Cluster-Level Attention-Guided Parallel Decoding for Masked Diffusion Language Models

Anonymous ACL submission

Abstract

Masked diffusion language models (MDLMs) enable parallel decoding by predicting all masked positions at each denoising step, yet existing training-free samplers usually decide which positions to commit at token-level granularity. We revisit this granularity and observe that reliable predictions often emerge as contiguous high-confidence spans, suggesting that the unit of parallel commitment can be larger than a single token. We first group adjacent high-confidence candidates into confidence-induced clusters (CICs) as span-level update units. We then use self-attention maps from the same forward pass to estimate inter-cluster dependencies, enabling conflict-aware selection of mutually compatible CICs for parallel commitment. This yields CLAD (**C**luster-**L**evel **A**ttention-Guided **D**ecoding), a training-free cluster-level decoder for MDLMs. Experiments on LLaDA and Dream model families across four reasoning and code-generation benchmarks show that CLAD achieves $1.77\times$–$8.47\times$ speedups over Vanilla decoding while maintaining broadly comparable task accuracy in most settings.

1 Introduction

Masked diffusion language models (MDLMs) (Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2025; Nie et al., 2025; Ye et al., 2025) have recently emerged as a promising alternative to autoregressive language models (Brown et al., 2020; Ouyang et al., 2022; Achiam et al., 2023; Bai et al., 2023; Grattafiori et al., 2024; Guo et al., 2025) for text generation. Instead of generating tokens strictly from left to right, MDLMs start from a masked sequence and iteratively recover masked positions. Since all currently masked positions can be predicted within each denoising step, the decoding policy must decide which predictions should be committed and used as context for subsequent steps.

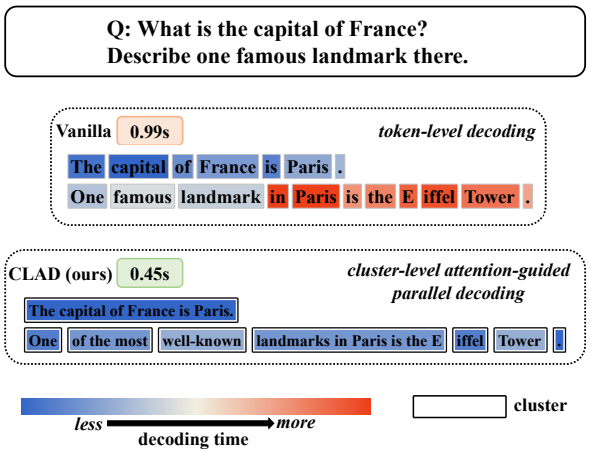


Figure 1: Qualitative comparison of Vanilla decoding and CLAD. Colors indicate relative decoding time, with cooler colors denoting earlier commitments and warmer colors denoting later commitments; tokens with the same color are committed in parallel. Unlike Vanilla token-level decoding, CLAD commits contiguous high-confidence spans as cluster-level update units, reaching the same correct answer with lower latency.

This commitment decision largely determines how much parallelism an MDLM decoder can exploit without excessive quality degradation (Feng et al., 2025; Kang et al., 2025). A decoder that commits more reliable predictions per step can reduce the number of denoising iterations and improve throughput. However, overly aggressive commitment may accept predictions that are reliable in isolation yet incompatible when committed together. Existing training-free samplers mostly make this decision at token-level granularity. Confidence- and uncertainty-based methods rank individual masked positions by maximum probability, entropy, margin, or distributional stability (Wu et al., 2025; Kim et al., 2025a; Ben-Hamu et al., 2025; Kim et al., 2025b), while dependency-aware methods estimate token-level interactions from self-attention to avoid committing strongly coupled positions (Kim et al., 2026; Luo et al.,

2026). Despite these advances, the basic scheduling unit remains an individual token position.

We revisit this decoding granularity. Although MDLMs expose token-wise predictive distributions, token positions need not be the only useful unit of decoding progress. This view is consistent with recent efforts that decouple global semantic organization from local textual realization in language diffusion models (Guo et al., 2026), as well as observations that local neighborhoods around high-confidence predictions can become reliable during MDLM decoding (Kong et al., 2025; Du et al., 2026). This leads to the central question of this work: *Can an MDLM decoder commit contiguous high-confidence spans as update units while avoiding strongly dependent spans in the same denoising step?*

The first part of this question concerns the unit of progress. We observe that high-confidence candidates often emerge as contiguous spans rather than isolated positions. Once such a span becomes well supported by the current partially unmasked context, committing its tokens one by one can spend unnecessary refinement steps on a local region that is already predictable. We therefore define a *confidence-induced cluster* (CIC) as a maximal contiguous span of high-confidence masked positions. A CIC serves as an operational notion of a locally reliable span: its positions are adjacent, individually confident, and can be treated as a candidate update unit.

The second part concerns compatibility. Cluster-level commitment should not simply decode all confident spans at once. Even if each CIC is locally reliable, different CICs may remain mutually dependent under the current context. Committing such clusters simultaneously may accept predictions that are confident marginally but inconsistent jointly. We extend this attention-based dependency signal from individual positions to CICs, using self-attention maps from the same forward pass to estimate inter-cluster dependencies and model them as *dependency conflicts*. This yields a simple cluster-level view of MDLM decoding: CICs define the units of progress, while attention-derived conflicts determine which units are compatible for same-step commitment.

Based on this view, we propose CLAD (**C**luster-**L**evel **A**ttention-Guided **D**ecoding), a training-free cluster-level decoder for MDLMs. At each denoising step, CLAD converts token-level confidence estimates into CIC-level update candidates, constructs a sparse conflict graph over CICs using attention-derived inter-cluster dependencies, and selects a maximum-weight set of non-conflicting CICs for parallel commitment. By keeping only mutually strongest inter-cluster dependencies, CLAD obtains a compact conflict graph and performs cluster selection efficiently.

We evaluate CLAD on mathematical reasoning and code-generation benchmarks using pretrained MDLMs from the LLaDA and Dream families (Nie et al., 2025; Zhu et al., 2025; Ye et al., 2025). Across four benchmarks, CLAD achieves $1.77\times$–$8.47\times$ speedups over Vanilla decoding while maintaining broadly comparable task accuracy in most settings. Compared with token-level dependency-aware baselines such as DAPD (Kim et al., 2026) and DAWN (Luo et al., 2026), CLAD improves throughput in most evaluated settings. These results suggest that changing the decoding unit from tokens to confidence-induced spans is an effective way to improve throughput, while attention-derived compatibility modeling helps preserve accuracy under more aggressive parallel commitment.

Our contributions are summarized as follows:

- We revisit the decoding granularity of MDLMs and propose confidence-induced clusters (CICs), maximal contiguous spans of high-confidence masked positions, as explicit span-level update units.
- We propose CLAD (**C**luster-**L**evel **A**ttention-Guided **D**ecoding), a training-free cluster-level decoder that builds attention-derived inter-cluster conflict graphs and selects compatible CICs for parallel commitment.
- Experiments on four representative reasoning and code-generation benchmarks across the LLaDA and Dream model families show that CLAD achieves $1.77\times$–$8.47\times$ speedups over Vanilla decoding while maintaining broadly comparable task accuracy in most settings.

