, whereas strict verification can require additional branches whose cost quickly offsets the speedup. The key challenge is therefore not merely to speculate more tokens, but to control the accuracy–efficiency trade-off of verification under limited computation.

To address this challenge, we present **PV-Decoding**, a confidence-driven diffusion decoding framework with progressive verification. PV-Decoding separates *direct decoding* from *speculative filling* according to token confidence, and then verifies speculative tokens with policies of different rigor. This design exposes an explicit operating knob: users can choose lighter or stronger verification depending on their compute budget and tolerance for quality degradation. PV-Decoding is training-free and integrates confidence decoding with verification for mutually conditioned block tokens; unlike lossless methods, it may change the deterministic decoding trajectory.

Our contributions are three-fold:

- We formulate confidence-driven speculation for block diffusion decoding, partitioning masked positions into directly decoded, speculative, and unresolved sets through two confidence thresholds.
- We propose three progressive verification policies—single-batch, dual-batch, and multi-batch verification—with next-step verification and decode gating that provide increasingly strong safeguards against erroneous speculative commitments while incurring progressively increasing verification cost.
- We provide a unified view of speculative decoding for dLLMs in which verification strength, acceptance precision, and runtime overhead can be traded off explicitly across deployment settings. Wall-clock tests under the same hardware budget verify PV-Decoding (1-batch) and PV-Decoding (2-batch) speedups and expose the branch overhead of PV-Decoding (multi-batch).

2 Related Work

Discrete Diffusion Language Models And Block Decoding Foundations. Discrete diffusion and masked diffusion models provide the foundation for denoising over token spaces, from structured discrete diffusion processes with absorbing states (Austin et al., 2021a) to modern masked diffusion language modeling objectives (Sahoo et al., 2024). Recent dLLMs build on this foundation and show that discrete denoising can support competitive language generation while exposing richer parallelism than autoregressive decoding (Nie et al., 2025; Ye et al., 2025; Ascend Tribe, 2025; Google DeepMind, 2025; Inception Labs et al., 2025). Training-free acceleration methods further adapt block diffusion to practical inference by combining block-wise context reuse with confidence-aware decoding (Wu et al., 2025). dParallel introduces certainty-forcing distillation to accelerate masked-token certainty convergence, further improving the parallelism of dLLM decoding (Chen et al., 2026b). These works establish the modeling and decoding foundation on which PV-Decoding operates, but they do not directly address how speculative tokens should be verified once multiple positions are updated in parallel.

Speculative Decoding For dLLMs. Speculative decoding for autoregressive LLMs is commonly formulated as a draft-and-verify procedure: inexpensive proposals from a small model, auxiliary

heads, or feature-level predictors are checked by the target model in parallel so that multiple tokens can be accepted without changing the target decoding distribution (Leviathan et al., 2023; Chen et al., 2023; Cai et al., 2024; Li et al., 2024). Earlier and concurrent parallel-decoding systems also validate multiple proposed continuations in one target-model pass, including blockwise parallel decoding with prefix validation (Stern et al., 2018) and tree-based speculative inference and verification (Miao et al., 2024). In dLLMs, however, speculative verification must account for bidirectional, block-wise conditioning, where tentative updates at different masked positions may mutually influence the verification distribution. Spiffy (Agrawal et al., 2025) adapts speculative decoding to block diffusion by constructing auto-speculative draft states from the dLLM’s own distribution, organizing them with a directed draft graph, and verifying candidate states in parallel while preserving the model’s output distribution. FreeDave (Wu and Zhang, 2026) similarly frames lossless dLLM acceleration as generating parallel-decoded candidate branches and verifying them so as to reproduce the static decoding trajectory without model modification. DMax (Chen et al., 2026a), Rejection Mixing (Ye et al., 2026), and WINO (Hong et al., 2025) are adjacent parallel-decoding approaches because they accept or propagate additional tokens beyond a conservative confidence-based baseline, although their mechanisms are not all framed as speculative verification. Relative to these methods, PV-Decoding focuses on the verification bottleneck itself and exposes a hierarchy of verification policies, allowing the accuracy–efficiency trade-off to be adjusted according to the available compute budget.

A second line of work improves dLLM inference through more aggressive parallel decoding, rather than through explicit speculative-token verification. LoPA (Xu et al., 2025), FDM (Mo et al., 2025) accelerates inference by exploring multiple candidate token-filling orders and selecting the branch with the highest average masked token confidence. LocalLeap (Kong et al., 2025), for example, leverages local determinism propagation to reduce decoding steps. Compared with this line of work, PV-Decoding is less concerned with enlarging the parallel decoding frontier itself and more concerned with how tentative token updates should be verified and committed under different compute budgets.

3 Preliminaries

Discrete Diffusion Language Modeling. Following discrete diffusion and masked diffusion language modeling (Austin et al., 2021a; Sahoo et al., 2024), a discrete diffusion language model (dLLM) generates a token sequence by iteratively denoising masked inputs. Let $\mathbf{x}^{(0)} = (x_1^{(0)}, \dots, x_L^{(0)})$ be the target sequence of length L , where each token belongs to a vocabulary $\mathcal{V}$ and [MASK] denotes the mask symbol. A standard forward process progressively corrupts tokens into masks according to

$$q(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}) = \prod_{i=1}^L q(x_i^{(t)} | x_i^{(t-1)}),$$

with

$$\begin{aligned} q(x_i^{(t)} = [\text{MASK}] | x_i^{(t-1)}) &= \beta_t, \\ q(x_i^{(t)} = x_i^{(t-1)} | x_i^{(t-1)}) &= 1 - \beta_t, \end{aligned}$$

where $\beta_t \in (0, 1)$ is the corruption probability at step t . The reverse process starts from a masked sequence and predicts a less corrupted state through a parameterized denoising model

$$p_\theta(\mathbf{x}^{(t-1)} | \mathbf{x}^{(t)}) = \prod_{i=1}^L p_\theta(x_i^{(t-1)} | \mathbf{x}^{(t)}).$$

