

CreditDecoding: Accelerating Parallel Decoding in Diffusion Large Language Models with Trace Credit

Kangyu Wang^{1,2,*}, Zhiyun Jiang^{2,3,4,*}, Haibo Feng^{2,5}, Weijia Zhao²
Lin Liu², Jianguo Li^{2,†}, Zhenzhong Lan^{2,3,†}, Weiyao Lin^{1,†}

¹Shanghai Jiao Tong University, ²Ant Group, ³Westlake University
⁴Zhejiang University, ⁵Southern University of Science and Technology

Abstract

Diffusion large language models (dLLMs) generate text through iterative denoising. In commonly adopted parallel decoding schemes, each step confirms only high-confidence positions while remasking the others. By analyzing dLLM denoising traces, we uncover a key inefficiency: models often predict the correct target token several steps before its confidence becomes high enough to be decoded. This gap between early prediction and late decoding forces repeated remasking of already-correct tokens, causing redundant iterations and limiting acceleration. To exploit this temporal redundancy, we introduce *Trace Credit* to quantify a token’s decoding potential by accumulating historical evidence. Building on this, we propose **CreditDecoding**, a *training-free* parallel decoding method that fuses Trace Credit with current logits to boost the confidence of correct but underconfident tokens, thereby accelerating denoising and improving robustness. On eight benchmarks, CreditDecoding achieves up to $5.48\times$ speedup with $+0.48$ accuracy on LLaDA-8B-Instruct, and consistently improves performance across diverse dLLM architectures and parameter scales. It further scales to long contexts and remains orthogonal to mainstream inference optimizations, making it a practical and applicable solution.

1 Introduction

Diffusion-based large language models (dLLMs) have recently emerged as a promising alternative to autoregressive models (ARMs) for text generation (Ye et al., 2025; Nie et al., 2025; Zhu et al., 2025a,b; Gong et al., 2025b,a; Song et al., 2025; Yang et al., 2025; Kim et al., 2025a). Unlike ARMs that predict tokens strictly left-to-right, dLLMs generate

*Equal Contribution. Work done during internship at Ant Group. Correspondence to kangyuwang@sjtu.edu.cn and zhiyunjiang@westlake.edu.cn.

†Corresponding authors.

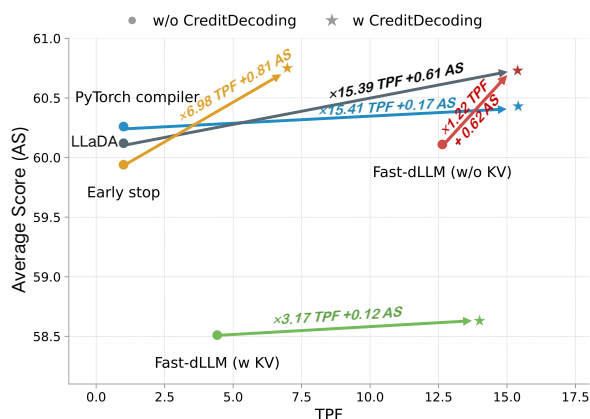


Figure 1: Acceleration methods (dots) vs. their performance with CreditDecoding (stars) on LLaDA.

text through iterative denoising with bidirectional attention, enabling richer contextual dependencies. This paradigm has demonstrated advantages in reasoning and generation quality (Nie et al., 2025; Ye et al., 2025). However, inference efficiency remains a major bottleneck: dLLMs typically require redundant denoising steps to predict all masked tokens, and the use of bidirectional attention precludes a lossless KV cache (Yu et al., 2025).

To accelerate inference, recent work has focused on improving the effectiveness and efficiency of *parallel decoding* in dLLMs (Yu et al., 2025; Wei et al., 2025). At each denoising step, the model first predicts all masked tokens and then selects a subset of high-confidence positions to decode, while remasking the remaining uncertain tokens for future refinement (Yu et al., 2025). This strategy allows multiple tokens to be updated in parallel and has proven both simple and effective. Nevertheless, it suffers from two key limitations:

(i) **Computational redundancy.** In many cases, tokens are predicted early but decoded late due to low confidence, causing repeated predictions. Figure 5 shows the resulting temporal gap between initial prediction and final decoding.