2 Related Work

Discrete Diffusion Language Models. The pioneering work D3PM (Austin et al., 2021a) establishes the paradigm for early discrete diffusion models, which typically define a corruption process over categorical variables through transition matrices. Subsequent work further generalizes this line of research with continuous-time Markov chain formulations (Campbell et al., 2022) and

score-based discrete diffusion models (Meng et al., 2022). More recently, masked diffusion language models have attracted increasing attention due to their simple absorbing-state corruption process and strong empirical performance (Sahoo et al., 2024; Shi et al., 2024; Ou et al., 2025). The scaling of masked diffusion language models to large language model regimes has been actively pursued. LLaDA (Nie et al., 2025) demonstrates competitive performance on reasoning, instruction following, and code generation tasks. Dream (Ye et al., 2025) further explores diffusion language modeling at scale by leveraging autoregressive initialization.

Decoding Strategies for Masked Diffusion Language Models. Existing training-free samplers for MDLMs mostly make the commitment decision at token-level granularity. Confidence-based decoding ranks individual masked positions by the maximum probability of their predicted token distributions (Chang et al., 2022; Nie et al., 2025). Other uncertainty-aware criteria have also been explored, including entropy-based selection (Koh et al., 2024) and margin-based confidence (Kim et al., 2025a). To improve inference efficiency, recent methods commit multiple token positions in parallel while attempting to control quality degradation. Fast-dLLM (Wu et al., 2025) accelerates inference by combining confidence-threshold parallel decoding with cache optimization. EB-Sampler (Ben-Hamu et al., 2025) selects tokens under an entropy-bounded criterion, while KLASS (Kim et al., 2025b) uses distributional stability across denoising steps to identify reliable tokens. Dependency-aware methods further refine this token-level commitment decision with self-attention (Vaswani et al., 2017) signals. DOS (Zhou et al., 2026a) and Attn-Sampler (Zhou et al., 2026b) prioritize masked tokens according to their dependence on the current unmasked context, while DAWN (Luo et al., 2026) and DAPD (Kim et al., 2026) use attention-derived token interactions to avoid simultaneously decoding strongly coupled masked positions.

A related line of work exploits local structure beyond isolated token positions. R^2-dLLM (Du et al., 2026) reduces decoding redundancy via local confidence aggregation and stability checks, while LocalLeap (Kong et al., 2025) relaxes parallel decoding around locally deterministic high-confidence anchors.

3 Preliminaries

Masked Diffusion Language Models. Masked diffusion language models formulate text generation as the reverse of an absorbing-state masking process. Let $\mathbf{x}_0 = (x_0^1, \ldots, x_0^L) \in \mathcal{V}^L$ denote a clean token sequence, where $\mathcal{V}$ is the vocabulary and L is the sequence length. The vocabulary is augmented with a special mask token $m = [\texttt{MASK}]$.

The forward corruption process is governed by a monotonically decreasing schedule α_t, with $\alpha_0 \approx 1$ and $\alpha_1 \approx 0$. At timestep $t \in [0, 1]$, each token is independently preserved with probability α_t or replaced by the absorbing mask state:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \prod_{i=1}^{L} \mathrm{Cat}\Big(x_t^i;\ \alpha_t \delta_{x_0^i} + (1-\alpha_t)\delta_m\Big). \tag{1}$$

As t increases, α_t decreases and the sequence is progressively corrupted toward the fully masked state.

The generative process reverses this corruption by iteratively unmasking tokens. Given a denoising model f_θ, the reverse transition from timestep t to an earlier timestep s is parameterized as

$$p_\theta(x_s^i \mid \mathbf{x}_t) = \begin{cases} \mathrm{Cat}\Big(x_s^i; \delta_{x_t^i}\Big), & x_t^i \neq m, \\ \mathrm{Cat}\Big(x_s^i;\ \frac{1-\alpha_s}{1-\alpha_t}\delta_m + \frac{\alpha_s - \alpha_t}{1-\alpha_t} f_\theta^i(\mathbf{x}_t)\Big), & x_t^i = m. \end{cases} \tag{2}$$

Here $f_\theta^i(\mathbf{x}_t)$ denotes the predicted distribution over clean vocabulary tokens at position i. Therefore, already visible tokens are deterministically carried over, while masked tokens are gradually converted into clean tokens according to the model prediction.

The denoising model is trained by predicting the original tokens at masked positions, yielding the following training objective:

$$\mathcal{L}(\theta) = \mathrm{E}_{\mathbf{x}_0, t, \mathbf{x}_t}\left[\frac{\alpha_t'}{1-\alpha_t} \sum_{i: x_t^i = m} \log f_\theta^i(x_0^i \mid \mathbf{x}_t)\right]. \tag{3}$$

After training, generation starts from a masked sequence and repeatedly applies the reverse process. Since all masked positions can be predicted simultaneously at each denoising step, the decoding strategy that selects which positions to commit becomes a key factor for both generation accuracy and inference efficiency.

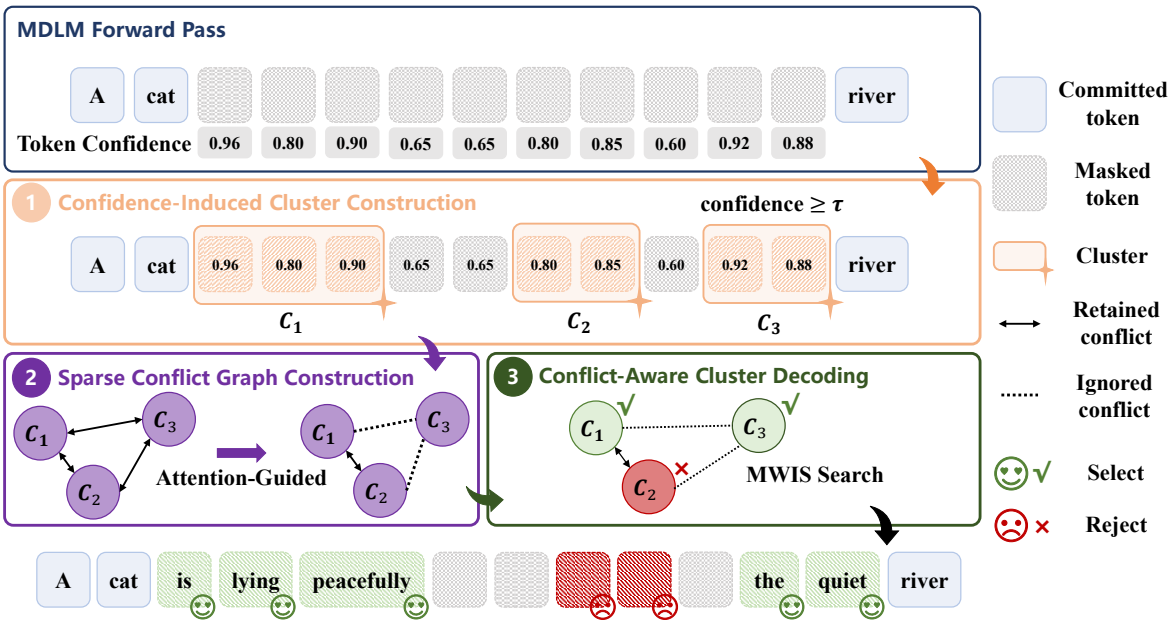


Figure 2: Overview of CLAD. The framework contains three main components: Confidence-Induced Cluster Construction, Sparse Conflict Graph Construction, and Conflict-Aware Cluster Decoding.

4 Methodology

4.1 Overview

The introduction motivates a cluster-level view of MDLM decoding: the decoder can improve throughput by committing locally reliable spans rather than isolated token positions, but it must avoid committing span-level units that remain strongly dependent in the same denoising step. We now instantiate this view as CLAD, a training-free parallel decoder that operates directly on the model's current predictions and attention maps.