In practical dLLM inference, we follow the block diffusion paradigm of Fast-dLLM (Wu et al., 2025), where the current model input consists of a decoded prefix $\mathbf{x}_0$ and a partially masked block $\mathbf{x}_b = (x_{b,1}, \dots, x_{b,B})$ of length B . Let $\mathcal{M} \subseteq \{1, \dots, B\}$ denote the masked positions in the current block. A single forward pass of the denoising model produces a token distribution for every masked position:

$$p_i(v) = p_\theta(x_{b,i} = v | [\mathbf{x}_0; \mathbf{x}_b]), \quad i \in \mathcal{M}, v \in \mathcal{V}. \quad (1)$$

We use

$$c_i = \max_{v \in \mathcal{V}} p_i(v), \quad \hat{x}_{b,i} = \arg \max_{v \in \mathcal{V}} p_i(v), \quad (2)$$

to denote the confidence and argmax prediction at position i , respectively. Thus, block diffusion can be viewed as the inference-time realization of the dLLM’s parallel denoising capability, and these quantities form the basic interfaces shared by the baseline decoder and our method.

Confidence Decoding Baseline. A standard confidence-based block decoder commits all masked positions whose confidence exceeds a decoding threshold τ_{dec} . Formally, the set of directly decodable positions is

$$\mathcal{D} = \{i \in \mathcal{M} \mid c_i \geq \tau_{\text{dec}}\}. \quad (3)$$

For every $i \in \mathcal{D}$, the decoder commits $\hat{x}_{b,i}$ in parallel. If $\mathcal{D} = \emptyset$, the decoder falls back to greedy decoding and commits only the most confident masked position,

$$i^* = \arg \max_{i \in \mathcal{M}} c_i. \quad (4)$$

This baseline is important for two reasons. First, it is the practical decoding rule used by existing block-diffusion systems. Second, PV-Decoding is defined as an extension of this baseline: instead of committing only high-confidence tokens, it also tentatively fills moderate-confidence tokens and then verifies them in the next step.

4 Methodology

4.1 Problem Formulation

PV-Decoding is designed for block diffusion decoding, where one forward pass updates multiple masked positions at once. This parallelism is beneficial for efficiency, but it also introduces a verification challenge that does not arise in standard autoregressive decoding. In particular, a speculative token may be conditioned on other tentative tokens proposed in the same round. If such tokens are committed too aggressively, errors can contaminate the context used for subsequent decoding steps, and stronger verification mechanisms incur heavier verification cost. Observing that different types of tasks entail varying requirements for inference accuracy, we propose PV-Decoding, a method that couples progressive multi-level verification with corresponding verification computation cost.

Figure 1 summarizes the overall idea. Given a partially masked block, one model pass produces confidence scores for all masked positions. PV-Decoding then partitions them into three groups: directly decoded tokens, speculative tokens, and unresolved masked tokens. Directly decoded tokens are committed immediately, while speculative tokens are only tentatively filled and must be verified in the next iteration. The role of verification is therefore to decide how much speculative

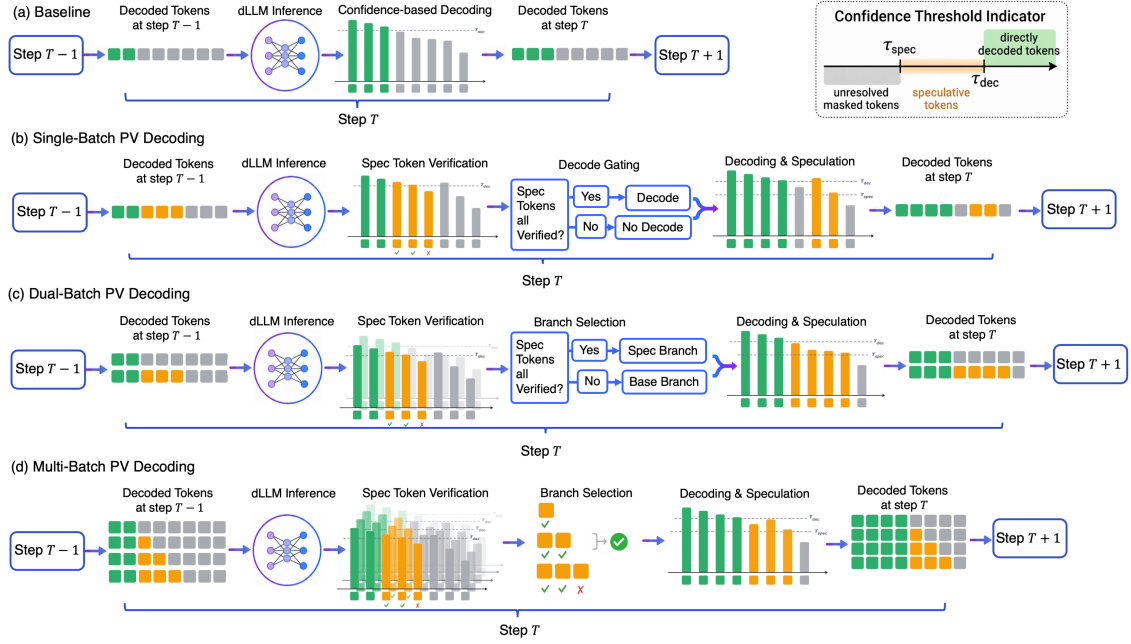


Figure 1: Overview of PV-Decoding under block diffusion decoding. Given an input sequence with partially masked tokens, one forward pass produces token predictions and confidence scores. Two thresholds, τ_{spec} and τ_{dec} , partition masked positions by confidence c_i into unresolved tokens ($c_i < \tau_{\text{spec}}$), speculative tokens ($\tau_{\text{spec}} \leq c_i < \tau_{\text{dec}}$), and directly decoded tokens ($c_i \geq \tau_{\text{dec}}$). The speculative tokens are then processed by progressively stronger verification policies: single-batch, dual-batch, and multi-batch verification.

context can be trusted before more tokens are committed.