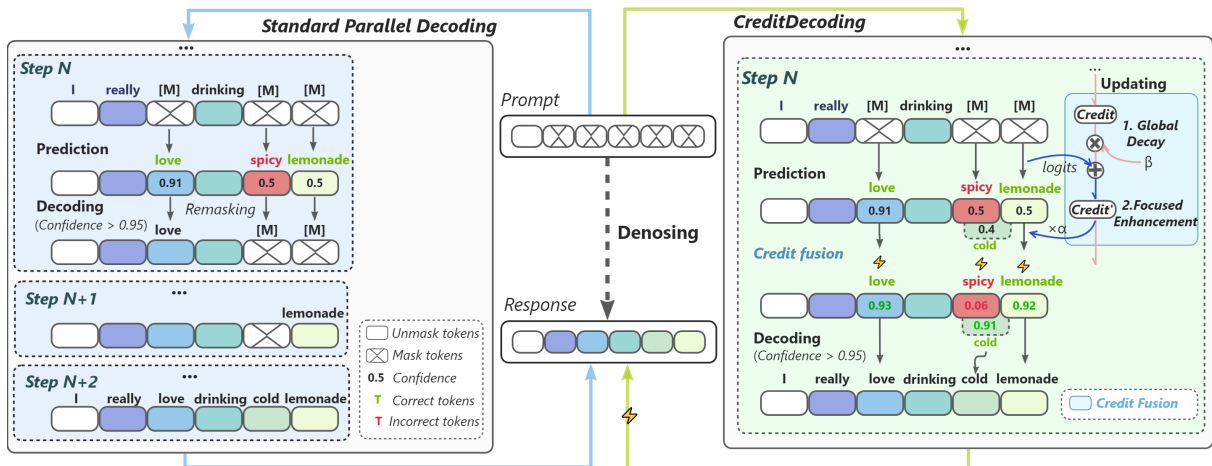


Figure 2: Comparison between **standard dLLM parallel decoding** (left) and the proposed **CreditDecoding** (right). The left diagram illustrates how existing methods predict solely on instantaneous predictions at each step, causing the repetitive remasking of correct tokens. In contrast, CreditDecoding maintains a token-level credit value across steps, using *trace credit* as a prior to enhance and calibrate current predictions.

(ii) History-agnostic Decoding. As the context is iteratively updated and may include mispredicted tokens, the confidence of otherwise stable tokens can fluctuate or even regress (Wang et al., 2025). However, token decoding is typically independent of predictions from previous steps, without leveraging the historical consistency of tokens. This undermines convergence and propagates errors across steps, reducing decoding robustness.

To address these issues, we propose **CreditDecoding**, a *training-free* parallel decoding strategy for dLLMs. It assigns each token a *trace credit* score that accumulates historical logits across steps. This score acts as a prior condition fused with the current logits to boost confidence. CreditDecoding reduces redundant iterations while stabilizing predictions against temporary inconsistencies.

We evaluate CreditDecoding on four dLLMs across eight benchmarks covering knowledge, reasoning, and coding. Experiments show that CreditDecoding achieves consistent speedups with modest performance uplift across a diverse set of dLLM architectures and benchmarks, while remaining orthogonal to mainstream optimizations such as KV cache (Feng et al., 2025) and kernel fusion (Jain et al., 2023). In summary, our work makes the following contributions:

- We analyze threshold-based parallel decoding and identify *computational redundancy* and *history-agnostic decisions* as key bottlenecks. We further find *temporal consistency* in confidence trajectories across denoising steps, which provides an effective prior for accelera-

tion.

- We propose **CreditDecoding**, a *training-free* method that accumulates trace credit as a token-level prior to accelerate inference. We also introduce a *tuning-free* variant to improve usability. On LLaDA-8B-Instruct, the primary approach achieves up to $5.48\times$ speedup while *improving* task performance.
- We demonstrate the *scalability*, *generality*, and *orthogonality* of CreditDecoding. It scales effectively across a wide range of model sizes and context lengths, while maintaining seamless compatibility with mainstream optimizations (Figure 1).

2 Related Work

Diffusion Language Models Diffusion large language models (dLLMs) replace left-to-right prediction with iterative denoising, enabling order-agnostic and parallel token updates with bidirectional context (Nie et al., 2025; Ye et al., 2025). Representative systems include Dream and the LLaDA family, with extensions to code, large-scale training, and multimodal/vision-conditioned settings (Ye et al., 2025; Nie et al., 2025; Zhu et al., 2025a,b; Gong et al., 2025b,a; Song et al., 2025; You et al., 2025; Yang et al., 2025). Variants further explore flexible-length and any-order masking (Kim et al., 2025a). Theoretically, dLLMs can approach autoregressive quality but typically require multiple denoising steps, with complexity that may grow with stricter sequence-level correctness and longer context (Feng et al., 2025; Liu et al., 2025a).