Figure 2 summarizes the overall procedure. At each denoising step, CLAD uses the model's token-level confidence estimates to construct *confidence-induced clusters* (CICs), i.e., maximal contiguous spans of high-confidence masked positions. These CICs serve as span-level update candidates and are the main source of acceleration: committing a selected CIC reveals multiple adjacent tokens in one step rather than spending separate refinement steps on each token.

However, faster commitment should not mean committing every confident span. Different CICs may still depend on each other under the current partially unmasked context, so committing them simultaneously can introduce incompatible updates. We refer to this incompatibility as a *dependency conflict*. To mitigate such conflicts, CLAD reuses self-attention maps from the same forward pass to estimate inter-cluster dependencies and constructs a sparse conflict graph. Nodes in this graph are CICs, and edges indicate pairs of CICs that should not be committed together. The final decoding step selects a set of non-conflicting CICs and commits their greedy predictions in parallel, thereby increasing the number of tokens committed per step while avoiding cluster pairs estimated to be strongly dependent.

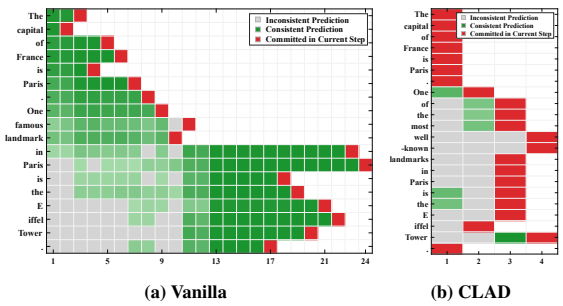


Figure 3: Detailed decoding traces for the two decoding processes illustrated in Figure 1. (a) shows the Vanilla process, while (b) shows the CLAD process. Green, gray, and red cells denote consistent predictions, inconsistent predictions, and tokens committed at the current denoising step, respectively.

Figure 3 provides a detailed trace of this behavior. The example illustrates the primary acceleration mechanism of CLAD: once several adjacent

positions become reliable together, CLAD can commit them as a single cluster-level update. Conflict-aware selection is applied when multiple candidate CICs are present, deciding which span-level updates can be safely committed in the same step. The following subsections formalize the three components of CLAD: confidence-induced cluster construction, sparse conflict graph construction, and conflict-aware cluster decoding.

4.2 Confidence-Induced Cluster Construction

At denoising step t, let $\mathbf{x}_t$ denote the current sequence state and $\mathcal{M}_t = \{i : x_t^i = m\}$ denote the set of masked positions. For each masked position, the model produces a predictive distribution. We define the greedy prediction and confidence as

$$\hat{x}_0^i = \arg\max_{v \in \mathcal{V}} p_\theta(x_0^i = v \mid \mathbf{x}_t), \tag{4}$$

$$c^i = p_\theta(x_0^i = \hat{x}_0^i \mid \mathbf{x}_t). \tag{5}$$

A position is regarded as a candidate if its confidence is above a threshold τ:

$$\mathcal{P}_t = \{i \in \mathcal{M}_t : c^i \geq \tau\}. \tag{6}$$

The candidate set is then decomposed into maximal contiguous spans. We denote the resulting span set as

$$\mathcal{C}_t = \{C_1, \ldots, C_K\}, \quad C_k = \{l_k, l_k + 1, \ldots, r_k\}. \tag{7}$$

Each C_k is a *confidence-induced cluster (CIC)*. By construction, all positions inside C_k are high-confidence candidates, and no adjacent candidate outside $[l_k, r_k]$ can be added to the cluster.

4.3 Sparse Conflict Graph Construction

CICs provide span-level candidates for faster commitment, but confidence alone does not determine whether different CICs can be safely committed together. Strongly dependent CICs may produce incompatible same-step updates, so we treat salient inter-cluster dependency as a *dependency conflict*. To identify these conflicts, we estimate inter-cluster dependency using the attention map produced by the same forward pass. The resulting conflict graph connects pairs of CICs that should not be committed in the same denoising step.

Let $A_t \in \mathbb{R}^{L \times L}$ be the head-averaged attention matrix at step t. Since the graph is used to detect conflicts rather than directional generation dependencies, we symmetrize the attention scores:

$$R_t^{ij} = \max\left(A_t^{ij}, A_t^{ji}\right), \quad R_t^{ii} = 0. \tag{8}$$

For two clusters C_a and C_b, their dependency score is computed by aggregating token-level attention into a cluster-level interaction:

$$s_{ab} = \max\left\{\max_{i \in C_a} \frac{1}{|C_b|} \sum_{j \in C_b} R_t^{ij},\ \max_{j \in C_b} \frac{1}{|C_a|} \sum_{i \in C_a} R_t^{ji}\right\}. \tag{9}$$

This score measures the strongest cross-cluster dependency in either direction. The averaging operation avoids favoring longer target clusters, while the maximum operation captures the most influential token-level interaction between two clusters.

We then construct a sparse conflict graph $G_t = (\mathcal{C}_t, \mathcal{E}_t)$, where the node set $\mathcal{C}_t$ consists of all CICs at step t, and the edge set $\mathcal{E}_t$ records inter-cluster conflicts. Instead of connecting all pairs with large attention scores, we retain only reciprocal salient dependencies. For each cluster C_a, let

$$b^\star(a) = \arg\max_{b \neq a} s_{ab} \tag{10}$$

be its strongest external dependency. An edge is added between C_a and C_b when they are mutually strongest dependencies and their interaction is above the average dependency level of both clusters:

$$(a, b) \in \mathcal{E}_t \iff \begin{cases} b = b^\star(a), & a = b^\star(b), \\ s_{ab} \geq \bar{s}_a, & s_{ab} \geq \bar{s}_b, \end{cases} \tag{11}$$

where

$$\bar{s}_a = \frac{1}{K-1} \sum_{b \neq a} s_{ab}, \quad K \neq 1. \tag{12}$$

When $K = 1$, we skip graph construction and directly select the only CIC. This mutual-top criterion retains only salient reciprocal conflicts. The resulting sparse graph directly indicates which CICs should not be committed together, without introducing an additional conflict-threshold hyperparameter. Edges are selected adaptively based on reciprocal salience and the cluster-specific average scores $\bar{s}_a$ and $\bar{s}_b$.

4.4 Conflict-Aware Cluster Decoding

Given the conflict graph, the goal is to commit as many reliable tokens as possible while avoiding conflicting clusters. We formulate this as a maximum-weight independent set problem (MWIS) (Nemhauser and Trotter Jr, 1975):

$$\begin{aligned} \mathcal{S}_t = \arg\max_{\mathcal{S} \subseteq \mathcal{C}_t} & \sum_{C_a \in \mathcal{S}} |C_a|, \\ \text{s.t. } & (C_a, C_b) \notin \mathcal{E}_t, \quad \forall C_a, C_b \in \mathcal{S}. \end{aligned} \tag{13}$$

Algorithm 1 CLAD: Cluster-Level Attention-Guided Decoding

Require: Current state $\mathbf{x}_t$, model p_θ, confidence threshold τ
Ensure: Updated state $\mathbf{x}_{t-1}$