This view motivates a two-stage design. First, PV-Decoding performs *confidence-based speculation*, which enlarges the set of candidate updates beyond the baseline confidence decoder. Second, it applies a *verification policy* that controls how conservatively these tentative updates are accepted. Stronger policies improve acceptance precision but consume more verification compute.

Tentative tokens may affect later predictions before verification; hence PV-Decoding is deterministic but neither trajectory- nor distribution-preserving.

4.2 Confidence-Based Speculation

Building on the notation in Section 3, PV-Decoding introduces a second threshold $\tau_{\text{spec}} < \tau_{\text{dec}}$. At decoding step r , each masked position $i \in \mathcal{M}^{(r)}$ is assigned to one of three sets:

$$\mathcal{D}^{(r)} = \left\{ i \in \mathcal{M}^{(r)} \mid c_i^{(r)} \geq \tau_{\text{dec}} \right\}, \quad (5)$$

$$\mathcal{S}^{(r)} = \left\{ i \in \mathcal{M}^{(r)} \mid \tau_{\text{spec}} \leq c_i^{(r)} < \tau_{\text{dec}} \right\}, \quad (6)$$

$$\mathcal{U}^{(r)} = \mathcal{M}^{(r)} \setminus \left(\mathcal{D}^{(r)} \cup \mathcal{S}^{(r)} \right), \quad (7)$$

where $\mathcal{D}^{(r)}$, $\mathcal{S}^{(r)}$, and $\mathcal{U}^{(r)}$ denote the directly decoded, speculative, and unresolved positions, respectively.

For every $i \in \mathcal{D}^{(r)} \cup \mathcal{S}^{(r)}$, PV-Decoding tentatively uses the argmax prediction $\hat{x}_{b,i}^{(r)} = \arg \max_v p_i^{(r)}(v)$. The distinction is in commitment: positions in $\mathcal{D}^{(r)}$ are committed immediately, whereas positions in $\mathcal{S}^{(r)}$ are only inserted as speculative placeholders for the next verification step. As illustrated in Figure 1, this rule separates easy predictions from moderate-confidence predictions while leaving low-confidence positions masked.

4.3 Progressive Verification Policy

After speculation, PV-Decoding verifies the tokens proposed in the previous iteration. Let $\mathcal{S}_{\text{prev}}$ denote the previously speculated positions and let $\hat{x}_{b,i}^{\text{prev}}$ be the token proposed for position $i \in \mathcal{S}_{\text{prev}}$. A verification policy accepts a subset of these positions only when the current forward pass predicts the same token with confidence at least τ_{dec} . We refer to these verified and committed positions as *accepted speculative tokens*; they are distinct from directly decoded tokens, which cross τ_{dec} before speculation, and from unaccepted speculative tokens, which are re-masked. The difference between our three variants lies in how much speculative context is used during this check.

Single-Batch Verification. Single-batch verification uses only the standard decoding pass and therefore has the lowest overhead. All specula-

tive tokens are checked jointly in one branch. A speculative token is accepted when the current argmax matches the previous proposal and its confidence exceeds τ_{dec} ; otherwise, that position is re-masked. Importantly, PV-Decoding commits new high-confidence unresolved positions in $\mathcal{U}^{(r)}$ only when *all* speculative tokens from the previous step are verified. This verification-gated rule prevents the decoder from committing fresh tokens on top of a partially incorrect speculative context. Its effective verification batch size is 1.

Dual-Batch Verification. Dual-batch verification evaluates two branches in parallel: a *baseline branch* without speculative tokens and a *speculative branch* with all speculative tokens filled in. The speculative branch is accepted only if every speculative token passes token-consistency and confidence checks; otherwise, decoding falls back to the baseline branch. Relative to single-batch verification, this policy improves acceptance precision because the fallback branch is guaranteed to remain speculation-free, while the verification cost increases to an effective verification batch size of 2.

Multi-Batch Verification. Multi-batch verification further strengthens conditioning by verifying speculative tokens over a subset graph. Following the subset construction of Spiffy (Agrawal et al., 2025), each graph node represents a subset of the previously speculative positions. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be this directed acyclic graph (DAG), where each node $v \in \mathcal{V}$ stores a subset $\mathcal{C}(v) \subseteq \mathcal{S}_{\text{prev}}$ and an edge $u \rightarrow v$ indicates $\mathcal{C}(u) \subset \mathcal{C}(v)$. The root is the empty subset, $\mathcal{C}(v_0) = \emptyset$, which corresponds to the baseline branch with all previous speculative positions masked. For a verification budget of $L + 1$ branches, PV-Decoding selects at most L nested non-root nodes from this graph, fills the corresponding speculative tokens in each branch, and evaluates all selected branches in one verification batch. A node is verified only if all speculative tokens in the preceding branch have already been accepted and the current branch verifies the same token with confidence above the decoding threshold τ_{dec} . Among verified nodes, PV-Decoding chooses the node with the largest accepted subset, using cardinality as the primary key and the graph order as a deterministic tie-breaker; this branch becomes the new baseline context, while speculative positions outside the chosen subset are re-masked. Thus L controls the number of subset nodes tested, and the effective verification batch