$\mathcal{M}_t \leftarrow \{i : x_t^i = m\}$
if $\mathcal{M}_t = \emptyset$ **then**
  return $\mathbf{x}_t$
end if
$(\hat{\mathbf{x}}_0, \mathbf{c}, A_t) \leftarrow \mathrm{Forward}(p_\theta, \mathbf{x}_t)$
$\mathcal{P}_t \leftarrow \{i \in \mathcal{M}_t : c^i \geq \tau\}$
if $\mathcal{P}_t = \emptyset$ **then**
  $j^\star \leftarrow \arg\max_{i \in \mathcal{M}_t} c^i$
  $\mathbf{x}_{t-1} \leftarrow \mathbf{x}_t$
  $x_{t-1}^{j^\star} \leftarrow \hat{x}_0^{j^\star}$
  return $\mathbf{x}_{t-1}$
end if
$\mathcal{C}_t \leftarrow \mathrm{CIC\ Construction}(\mathcal{P}_t)$
$G_t = (\mathcal{C}_t, \mathcal{E}_t) \leftarrow \mathrm{Sparse\ Graph}(\mathcal{C}_t, A_t)$
$\mathcal{S}_t \leftarrow \mathrm{MWIS}(G_t, \omega(C_a) = |C_a|)$
$\mathcal{U}_t \leftarrow \bigcup_{C_a \in \mathcal{S}_t} C_a$
$\mathbf{x}_{t-1} \leftarrow \mathbf{x}_t$
$x_{t-1}^i \leftarrow \hat{x}_0^i, \quad \forall i \in \mathcal{U}_t$
return $\mathbf{x}_{t-1}$

Although MWIS is generally hard on arbitrary graphs, our mutual-top construction makes the resulting conflict graph a matching, where each CIC participates in at most one retained conflict edge. Therefore, Eq. (13) can be solved exactly rather than greedily: we select all isolated CICs and, for each retained conflict edge, keep the CIC that contains more token positions, i.e., the higher-weight CIC. This selection costs $O(K + |\mathcal{E}_t|)$ after graph construction, where $K = |\mathcal{C}_t|$. The selected CICs are therefore mutually non-conflicting, and the final update set is the union of the selected clusters:

$$\mathcal{U}_t = \bigcup_{C_a \in \mathcal{S}_t} C_a. \tag{14}$$

The sequence is updated by committing the greedy prediction at every selected position:

$$x_{t-1}^i = \begin{cases} \hat{x}_0^i, & i \in \mathcal{U}_t, \\ x_t^i, & i \notin \mathcal{U}_t. \end{cases} \tag{15}$$

If no CIC is formed because no masked position passes the confidence threshold, we fall back to committing the single masked position with the highest confidence, ensuring that the denoising process always makes progress.

The above components together define the decoding procedure of CLAD. Algorithm 1 summarizes the overall procedure.

5 Experiments

5.1 Experimental Setup

Models. We evaluate CLAD on four open-source masked diffusion language models, including LLaDA-8B-Instruct (Nie et al., 2025), LLaDA-1.5 (Zhu et al., 2025), Dream-v0-Base-7B, and Dream-v0-Instruct-7B (Ye et al., 2025). We keep model weights fixed and only modify the decoding strategy, so all compared methods are evaluated in a training-free setting.

Datasets. We evaluate on benchmarks covering mathematical reasoning and code generation. For mathematical reasoning, we use GSM8K (5-shot) (Cobbe et al., 2021) and MATH (4-shot) (Hendrycks et al., 2021), which require multi-step reasoning and longer-form answer generation. For code generation, we use MBPP (3-shot) (Austin et al., 2021b) and HumanEval (0-shot) (Chen et al., 2021), which require generating executable Python solutions from natural language specifications.

Baselines. We compare CLAD with the following training-free decoding methods. **Vanilla** denotes the original top-1 decoding strategy of MDLMs. **Fast-dLLM** (Wu et al., 2025) dynamically commits positions whose confidence exceeds a predefined threshold. **KLASS** (Kim et al., 2025b) selects tokens using both confidence and token-level distributional stability across denoising steps. **DAPD** (Kim et al., 2026) and **DAWN** (Luo et al., 2026) use attention-derived token-level dependency graphs to guide dependency-aware parallel commitment. These baselines represent both uncertainty-based and dependency-aware training-free decoding strategies.

Evaluation Metrics. We report accuracy and tokens per second (TPS). For code generation, accuracy is measured by the pass rate on the official unit tests. For mathematical reasoning, accuracy is computed by matching the extracted final answer against the ground-truth answer. TPS is measured end-to-end over the evaluation generation loop, including prompt preprocessing, decoding, and output post-processing.

Implementation Details. Unless otherwise specified, we use the same generation length 256 and block length 32 for all compared methods under each benchmark. Additional implementation details are provided in Appendix A.

Table 1: Main results on reasoning and code generation benchmarks across LLaDA-8B-Instruct and Dream-v0-Instruct-7B. Acc. denotes task accuracy, TPS denotes tokens per second, and Speedup is measured relative to Vanilla top-1 decoding under the same setting. Colored subscripts on Acc. report the absolute change relative to Vanilla. Higher values are better for all metrics, and the best TPS and Speedup are highlighted in bold.

Task	Method	LLaDA-8B-Instruct			Dream-v0-Instruct-7B		
		Acc.↑	TPS↑	Speedup↑	Acc.↑	TPS↑	Speedup↑
GSM8K (5-shot)	Vanilla	77.71	5.22	1.00×	76.49	3.30	1.00×
	Fast-dLLM	$78.32_{\uparrow 0.61}$	15.92	3.05×	$75.89_{\downarrow 0.60}$	13.27	4.02×
	KLASS	$75.59_{\downarrow 2.12}$	12.40	2.38×	$73.46_{\downarrow 3.03}$	5.40	1.64×
	DAPD	$74.07_{\downarrow 3.64}$	14.64	2.80×	$70.51_{\downarrow 5.98}$	7.95	2.41×
	DAWN	$76.88_{\downarrow 0.83}$	21.94	4.20×	$74.37_{\downarrow 2.12}$	**15.64**	**4.74×**
	CLAD (ours)	$77.79_{\uparrow 0.08}$	**25.59**	**4.90×**	$73.69_{\downarrow 2.80}$	14.54	4.41×
MATH (4-shot)	Vanilla	32.84	8.25	1.00×	38.36	7.88	1.00×
	Fast-dLLM	$32.80_{\downarrow 0.04}$	21.82	2.64×	$38.12_{\downarrow 0.24}$	16.95	2.15×
	KLASS	$31.96_{\downarrow 0.88}$	15.62	1.89×	$34.78_{\downarrow 3.58}$	7.72	0.98×
	DAPD	$31.44_{\downarrow 1.40}$	21.08	2.56×	$32.00_{\downarrow 6.36}$	13.11	1.66×
	DAWN	$32.36_{\downarrow 0.48}$	28.76	3.49×	$37.98_{\downarrow 0.38}$	19.69	2.50×
	CLAD (ours)	$31.92_{\downarrow 0.92}$	**31.01**	**3.76×**	$37.34_{\downarrow 1.02}$	**20.44**	**2.59×**
MBPP (3-shot)	Vanilla	31.60	5.45	1.00×	54.80	3.74	1.00×
	Fast-dLLM	$31.80_{\uparrow 0.20}$	20.01	3.67×	$56.00_{\uparrow 1.20}$	17.71	4.74×
	KLASS	$31.96_{\uparrow 0.36}$	15.62	2.87×	$56.20_{\uparrow 1.40}$	9.44	2.52×
	DAPD	$32.00_{\uparrow 0.40}$	8.08	1.48×	$54.40_{\downarrow 0.40}$	5.35	1.43×
	DAWN	$30.60_{\downarrow 1.00}$	26.53	4.87×	$56.00_{\uparrow 1.20}$	20.15	5.39×
	CLAD (ours)	$30.20_{\downarrow 1.40}$	**26.66**	**4.89×**	$54.00_{\downarrow 0.80}$	**20.64**	**5.52×**
HumanEval (0-shot)	Vanilla	40.24	16.66	1.00×	52.44	11.07	1.00×
	Fast-dLLM	$41.46_{\uparrow 1.22}$	55.11	3.31×	$56.71_{\uparrow 4.27}$	26.80	2.42×
	KLASS	$40.24_{0.00}$	31.81	1.91×	$54.27_{\uparrow 1.83}$	12.80	1.16×
	DAPD	$39.63_{\downarrow 0.61}$	32.51	1.95×	$51.83_{\downarrow 0.61}$	12.28	1.11×
	DAWN	$40.24_{0.00}$	67.99	4.08×	$54.27_{\uparrow 1.83}$	31.37	2.83×
	CLAD (ours)	$40.24_{0.00}$	**70.65**	**4.24×**	$54.27_{\uparrow 1.83}$	**31.62**	**2.86×**