Algorithm 1 Unified PV-Decoding

Require: Diffusion LLM f_θ , prefix $\mathbf{x}_0$, initial block $\mathbf{x}_b^{(0)}$
Require: Decode threshold τ_{dec} , speculation threshold τ_{spec} , verification policy π
Ensure: Fully decoded block $\mathbf{x}_b^*$

```

1:  $\mathbf{x}_b \leftarrow \mathbf{x}_b^{(0)}$ 
2:  $\mathcal{S}_{\text{prev}} \leftarrow \emptyset; \{\hat{x}_{b,i}^{\text{prev}}\} \leftarrow \emptyset$ 
3: while  $\mathbf{x}_b$  still contains [MASK] do
4:   Verify previous speculative tokens before selecting
     any new direct or speculative positions.
5:   if  $\mathcal{S}_{\text{prev}} \neq \emptyset$  then
6:      $(\mathbf{x}_b, \alpha, \mathbf{p}) \leftarrow \text{VERIFYPOLICY}(\pi, \mathbf{x}_0, \mathbf{x}_b,$ 
        $\mathcal{S}_{\text{prev}}, \{\hat{x}_{b,i}^{\text{prev}}\}, \tau_{\text{dec}})$ 
7:      $\mathcal{S}_{\text{prev}} \leftarrow \emptyset; \{\hat{x}_{b,i}^{\text{prev}}\} \leftarrow \emptyset$ 
8:     if  $\alpha = \text{FALSE}$  then
9:       continue
10:    end if
11:  else
12:     $\mathbf{p} \leftarrow f_\theta([\mathbf{x}_0; \mathbf{x}_b])$ 
13:  end if
14:   $\mathcal{S}_{\text{next}} \leftarrow \emptyset; \{\hat{x}_{b,i}^{\text{next}}\} \leftarrow \emptyset$ 
15:  for each masked position  $i$  in  $\mathbf{x}_b$  do
16:     $c_i \leftarrow \max_v p_i(v); \hat{x}_{b,i} \leftarrow \arg \max_v p_i(v)$ 
17:    if  $c_i \geq \tau_{\text{dec}}$  then
18:      commit  $\hat{x}_{b,i}$  into  $\mathbf{x}_b$ 
19:    else if  $\tau_{\text{spec}} \leq c_i < \tau_{\text{dec}}$  then
20:      tentatively fill  $\hat{x}_{b,i}$  and add  $i$  to  $\mathcal{S}_{\text{next}}$ 
21:      store  $\hat{x}_{b,i}$  in  $\{\hat{x}_{b,i}^{\text{next}}\}$ 
22:    end if
23:  end for
24:   $\mathcal{S}_{\text{prev}} \leftarrow \mathcal{S}_{\text{next}}; \{\hat{x}_{b,i}^{\text{prev}}\} \leftarrow \{\hat{x}_{b,i}^{\text{next}}\}$ 
25: end while
26: return  $\mathbf{x}_b^* \leftarrow \mathbf{x}_b$ 

```

size is $L + 1$ including the root baseline branch. Unlike Spiffy, PV-Decoding allows the number of speculative tokens to vary adaptively through τ_{spec} and is not tied to a particular DAG construction; we use Spiffy’s subset DAG here for convenience, while alternative DAG-selection strategies can also be used.

Unified Perspective. The three policies therefore form a clear accuracy–efficiency spectrum: single-batch verification has the lowest overhead but the weakest conditioning; dual-batch verification adds one extra branch to reduce spurious acceptance; multi-batch verification applies the strongest conditioning at the highest verification cost. In all cases, speculation is still driven by the same confidence rule, while verification determines how aggressively tentative updates can be committed.

5 Experiments

5.1 Experimental Settings

For brevity, we refer to PV-Decoding (1-batch), PV-Decoding (2-batch), and PV-Decoding (multi-batch) as PV-1, PV-2, and PV-multi, respectively.

All experiments are conducted on Huawei Ascend 910B3 NPU hardware. We evaluate PV-

Algorithm 2 Progressive Verification Policy

Require: Policy type $\pi \in \{\text{single}, \text{dual}, \text{multi}\}$, prefix $\mathbf{x}_0$, block $\mathbf{x}_b$
Require: Speculative positions $\mathcal{S}_{\text{prev}}$, tokens $\{\hat{x}_{b,i}^{\text{prev}}\}$, decode threshold τ_{dec}
Ensure: Updated block $\mathbf{x}_b$, new-commit flag α , and branch distribution $\mathbf{p}$

- 1: Return the branch distribution used after verification so the decoder does not reuse stale pre-verification probabilities.
- 2: **if** $\pi = \text{single}$ **then**
- 3: $\mathbf{p} \leftarrow f_{\theta}([\mathbf{x}_0; \mathbf{x}_b])$
- 4: **accept** i **iff** $\arg \max_v p_i(v) = \hat{x}_{b,i}^{\text{prev}}$ and $p_i(\hat{x}_{b,i}^{\text{prev}}) \geq \tau_{\text{dec}}$
- 5: re-mask rejected positions; set α to whether all positions were accepted
- 6: **else if** $\pi = \text{dual}$ **then**
- 7: evaluate a baseline branch $\mathbf{p}^{(0)}$ and a speculative branch $\mathbf{p}^{(1)}$
- 8: **if** all speculative tokens pass consistency and confidence checks in $\mathbf{p}^{(1)}$ **then**
- 9: keep the speculative branch and set $\mathbf{p} \leftarrow \mathbf{p}^{(1)}$
- 10: **else**
- 11: fall back to the baseline branch, re-mask speculative positions, and set $\mathbf{p} \leftarrow \mathbf{p}^{(0)}$
- 12: **end if**
- 13: $\alpha \leftarrow \text{TRUE}$
- 14: **else**
- 15: build a subset DAG $\mathcal{G}$ over $\mathcal{S}_{\text{prev}}$ with root subset $\emptyset$
- 16: select up to L non-root nodes $\{v_{\ell}\}_{\ell=1}^L$ from $\mathcal{G}$
- 17: evaluate one verification batch containing the root branch and branches with $\mathcal{C}(v_{\ell})$ filled
- 18: mark the root verified by default
- 19: mark v_{ℓ} verified **iff** all tokens in $\mathcal{C}(v_{\ell})$ pass consistency and confidence checks
- 20: choose the verified node v^* with largest $|\mathcal{C}(v^*)|$, using graph order to break ties
- 21: **accept** $\mathcal{C}(v^*)$, re-mask $\mathcal{S}_{\text{prev}} \setminus \mathcal{C}(v^*)$, and set $\mathbf{p}$ to the distribution of branch v^*
- 22: $\alpha \leftarrow \text{TRUE}$
- 23: **end if**
- 24: **return** $(\mathbf{x}_b, \alpha, \mathbf{p})$

Decoding on three diffusion language models: LLaDA-8B-Instruct (Nie et al., 2025), Dream-7B-Instruct (Ye et al., 2025), and openPangu-7B-Diffusion-Base (Ascend Tribe, 2025). Evaluation is performed on GSM8K (Cobbe et al., 2021), MBPP (Austin et al., 2021b), and HumanEval (Chen et al., 2021), which together cover mathematical reasoning and code generation.

Following Fast-dLLM (Wu et al., 2025), we use block diffusion with KV-cache reuse. The diffusion block size is set to 32 tokens. All evaluations are implemented using the Language Model Evaluation Harness (lm_eval) (Gao et al., 2023). We use a batch size of 1 for accuracy and model forward pass (MFP) evaluation, while wall-clock performance is measured under varying levels of concurrency. The baseline decoding threshold is fixed at $\tau_{\text{dec}} = 0.9$ in accordance with Fast-dLLM, and the maximum number of generated tokens is

set to 512. For PV-Decoding, the default speculative threshold is $\tau_{\text{spec}} = 0.8$. For multi-batch verification, the maximum number of verification branches is capped at 8. This threshold pair is used throughout without task-specific tuning.

We compare PV-Decoding against Fast-dLLM and representative accelerated decoding baselines, including LoPA (Xu et al., 2025), Spiffy (Agrawal et al., 2025), and LocalLeap (Kong et al., 2025). LoPA uses 7 branches and Spiffy 8. Cross-method comparisons use MFPs as the efficiency metric. Table 2 separately reports hardware-matched wall-clock results for Fast-dLLM and PV-Decoding.

5.2 Main Results

The overall accuracy–efficiency trade-off is summarized in Figure 2. To emphasize the general trend rather than model-specific variance, each point aggregates the corresponding benchmark–method result over the three evaluated diffusion language models. The x-axis reports relative decoding efficiency, defined as the ratio between the number of greedy MFPs and that required by each method, i.e., $S_{\text{greedy}}/S_{\text{method}}$. The y-axis reports ΔAcc relative to greedy decoding, so positive values indicate accuracy gains over greedy decoding and negative values indicate accuracy drops. Hence, the upper-right region indicates a favorable accuracy–efficiency trade-off. The efficiency reported here is measured in terms of sequential model forward passes. Achieving the optimal efficiency of the 2-branch and multi-branch variants would require either optimized operators or a proportional increase in hardware resources.

Table 1 reports the underlying model-specific accuracy and sequential MFP measurements behind the model-averaged Pareto view.

Compared with both greedy decoding and confidence-based decoding in Fast-dLLM, PV-Decoding consistently improves sequential-MFP efficiency. In particular, PV-Decoding reduces the number of model forward passes by approximately 8%–37% relative to confidence-based decoding, with PV-1 providing the largest step reduction. Even PV-2, despite its stricter all-or-nothing verification rule, still reduces model forward passes by 7%–22% relative to the same baseline.

The relative decoding efficiency in Figure 2 also reflects the expected ordering among PV-Decoding variants: overall, PV-1 achieves the highest step-based efficiency, followed by PV-multi and then PV-2. This trend is consistent

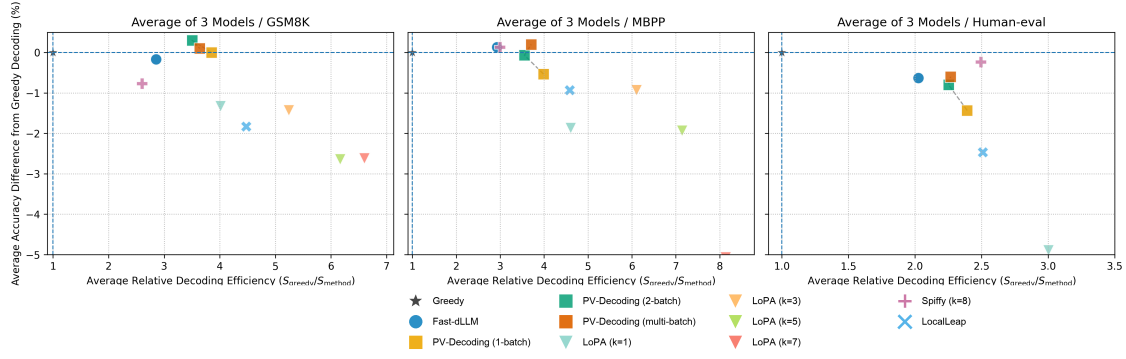


Figure 2: Model-averaged accuracy–efficiency trade-off across benchmarks. For each benchmark and method, relative decoding efficiency and ΔAcc are averaged over LLaDA-8B-Instruct, Dream-7B-Instruct, and openPangu-7B-Diffusion-Base. The x-axis shows the relative decoding efficiency, computed as the ratio between the number of sequential model forward passes (MFPs) required by greedy decoding and that required by each method. The y-axis reports the accuracy difference with respect to greedy decoding.

with their verification policies. PV-1 verifies all draft tokens within a single branch and therefore minimizes verification overhead; PV-2 uses an all-or-nothing comparison between a baseline branch and a speculative branch, which is more conservative and yields smaller step reductions; and PV-multi verifies nested subsets of draft tokens, allowing partial speculative progress when the full speculative branch is not accepted and thus typically placing its efficiency between the other two variants.