5.2 Main Results

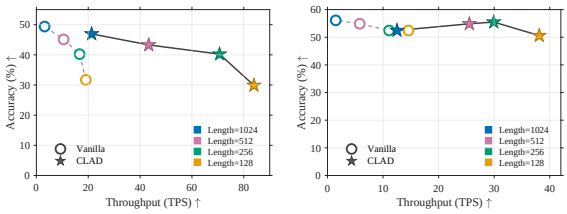
Table 1 reports the accuracy and decoding speed of different methods across two masked diffusion language models and four benchmarks. Overall, CLAD substantially improves decoding throughput in most settings while maintaining broadly comparable task accuracy. On LLaDA-8B-Instruct, CLAD achieves the highest TPS on all four benchmarks, obtaining 4.90×, 3.76×, 4.89×, and 4.24× speedup on GSM8K, MATH, MBPP, and HumanEval, respectively. On Dream-v0-Instruct-7B, CLAD also achieves the best throughput on three out of four benchmarks, including MATH, MBPP, and HumanEval. At the same time, we acknowledge that CLAD still shows modest accuracy drops in a few settings, particularly on GSM8K with Dream-v0-Instruct-7B (−2.80 points) and MBPP with LLaDA-8B-Instruct (−1.40 points). Additional results on LLaDA-1.5 and Dream-v0-Base-7B are in Appendix C.

Compared with DAWN and DAPD, which perform dependency-aware scheduling at the token level, CLAD achieves higher throughput in most settings. This supports the central motivation of CLAD: using CICs as scheduling units can increase effective parallelism by committing multiple adjacent high-confidence candidates in a single update. Meanwhile, the broadly comparable accuracy suggests that conflict-aware selection helps maintain generation quality under this more aggressive cluster-level commitment strategy. Overall, CLAD provides a more attractive alternative to other token-level parallel decoding methods.

5.3 Ablation Study

Impact of generation length. We study the effect of generation length on HumanEval (0-shot) by varying the generation budget in $\{128, 256, 512, 1024\}$. As shown in Figure 4, CLAD achieves higher throughput than Vanilla decoding across all lengths while maintaining broadly

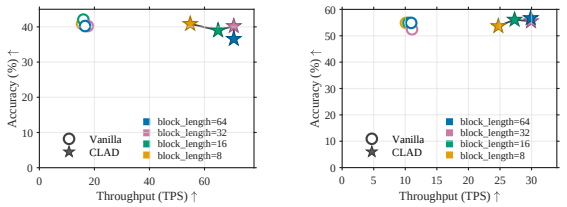
comparable accuracy, with the largest gains at medium lengths. This indicates that cluster-level commitment remains effective under different generation budgets.



(a) LLaDA-8B-Instruct (b) Dream-v0-Instruct-7B

Figure 4: Ablation study on generation length using HumanEval (0-shot). Circles denote Vanilla decoding, and stars denote CLAD; colors indicate different generation lengths. The x-axis reports throughput in tokens per second (TPS), and the y-axis reports accuracy.

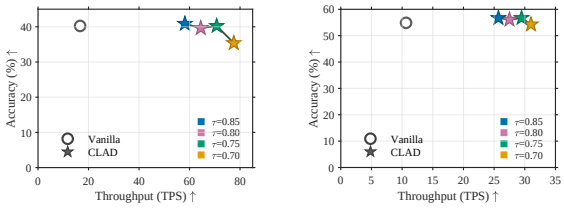
Impact of block size. We next examine the effect of block size with the generation length fixed to 256. Figure 5 shows that CLAD achieves higher throughput under all block sizes on both models. For LLaDA, CLAD achieves its highest speedup at a block size of 32, while increasing the block size further preserves high throughput with a slight accuracy decrease. On Dream, CLAD remains robust across block sizes and improves both throughput and accuracy in most settings.



(a) LLaDA-8B-Instruct (b) Dream-v0-Instruct-7B

Figure 5: Ablation study on block size using HumanEval (0-shot). Circles denote Vanilla decoding, and stars denote CLAD; colors indicate different block sizes. The x-axis reports throughput in tokens per second (TPS), and the y-axis reports accuracy.

Impact of confidence threshold τ. We also analyze the confidence threshold τ, which controls the candidate positions used to form CICs. As shown in Figure 6, smaller thresholds admit more candidates and therefore improve throughput, but may also introduce less reliable updates. Larger thresholds make candidate selection more selective, which tends to stabilize accuracy at the cost of reduced speed. Across both models, moderate thresholds generally provide a practical balance between throughput and accuracy.



(a) LLaDA-8B-Instruct (b) Dream-v0-Instruct-7B

Figure 6: Ablation study on the confidence threshold τ using HumanEval (0-shot). Circles denote Vanilla decoding, and stars denote CLAD with different thresholds. Colors indicate different values of τ. The x-axis reports throughput in tokens per second (TPS), and the y-axis reports accuracy.

Table 2: Component ablation on HumanEval (0-shot) with LLaDA-8B-Instruct. Graph denotes the conflict graph.

Variant	CIC	Graph	MWIS	Acc.	TPS
Vanilla	×	×	×	40.24	16.66
+CIC	✓	×	×	36.58	76.03
+Graph	✓	✓	×	36.58	66.13
+MWIS	✓	✓	✓	40.24	70.65

Component ablation. Table 2 shows that CIC alone brings high speed but degrades accuracy. Randomly selecting clusters after constructing the conflict graph still fails to recover accuracy. With MWIS, CLAD matches Vanilla accuracy while reaching 70.65 TPS, validating the necessity of conflict graph-based cluster selection.

6 Conclusion

We proposed CLAD, a training-free cluster-level parallel decoder for MDLMs. The key idea is to move beyond token-level commitment by grouping adjacent high-confidence candidates into confidence-induced clusters and using them as span-level update units. To control the risk of aggressive cluster-level commitment, CLAD estimates inter-cluster dependencies from self-attention and selects a set of non-conflicting CICs for parallel decoding. Experiments on LLaDA and Dream model families show that CLAD substantially improves decoding throughput while maintaining broadly comparable task accuracy in most settings.

Limitations

CLAD uses self-attention as a lightweight proxy for estimating inter-cluster dependencies when constructing the conflict graph. This choice is straightforward, efficient, and intuitive: attention maps are already available from the same forward pass, require no additional training, and provide a useful signal for identifying clusters that may interact during same-step commitment. Empirically, this signal works well for guiding conflict-aware selection. However, attention is still an indirect heuristic rather than a theoretically grounded quantitative measure of causal dependence. A large attention score does not necessarily imply that two clusters will lead to incompatible updates, and a small score does not fully rule out latent semantic or logical dependencies. Therefore, the conflict graph should be viewed as an efficient approximation for reducing risky parallel commitments, rather than a formal guarantee of cluster independence.

The benefit of CLAD may also vary across models and tasks. Cluster-level decoding is most effective when the current denoising state contains locally coherent spans that can be committed together with limited inter-cluster incompatibility. When candidate updates are highly fragmented, when dependencies among clusters are dense, or when a task requires more fine-grained step-by-step refinement, the available parallelism can be smaller. Different MDLM backbones may also exhibit different attention patterns and decoding dynamics, which can affect how reliably CICs serve as update units. Thus, CLAD should not be interpreted as universally optimal for every setting; rather, it provides a practical cluster-level decoding strategy whose effectiveness depends on how well the model and task support reliable span-level commitment.

References

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

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

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

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Heli Ben-Hamu, Itai Gat, Daniel Severo, Niklas S Nolte, and Brian Karrer. 2025. Accelerated sampling from masked diffusion models via entropy bounded unmasking. *Advances in Neural Information Processing Systems*, 38:55981–56007.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.

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

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

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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Zhenbang Du, Kejing Xia, Xinrui Zhong, Yonggan Fu, Nicolai Oswald, Binfei Ji, Brucek Khailany, Pavlo Molchanov, and Yingyan Lin. 2026. R^2-dllm: Accelerating diffusion large language models via spatio-temporal redundancy reduction. *arXiv preprint arXiv:2604.18995*.

Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. 2025. Theoretical benefit and limitation of diffusion language model. *Advances in Neural Information Processing Systems*, 38:24415–24459.

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

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

Hongcan Guo, Qinyu Zhao, Yian Zhao, Shen Nie, Rui Zhu, Qiushan Guo, Feng Wang, Tao Yang, Hengshuang Zhao, Guoqiang Wei, and Yan Zeng. 2026. Continuous latent diffusion language model. *arXiv preprint arXiv:2605.06548*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. 2025. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*.

Bumjun Kim, Dongjae Jeon, Moongyu Jeon, and Albert No. 2026. Dependency-aware parallel decoding via attention for diffusion llms. *arXiv preprint arXiv:2603.12996*.

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

Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. 2025b. Klass: Kl-guided fast inference in masked diffusion models. *Advances in Neural Information Processing Systems*, 38:92267–92301.

Hyukhun Koh, Minha Jhang, Dohyung Kim, Sangmook Lee, and Kyomin Jung. 2024. Plm-based discrete diffusion language models with entropy-adaptive gibbs sampling. *arXiv preprint arXiv:2411.06438v2*.

Fanheng Kong, Jingyuan Zhang, Yahui Liu, Zirui Wu, Yu Tian, Victoria W., and Guorui Zhou. 2025. Accelerating diffusion llm inference via local determinism propagation. *arXiv preprint arXiv:2510.07081*.

Lizhuo Luo, Zhuoran Shi, Jiajun Luo, Zhi Wang, Shen Ren, Wenya Wang, and Tianwei Zhang. 2026. Dawn: Dependency-aware fast inference for diffusion llms. *arXiv preprint arXiv:2602.06953*.

Chenlin Meng, Kristy Choi, Jiaming Song, and Stefano Ermon. 2022. Concrete score matching: Generalized score matching for discrete data. *Advances in Neural Information Processing Systems*, 35:34532–34545.

George L Nemhauser and Leslie E Trotter Jr. 1975. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025. Large language diffusion models. *Advances in Neural Information Processing Systems*, 38:50608–50646.

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

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

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

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. 2024. Simplified and generalized masked diffusion for discrete data. *Advances in Neural Information Processing Systems*, 37:103131–103167.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

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. *arXiv preprint arXiv:2505.22618*.

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*.

Xueyu Zhou, Yangrong Hu, and Jian Huang. 2026a. Dos: Dependency-oriented sampler for masked diffusion language models. *arXiv preprint arXiv:2603.15340*.

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

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

A Additional Implementation Details

Experiments on LLaDA-8B-Instruct, LLaDA-1.5, and Dream-v0-Base-7B are conducted on a single NVIDIA RTX 4090 GPU, while experiments on Dream-v0-Instruct-7B are conducted on a single NVIDIA L40 GPU. For a fair comparison, the attention signal used by DAPD, DAWN, and CLAD is computed in the same way, by averaging attention maps over all heads from the last four Transformer layers.

B Parameter Configurations

For compactness, we use LLaDA-Inst to denote LLaDA-8B-Instruct, Dream-Base to denote Dream-v0-Base-7B, and Dream-Inst to denote Dream-v0-Instruct-7B. We use the public implementations of Fast-dLLM, KLASS, and DAWN with their default or author-recommended hyperparameters. Since DAPD does not provide an official open-source implementation, we re-implement it following the original paper. For both DAPD and CLAD, we perform a small preliminary sweep over their method-specific thresholds on a limited subset of examples, and use the selected configurations for the full evaluation. The sweep is guided by how much parallelism an MDLM decoder can exploit without excessive quality degradation. All selected hyperparameter settings are provided below.

(a) Fast-dLLM. Fast-dLLM (Wu et al., 2025) is an uncertainty-based parallel decoding method. Its main hyperparameter is `threshold`. In our experiments, Fast-dLLM uses a fixed confidence threshold of 0.9 for all evaluated models and tasks.

Table 3: KLASS hyperparameters. The two LLaDA models use the same settings, and the two Dream models use the same settings. HE denotes HumanEval.

Model	Param.	GSM8K	MATH	HE	MBPP
LLaDA models	τ_{conf}	0.6	0.6	0.9	0.7
	τ_{KL}	0.015	0.01	0.01	0.01
Dream models	τ_{conf}	0.9	0.9	0.8	0.9
	τ_{KL}	0.001	0.005	0.001	0.001

(b) KLASS. KLASS (Kim et al., 2025b) selects candidate tokens using both prediction confidence and distributional stability across denoising steps. We denote its confidence threshold by τ_{conf} and its KL-stability threshold by τ_{KL}. The confidence threshold filters out low-confidence predictions, while the KL-stability threshold requires the predicted token distribution to remain stable between consecutive denoising steps; a smaller τ_{KL} imposes a stricter stability requirement. Table 3 reports the task-specific values of τ_{conf} and τ_{KL}.

(c) DAPD. DAPD (Kim et al., 2026) uses attention-based dependency estimates to avoid committing strongly dependent tokens in the same denoising step. Its main hyperparameters are $\tau_{\min}$ and $\tau_{\max}$, which control the lower and upper bounds of the dependency criterion. Other settings are fixed across experiments: the switching ratio is 0.5, the fast-stage confidence threshold is 0.9, and single-block decoding is disabled. Table 4 reports the task-specific values of $\tau_{\min}$ and $\tau_{\max}$.

Table 4: DAPD hyperparameters. HE denotes HumanEval.

Model	Param.	GSM8K / MATH	HE / MBPP
LLaDA-1.5	$\tau_{\min}$	0.0005	0.0005
	$\tau_{\max}$	0.0020	0.0010
LLaDA-Inst	$\tau_{\min}$	0.0010	0.0005
	$\tau_{\max}$	0.0040	0.0012
Dream-Base	$\tau_{\min}$	0.0005	0.0004
	$\tau_{\max}$	0.0020	0.0008
Dream-Inst	$\tau_{\min}$	0.0010	0.0005
	$\tau_{\max}$	0.0040	0.0010

(d) DAWN. DAWN (Luo et al., 2026) constructs a token-level dependency graph from attention maps and uses this graph to guide parallel commitment. Its hyperparameters control different parts of the graph-based decoder. The threshold τ_{sink} filters attention-sink positions, τ_{edge} controls whether an attention link is retained as a dependency edge, τ_{induce} controls the confidence requirement for anchor-induced candidate tokens, and τ_{conf} controls which remaining candidate tokens are considered for conflict-based scheduling. Table 5 and Table 6 report the fixed graph-construction hyperparameters and the task-specific τ_{conf}, respectively.