In terms of accuracy, PV-Decoding exhibits a clear trade-off governed by verification cost. Stronger verification generally improves stability, but small differences preclude a strict ordering. On GSM8K, all three variants achieve accuracy comparable to or slightly better than confidence-based decoding. On the programming benchmarks MBPP and HumanEval, PV-1 is less stable across models, with the largest observed ΔAcc drop relative to confidence-based decoding being 1.3% on openPangu-7B-Diffusion-Base for HumanEval. PV-2 and PV-multi generally narrow this gap by applying stricter verification: the PV-2 loss is below 1%, while PV-multi generally best preserves accuracy relative to Fast-dLLM. Overall, the re-

sults support the intended design of PV-Decoding: lighter verification yields larger step reduction, whereas stronger verification improves robustness.

This progressive verification design also makes PV-Decoding adaptable to different deployment scenarios. For tasks where small speculative deviations are less costly, such as GSM8K-style mathematical reasoning, PV-1 is often attractive because it incurs the lowest verification cost and provides the largest reduction in model forward passes while maintaining comparable accuracy. For accuracy-sensitive settings, including code generation and agentic workflows where a single incorrect token may propagate into execution failures or invalid tool calls, PV-2 or PV-multi provides a more conservative choice. These variants spend additional verification computation to reduce speculative errors, making them better suited when robustness is prioritized over maximum step reduction.

5.3 Wall-Clock Performance

We evaluate wall-clock performance from a fixed-resource deployment perspective. Table 2 measures Fast-dLLM and all PV-Decoding variants in the same SGLang runtime on Dream-7B-

Table 1: Main accuracy and model-forward-pass results across diffusion language models and benchmarks. Accuracy is reported in percentage, and MFP denotes the average number of model forward passes. Parenthetical changes of the accuracy and MFP in the PV-Decoding columns are reported relative to Fast-dLLM.

Model	Dataset	Greedy		Fast-dLLM		PV-Decoding (1-batch)		PV-Decoding (2-batch)		PV-Decoding (multi-batch)	
		Acc (%)	MFP	Acc (%)	MFP	Acc (%)	MFP	Acc (%)	MFP	Acc (%)	MFP
Dream-7B-Instruct	GSM8K	72.4	122.4	72.4	58.3	72.4 (+0.0)	46.2 (-12.1)	72.9 (+0.5)	52.3 (-6.0)	72.3 (-0.1)	46.5 (-11.8)
	MBPP	55.4	256.0	55.4	62.9	54.8 (-0.6)	42.7 (-20.2)	55.4 (+0.0)	47.2 (-15.7)	55.4 (+0.0)	45.6 (-17.3)
	HumanEval	56.1	173.6	54.8	86.4	54.3 (-0.5)	75.5 (-10.9)	54.3 (-0.5)	78.8 (-7.6)	54.9 (+0.1)	78.2 (-8.2)
LLaDA-8B-Instruct	GSM8K	78.5	210.0	78.2	77.1	78.2 (+0.0)	70.9 (-6.2)	78.5 (+0.3)	62.2 (-14.9)	78.8 (+0.6)	70.7 (-6.4)
	MBPP	36.2	56.0	36.6	33.2	36.4 (-0.2)	27.6 (-5.6)	36.4 (-0.2)	28.1 (-5.1)	36.8 (+0.2)	28.4 (-4.8)
	HumanEval	37.2	160.0	36.6	74.9	36.0 (-0.6)	63.5 (-11.4)	36.6 (+0.0)	64.5 (-10.4)	36.6 (+0.0)	64.6 (-10.3)
openPangu-7B-Diffusion-Base	GSM8K	77.8	407.1	77.6	108.8	78.1 (+0.5)	68.5 (-40.3)	78.2 (+0.6)	85.0 (-23.8)	77.9 (+0.3)	76.6 (-32.2)
	MBPP	51.4	511.0	51.4	169.0	50.2 (-1.2)	129.9 (-39.1)	51.0 (-0.4)	157.7 (-11.3)	51.4 (+0.0)	144.7 (-24.3)
	HumanEval	47.0	141.6	47.0	73.4	45.7 (-1.3)	60.1 (-13.3)	47.0 (+0.0)	68.3 (-5.1)	47.0 (+0.0)	67.3 (-6.1)

Table 2: Fixed-hardware performance on Dream-7B-Instruct/GSM8K using one Ascend NPU. Parentheses show changes from Fast-dLLM under the same concurrency.

Concurrency	Throughput (tok/s) $\uparrow$				Total evaluation time (s) $\downarrow$			
	Fast-dLLM	PV-Decoding (1-batch)	PV-Decoding (2-batch)	PV-Decoding (multi-batch)	Fast-dLLM	PV-Decoding (1-batch)	PV-Decoding (2-batch)	PV-Decoding (multi-batch)
1	183.8	208.7 (+13.6%)	204.9 (+11.5%)	134.8 (−26.6%)	338.29	294.73 (−12.9%)	300.85 (−11.1%)	458.51 (+35.5%)
4	280.6	306.3 (+9.1%)	303.0 (+8.0%)	155.0 (−44.8%)	220.22	199.41 (−9.5%)	201.86 (−8.3%)	395.36 (+79.5%)
8	311.5	341.4 (+9.6%)	333.3 (+7.0%)	116.5 (−62.6%)	197.26	179.01 (−9.3%)	182.91 (−7.3%)	524.30 (+165.8%)

Instruct/GSM8K, using one Ascend NPU at request concurrencies 1, 4, and 8. Holding the hardware budget fixed gives a direct comparison of the speedups available to a user without provisioning additional accelerators. It is nevertheless a deliberately resource-constrained view: PV-2 and PV-multi trade additional parallel branch computation for fewer sequential MFPs, so a single shared NPU—especially at high concurrency—cannot necessarily expose all of their available branch-level parallelism.