Table 5: DAWN fixed graph-construction hyperparameters.

Model	τ_{sink}	τ_{induce}	τ_{edge}
LLaDA-1.5	0.01	0.70	0.07
LLaDA-Inst	0.01	0.70	0.07
Dream-Base	0.03	0.75	0.05
Dream-Inst	0.03	0.75	0.10

Table 6: DAWN task-specific τ_{conf}. HE denotes HumanEval.

Model	GSM8K	MATH	HE	MBPP
LLaDA-1.5	0.75	0.75	0.80	0.75
LLaDA-Inst	0.75	0.75	0.80	0.70
Dream-Base	0.75	0.80	0.80	0.80
Dream-Inst	0.80	0.80	0.80	0.80

(e) CLAD. CLAD has a single tunable hyperparameter, the confidence threshold τ, which determines whether a masked position is included as a high-confidence candidate for CIC construction. A lower threshold allows more positions to form CICs and leads to more aggressive parallel commitment, while a higher threshold makes decoding more conservative. Table 7 reports the task-specific values of τ.

Table 7: CLAD hyperparameters. Each entry reports the confidence threshold τ used for CIC construction. HE denotes HumanEval.

Model	GSM8K	MATH	HE	MBPP
LLaDA-1.5	0.73	0.70	0.75	0.73
LLaDA-Inst	0.70	0.75	0.75	0.75
Dream-Base	0.75	0.70	0.73	0.75
Dream-Inst	0.75	0.70	0.71	0.70

C Additional Results on Other Models

We additionally report results on LLaDA-1.5 and Dream-v0-Base-7B in Table 8. We follow the same evaluation setting and reporting format as in the main experiments. Acc. denotes task accuracy, TPS denotes tokens per second, and Speedup is measured relative to Vanilla decoding. Colored subscripts in the Acc. columns indicate the absolute accuracy change compared with Vanilla decoding. We further provide qualitative examples comparing CLAD with Vanilla decoding on LLaDA-8B-Instruct across the four evaluated tasks, showing the actual generated outputs of both decoding methods.

Table 8: Additional results on reasoning and code generation benchmarks with LLaDA-1.5 and Dream-v0-Base-7B. Acc. denotes task accuracy, TPS denotes tokens per second, and Speedup is measured relative to Vanilla top-1 decoding under the same setting. Colored subscripts on Acc. report the absolute change relative to Vanilla. Higher values are better for all metrics, and the best TPS and Speedup are highlighted in bold.

Task	Method	LLaDA-1.5			Dream-v0-Base-7B		
		Acc.↑	TPS↑	Speedup↑	Acc.↑	TPS↑	Speedup↑
GSM8K (5-shot)	Vanilla	81.12	4.90	1.00×	75.13	7.16	1.00×
	Fast-dLLM	$81.35_{\uparrow 0.23}$	16.61	3.39×	$74.15_{\downarrow 0.98}$	12.04	1.68×
	KLASS	$81.50_{\uparrow 0.38}$	4.77	0.97×	$72.40_{\downarrow 2.73}$	7.10	0.99×
	DAPD	$81.20_{\uparrow 0.08}$	10.86	2.22×	$71.95_{\downarrow 3.18}$	10.62	1.48×
	DAWN	$81.12_{0.00}$	22.63	4.62×	$72.71_{\downarrow 2.42}$	**13.39**	**1.87**×
	CLAD (ours)	$81.65_{\uparrow 0.53}$	**23.07**	**4.71**×	$73.54_{\downarrow 1.59}$	12.67	1.77×
MATH (4-shot)	Vanilla	33.84	7.27	1.00×	34.56	9.01	1.00×
	Fast-dLLM	$33.70_{\downarrow 0.14}$	19.35	2.66×	$34.90_{\uparrow 0.34}$	21.76	2.42×
	KLASS	$33.50_{\downarrow 0.34}$	6.97	0.96×	$30.70_{\downarrow 3.86}$	7.89	0.88×
	DAPD	$33.04_{\downarrow 0.80}$	14.63	2.01×	$31.10_{\downarrow 3.46}$	18.16	2.02×
	DAWN	$32.10_{\downarrow 1.74}$	25.03	3.44×	$34.58_{\uparrow 0.02}$	25.68	2.85×
	CLAD (ours)	$32.72_{\downarrow 1.12}$	**26.78**	**3.68**×	$33.98_{\downarrow 0.58}$	**27.47**	**3.05**×
MBPP (3-shot)	Vanilla	38.60	1.99	1.00×	52.80	9.18	1.00×
	Fast-dLLM	$38.00_{\downarrow 0.60}$	11.78	5.92×	$53.20_{\uparrow 0.40}$	25.35	2.76×
	KLASS	$25.80_{\downarrow 12.80}$	1.45	0.73×	$56.43_{\uparrow 3.63}$	8.12	0.88×
	DAPD	$38.60_{0.00}$	3.89	1.95×	$49.20_{\downarrow 3.60}$	17.02	1.85×
	DAWN	$37.40_{\downarrow 1.20}$	16.27	8.18×	$53.73_{\uparrow 0.93}$	**33.20**	**3.62**×
	CLAD (ours)	$38.80_{\uparrow 0.20}$	**16.86**	**8.47**×	$54.20_{\uparrow 1.40}$	29.06	3.17×
HumanEval (0-shot)	Vanilla	42.68	5.77	1.00×	39.02	16.10	1.00×
	Fast-dLLM	$42.68_{0.00}$	15.00	2.60×	$43.29_{\uparrow 4.27}$	30.18	1.87×
	KLASS	$42.68_{0.00}$	5.46	0.95×	$43.90_{\uparrow 4.88}$	16.98	1.05×
	DAPD	$41.46_{\downarrow 1.22}$	10.83	1.88×	$39.63_{\uparrow 0.61}$	24.73	1.54×
	DAWN	$41.46_{\downarrow 1.22}$	19.15	3.32×	$42.68_{\uparrow 3.66}$	**33.68**	**2.09**×
	CLAD (ours)	$42.68_{0.00}$	**20.71**	**3.59**×	$39.63_{\uparrow 0.61}$	33.31	2.07×

Table 9: Example generation output of LLaDA-8B-Instruct with Vanilla decoding on GSM8K (5-shot)

Item	Content
Question	Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
Output	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1. Calculate the total number of eggs laid by Janet's ducks per day. 2. Subtract the number of eggs she eats for breakfast and bakes for her friends. 3. Calculate the number of eggs she sells. 4. Multiply the number of eggs sold by the price per egg to find her daily earnings. Janet's ducks lay 16 eggs per day. She eats 3 eggs for breakfast and bakes 4 eggs for her friends, so she eats a total of 3 + 4 = 7 eggs per day. The number of eggs she sells is the total number of eggs laid minus the number of eggs she eats, which is 16 - 7 = 9 eggs per day. She sells each egg for $2, so her daily earnings are 9 eggs * $2/egg = $18. Therefore, Janet makes $18 every day at the farmers' market.
Generation Time	48.92 s

Table 10: Example generation output of LLaDA-8B-Instruct with CLAD on GSM8K (5-shot)

Item	Content
Question	Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
Output	To determine how much Janet makes every day at the farmers' market, we need to follow these steps: 1. Calculate the total number of eggs laid by Janet's ducks per day. 2. Subtract the number of eggs Janet eats for breakfast and bakes for her friends. 3. Calculate the number of eggs left for sale. 4. Multiply the number of eggs left for sale by the price per egg to find the daily earnings. 1. Janet's ducks lay 16 eggs per day. 2. She eats 3 eggs for breakfast and bakes 4 eggs for her friends, so she uses a total of 3 + 4 = 7 eggs per day. 3. The number of eggs left for sale is the total number of eggs laid minus the number of eggs used, which is 16 - 7 = 9 eggs per day. 4. She sells each egg for $2, so her daily earnings are the number of eggs left for sale multiplied by the price per egg, which is 9 eggs * $2/egg = $18 per day. Therefore, Janet makes $18 every day at the farmers' market. #### 18
Generation Time	13.07 s

Table 11: Example generation output of LLaDA-8B-Instruct with Vanilla decoding on MATH (4-shot)

Item	**Content**
Question	How many vertical asymptotes does the graph of $y = \frac{2}{x^2+x-6}$ have?
Output	To determine the number of vertical asymptotes of the function $y = \frac{2}{x^2+x-6}$, we need to identify the values of x that make the denominator zero, as these are the points where the function is undefined and may have vertical asymptotes. First, we factor the denominator $x^2 + x - 6$: $$x^2 + x - 6 = (x + 3)(x - 2)$$ Next, we set each factor equal to zero and solve for x: $$x + 3 = 0 \implies x = -3$$ $$x - 2 = 0 \implies x = 2$$ Thus, the values of x that make the denominator zero are $x = -3$ and $x = 2$. These are the points where the function has vertical asymptotes. Therefore, the graph of $y = \frac{2}{x^2+x-6}$ has two vertical asymptotes. The final answer is: $$\boxed{2}$$
Generation Time	19.54 s

Table 12: Example generation output of LLaDA-8B-Instruct with CLAD on MATH (4-shot)

Item	**Content**
Question	How many vertical asymptotes does the graph of $y = \frac{2}{x^2+x-6}$ have?
Output	To determine the number of vertical asymptotes of the function $y = \frac{2}{x^2+x-6}$, we need to find the values of x that make the denominator zero, as vertical asymptotes occur where the function is undefined. First, we set the denominator equal to zero and solve for x: $$x^2 + x - 6 = 0$$ Next, we factor the quadratic equation: $$(x+3)(x-2) = 0$$ Setting each factor equal to zero gives us the solutions: $$x + 3 = 0 \quad \text{and} \quad x - 2 = 0$$ $$x = -3 \quad \text{and} \quad x = 2$$ These values of x make the denominator zero, and thus the function $y = \frac{2}{x^2+x-6}$ has vertical asymptotes at $x = -3$ and $x = 2$. Therefore, the number of vertical asymptotes is: $$\boxed{2}$$
Generation Time	4.91 s

Table 13: Example generation output of LLaDA-8B-Instruct with Vanilla decoding on MBPP (3-shot)

Item	Content
Task	Write a function to find whether a given array of integers contains any duplicate element.

Output	`def test_duplicate(arr):`
	`  seen = set()`
	`  for num in arr:`
	`  if num in seen:`
	`  return True`
	`  seen.add(num)`
	`  return False`

Reference Tests	`assert test_duplicate(([1,2,3,4,5]))==False`
	`assert test_duplicate(([1,2,3,4, 4]))==True`
	`assert test_duplicate([1,1,2,2,3,3,4,4,5])==True`

Evaluation Result	Passed, pass@1 = 1.0
Generation Time	28.21 s

Table 14: Example generation output of LLaDA-8B-Instruct with CLAD on MBPP (3-shot)

Item	Content
Task	Write a function to find whether a given array of integers contains any duplicate element.

Output	`def test_duplicate(arr):`
	`  seen = set()`
	`  for num in arr:`
	`  if num in seen:`
	`  return True`
	`  seen.add(num)`
	`  return False`

Reference Tests	`assert test_duplicate(([1,2,3,4,5]))==False`
	`assert test_duplicate(([1,2,3,4, 4]))==True`
	`assert test_duplicate([1,1,2,2,3,3,4,4,5])==True`

Evaluation Result	Passed, pass@1 = 1.0
Generation Time	2.23 s

Table 15: Example generation output of LLaDA-8B-Instruct with Vanilla decoding on HumanEval (0-shot)

Item	Content
Task	Calculate the mean absolute deviation of a given list of numbers around its mean.
Prompt	

```
from typing import List

def mean_absolute_deviation(numbers: List[float]) -> float:
    """ For a given list of input numbers, calculate Mean Absolute
    ↪  Deviation
    around the mean of this dataset.
    Mean Absolute Deviation is the average absolute difference between
    ↪  each
    element and a centerpoint (mean in this case):
    MAD = average | x - x_mean |
    >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])
    1.0
    """
```

Output

```
from typing import List

def mean_absolute_deviation(numbers: List[float]) -> float:
    """ For a given list of input numbers, calculate Mean Absolute
    ↪  Deviation
    around the mean of this dataset.
    Mean Absolute Deviation is the average absolute difference between
    ↪  each
    element and a centerpoint (mean in this case):
    MAD = average | x - x_mean |
    >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])
    1.0
    """
    if not numbers:
        return 0.0

    mean_value = sum(numbers) / len(numbers)
    absolute_deviations = [abs(x - mean_value) for x in numbers]
    mad = sum(absolute_deviations) / len(absolute_deviations)
    return mad

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Reference Tests

```
assert abs(candidate([1.0, 2.0, 3.0]) - 2.0/3.0) < 1e-6
assert abs(candidate([1.0, 2.0, 3.0, 4.0]) - 1.0) < 1e-6
assert abs(candidate([1.0, 2.0, 3.0, 4.0, 5.0]) - 6.0/5.0) < 1e-6
```

Evaluation Result	Passed, pass@1 = 1.0
Generation Time	15.10 s

Table 16: Example generation output of LLaDA-8B-Instruct with CLAD on HumanEval (0-shot)

Item	Content
Task	Calculate the mean absolute deviation of a given list of numbers around its mean.
Prompt	

```
from typing import List

def mean_absolute_deviation(numbers: List[float]) -> float:
    """ For a given list of input numbers, calculate Mean Absolute
    ↪  Deviation
    around the mean of this dataset.
    Mean Absolute Deviation is the average absolute difference between
    ↪  each
    element and a centerpoint (mean in this case):
    MAD = average | x - x_mean |
    >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])
    1.0
    """
```

Output

```
from typing import List

def mean_absolute_deviation(numbers: List[float]) -> float:
    """ For a given list of input numbers, calculate Mean Absolute
    ↪  Deviation
    around the mean of this dataset.
    Mean Absolute Deviation is the average absolute difference between
    ↪  each
    element and a centerpoint (mean in this case):
    MAD = average | x - x_mean |
    >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])
    1.0
    """
    if not numbers:
        return 0.0

    mean_value = sum(numbers) / len(numbers)
    absolute_deviations = [abs(x - mean_value) for x in numbers]
    mad = sum(absolute_deviations) / len(numbers)
    return mad

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Reference Tests

```
assert abs(candidate([1.0, 2.0, 3.0]) - 2.0/3.0) < 1e-6
assert abs(candidate([1.0, 2.0, 3.0, 4.0]) - 1.0) < 1e-6
assert abs(candidate([1.0, 2.0, 3.0, 4.0, 5.0]) - 6.0/5.0) < 1e-6
```

Evaluation Result	Passed, pass@1 = 1.0
Generation Time	2.69 s