Under this fixed hardware budget, PV-1 and PV-2 consistently improve throughput by 7.0%–13.6% and reduce total evaluation time by 7.3%–12.9% across all tested concurrencies. PV-2 is slightly slower than PV-1, consistent with the additional parallel work introduced by its second verification branch; nevertheless, its positive end-to-end speedup shows that the reduction in sequential decoding rounds already outweighs this overhead. PV-multi does not reach this crossover point in the current system: its larger set of verification branches reduces sequential MFPs but lowers throughput by 26.6%–62.6% and increases evaluation time by 35.5%–165.8%. Thus, the fixed-budget results establish practical acceleration for PV-1 and PV-2, but not for PV-multi.

These measurements reflect our current implementation rather than the full systems potential of branch-parallel verification. We implement each verification branch as an ordinary batch element, which incurs batching overhead, repeated memory and attention work across branches under the limited NPU capacity. This simple implementation leaves substantial optimization space analogous to tree-structured speculative verification: branches share a common context with potentially reusable states. Packing the branches into a tree-structured attention layout and verifying them with fused operators instead of independent batched forwards (Miao et al., 2024) can maximize the performance of multi-branch verification under fixed hardware budget. Such specialized kernels are particularly important for realizing PV-2 and PV-multi speedups; additional hardware par-

allelism is another possible operating point but is not evaluated here. We therefore interpret Table 2 together with the sequential-MFP results: the former reports speedups realized by the present single-hardware implementation, while the latter quantifies algorithmic depth reduction whose conversion into further wall-clock gains depends on more efficient branch execution.

5.4 Comparison with Prior Methods

In the model-averaged comparison with prior methods, PV-Decoding is competitive in accuracy versus sequential MFPs. LocalLeap achieves relatively large reductions in MFPs. However, these reductions often come with substantial accuracy degradation, especially on GSM8K and HumanEval. LoPA exhibits unstable accuracy across settings and causes accuracy drops exceeding 5% in some cases. Spiffy is competitive in selected scenarios, but it relies on multiple verification branches to verify multiple draft tokens, so its step reduction is closely tied to the branch budget k ; increasing k can expose more speculative progress, but also requires more verification computation. Moreover, its relative decoding efficiency is less stable across models and benchmarks.

Across most settings, PV-2 and PV-multi retain sequential-MFP reductions while improving robustness in task accuracy. PV-1 incurs no extra verification branch, PV-2 requires only one extra branch, and PV-multi increases branch count only as needed.

6 Discussion

6.1 Ablation on the Speculative Confidence Threshold

Figure 3 examines the speculative confidence threshold τ_{spec} on Dream-7B-Instruct with $\tau_{\text{dec}} = 0.9$. Lower thresholds enlarge the speculative set and reduce diffusion decoding steps, but also make speculative tokens less reliable, especially on programming benchmarks.

The figure shows a consistent accuracy–efficiency trade-off. When $\tau_{\text{spec}} = 0.6$ or 0.7 ,

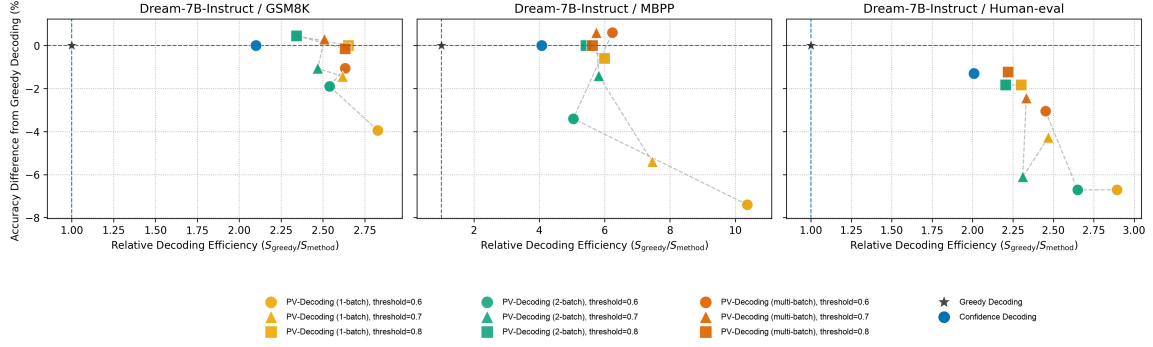


Figure 3: Accuracy–efficiency trade-off on Dream-7B-Instruct under different speculative confidence thresholds $\tau_{\text{spec}} \in \{0.6, 0.7, 0.8\}$.

Table 3: Ablation of verification-gated decoding in PV-Decoding (1-batch). “w/o decode gating” allows new high-confidence positions to be committed even when some speculative tokens fail verification.

Dataset	Method	Acc	MFP
GSM8K	Fast-dLLM	72.4	58.3
	PV-Decoding (1-batch)	72.4	46.2
	PV-Decoding (1-batch) w/o decode gating	70.6	49.9
HumanEval	Fast-dLLM	54.8	86.4
	PV-Decoding (1-batch)	54.3	75.5
	PV-Decoding (1-batch) w/o decode gating	50.0	70.8
MBPP	Fast-dLLM	55.4	62.9
	PV-Decoding (1-batch)	54.8	42.7
	PV-Decoding (1-batch) w/o decode gating	51.3	35.8

PV-Decoding often reaches higher relative decoding efficiency but suffers larger ΔAcc drops. Setting $\tau_{\text{spec}} = 0.8$ yields a more conservative policy: it sacrifices part of the step reduction while keeping ΔAcc close to confidence-based decoding. Stronger verification further mitigates errors caused by aggressive speculation, as dual-batch and multi-batch verification generally improve accuracy over single-batch verification at the same threshold. All main experiments use $(\tau_{\text{dec}}, \tau_{\text{spec}}) = (0.9, 0.8)$ without task-specific tuning; the thresholds are fixed, but the selected token set varies by input and step.

6.2 Ablation on Verification-Gated Decoding

We further ablate the verification-gated decoding rule in single-batch verification. This rule commits new masked positions with confidence above τ_{dec} only when all speculative tokens from the previous step pass verification; otherwise, the current step avoids committing additional direct tokens on an unreliable speculative context.

Table 3 shows that verification-gated decoding primarily serves as a robustness mechanism for the low-overhead PV-1 variant. Decode gating improves accuracy on GSM8K, HumanEval, and MBPP, indicating that suppressing new direct commits after failed verification reduces error propaga-

tion from incorrect speculative tokens.

7 Conclusion

We presented PV-Decoding, a confidence-driven speculative decoding framework for discrete diffusion language models. PV-Decoding extends confidence-based block decoding by tentatively filling medium-confidence tokens and verifying them through progressively stronger policies. This design makes the accuracy–efficiency trade-off explicit: single-batch verification offers the lowest-overhead acceleration, while dual-batch and multi-batch verification provide stronger safeguards against erroneous speculative commitments. Experiments across models and benchmarks show that PV-Decoding reduces model forward passes while maintaining competitive accuracy. Hardware-matched tests verify PV-1/PV-2 speedups; PV-multi requires optimized branch execution for wall-clock gains.

8 Limitations

PV-Decoding relies on confidence scores to select medium-confidence positions, so its effectiveness depends on model calibration and task sensitivity; adaptive speculative thresholds remain future work. Programming benchmarks also appear more vulnerable than GSM8K to incorrect speculative commitments, although a detailed error taxonomy is still needed. Finally, practical gains depend on efficient verification-branch batching. PV-1 and PV-2 add zero and one branch, respectively, and therefore have clearer paths to acceleration, whereas PV-multi incurs greater overhead and may require specialized kernels, efficient batching, or additional compute to translate its algorithmic gains into wall-clock speedups.

References

- Sudhanshu Agrawal, Risheek Garrepalli, Raghav Goel, Mingu Lee, Christopher Lott, and Fatih Porikli. 2025. *Spiffy: Multiplying diffusion llm acceleration via lossless speculative decoding*. *arXiv preprint arXiv:2509.18085*.
- Ascend Tribe. 2025. openpangu-7b-diffusion-base: Openpangu 7b diffusion base language model. <https://ai.gitcode.com/ascend-tribe/openPangu-7B-Diffusion-Base>.
- Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021a. *Structured denoising diffusion models in discrete state-spaces*. In *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993.
- 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*. *arXiv preprint arXiv:2108.07732*.
- Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. *Medusa: Simple LLM inference acceleration framework with multiple decoding heads*. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5209–5235. PMLR.
- Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. *Accelerating large language model decoding with speculative sampling*. *arXiv preprint arXiv:2302.01318*.
- 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*. *arXiv preprint arXiv:2107.03374*.
- Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. 2026a. *Dmax: Aggressive parallel decoding for dllms*. *arXiv preprint arXiv:2604.08302*.
- Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. 2026b. *dParallel: Learnable parallel decoding for dLLMs*. In *The Fourteenth International Conference on Learning Representations*.
- 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*. *arXiv preprint arXiv:2110.14168*.
- 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, and 5 others. 2023. *A framework for few-shot language model evaluation*.
- Google DeepMind. 2025. Gemini diffusion. <https://deepmind.google/models/gemini-diffusion/>.
- Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. 2025. *Wide-in, narrow-out: Revokable decoding for efficient and effective dllms*. *arXiv preprint arXiv:2507.18578*.
- Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. 2025. *Mercury: Ultra-fast language models based on diffusion*. *arXiv preprint arXiv:2506.17298*.
- Fanheng Kong, Jingyuan Zhang, Yahui Liu, Zirui Wu, Yu Tian, Victoria W., and Guorui Zhou. 2025. *Accelerating diffusion llm inference via local determinism propagation*. *Preprint*, arXiv:2510.07081.
- Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. *Fast inference from transformers via speculative decoding*. *arXiv preprint arXiv:2211.17192*.
- Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. *EAGLE: Speculative sampling requires rethinking feature uncertainty*. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28935–28948. PMLR.
- Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Danish Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. *SpecInfer: Accelerating generative large language model serving with tree-based speculative inference and verification*. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 932–949.
- Yichuan Mo, Quan Chen, Mingjie Li, Zeming Wei, and Yisen Wang. 2025. *Decoding large language diffusion models with foreseeing movement*. *Preprint*, arXiv:2512.04135.
- Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Jirong Wen, and Chongxuan Li. 2025. *Large language diffusion models*. *arXiv preprint arXiv:2502.09992*.
- Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu,

- Alexander M. Rush, and Volodymyr Kuleshov. 2024. [Simple and effective masked diffusion language models](#). In *Advances in Neural Information Processing Systems*.
- Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. [Blockwise parallel decoding for deep autoregressive models](#). In *Advances in Neural Information Processing Systems*, volume 31.
- 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](#). *Preprint*, arXiv:2505.22618.
- Shutong Wu and Jiawei Zhang. 2026. [Free draft-and-verification: Toward lossless parallel decoding for diffusion large language models](#). *Preprint*, arXiv:2510.00294.
- Chenkai Xu, Yijie Jin, Jiajun Li, Yi Tu, Guoping Long, Dandan Tu, Mingcong Song, Hongjie Si, Tianqi Hou, Junchi Yan, and Zhijie Deng. 2025. [Lopa: Scaling dllm inference via lookahead parallel decoding](#). *arXiv preprint arXiv:2512.16229*.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. [Dream 7b: Diffusion large language models](#). *arXiv preprint arXiv:2508.15487*.
- Yushi Ye, Feng Hong, Huangjie Zheng, Xu Chen, Zhiyong Chen, Yanfeng Wang, and Jiangchao Yao. 2026. Rejection mixing: Fast semantic propagation of mask tokens for efficient dllm inference. *arXiv preprint arXiv:2602.22868*.