Practically, unlike ARMs, dLLMs lack lossless KV cache and often incur high latency due to many denoising iterations (Cobbe et al., 2021; Hendrycks et al., 2021b; Chen et al., 2021; Jain et al., 2024).

Inference Acceleration and Decoding Strategies To reduce latency, parallel decoding samples multiple tokens per step, and reusing or caching of bidirectional-attention outputs or other stable computations can further cut runtime without re-training (Yu et al., 2025; Wei et al., 2025; Liu et al., 2025b). Beyond speed, planning and ordering methods (e.g., path-planning remasking, adaptive ordering, calibration, or attention pruning) improve robustness and efficiency (Peng et al., 2025; Kim et al., 2025b; Huang et al., 2025; Chen et al., 2025), and some incorporate temporal/historical signals (Wang et al., 2025). However, many score-based strategies remain largely history-agnostic, relying on current-step confidence, which can induce step-level instability and redundant remasking until confidence converges.

3 Preliminary

3.1 Inference Process of dLLMs

A Diffusion Large Language Model (dLLM) generates discrete text sequences by iteratively denoising a fully masked input. dLLMs formulate generation as a stochastic reverse denoising process that starts from an all-[Mask] sequence and gradually recovers the clean sequence.

Let $x \in \mathcal{V}^L$ be a sequence of length L over a vocabulary \mathcal{V} . At each discrete step $t \in \{0, \dots, T\}$, let x_t denote the corrupted sequence and $M_t \subseteq \{1, \dots, L\}$ be the set of masked positions. We use $\eta_t \in [0, 1]$ to denote the masked ratio, following a forward schedule where noise increases with t .

The core component is a denoising model f_θ that maps a corrupted sequence x_t to logits $l_t = f_\theta(x_t)$. These logits parameterize the probability distribution of the original token at each position i :

$$p_\theta^i(\cdot | x_t) = \text{Softmax}(l_t^i). \quad (1)$$

This order-agnostic formulation enables the model to predict masked tokens based on arbitrary visible context. During inference, the denoising process starts from x_T and iteratively refines the state until the clean sequence x_0 is fully recovered. At each denoising step t , the transition from x_t to x_{t-1} is defined as:

$$x_{t-1} \sim \prod_{i=1}^L g_\theta(x_{t-1}^i | x_t), \quad (2)$$

where the transition kernel g_θ is given by:

$$g_\theta(x_{t-1}^i | x_t) = \begin{cases} x_t^i, & i \notin M_t, \\ \text{Cat}(\pi_t^i), & i \in M_t. \end{cases} \quad (3)$$

Here, $\pi_t^i(\cdot) = \frac{\eta_{t-1}}{\eta_t} \mathbf{e}_M + \frac{\eta_t - \eta_{t-1}}{\eta_t} p_\theta^i(\cdot | x_t)$, and \mathbf{e}_M denotes the one-hot vector of the [Mask].

Intuitively, at each masked position, the token either remains masked or is predicted by the model based on the schedule. In practical parallel decoding, high-confidence tokens from $p_\theta(\cdot | x_t)$ are decoded and updated into x_{t-1} , while uncertain positions are remasked for future refinement. This procedure iterates until $M_0 = \emptyset$.

Moreover, many implementations adopt a **block-wise** strategy: the sequence is partitioned into blocks, and tokens within the current block can be decoded while external tokens serve as fixed context. This limits the impact of uncertain tokens, reducing error propagation and improving stability.

3.2 Parallel Decoding

dLLMs naturally support recovering masked tokens in parallel. Assuming conditional independence at each step t , the joint distribution is approximated by the product of marginals. For each masked position $i \in M_t$, we *greedily sample* the top-1 token $\tilde{x}_t^i = \arg \max v p_\theta^i(v | x_t)$ and its confidence score $s_t^i = p_\theta^i(\tilde{x}_t^i | x_t)$ from all candidate tokens $v \in \mathcal{V}$.

Previous work validates that high-confidence marginals effectively approximate the joint distribution (Wu et al., 2025). Based on this, the mainstream strategy decodes a subset of tokens $I_t = \{i \in M_t | s_t^i \geq \tau\}$ whose confidence exceeds a threshold τ . Tokens in I_t are decoded in parallel, while the rest remain masked (Figure 2).

Beyond simple probability, other scoring functions (e.g., negative entropy, probability margins) and sampling schemes (e.g., Top- K , adaptive schedules) have also been explored.

4 Methodology

4.1 Observations

In this section, we analyze limitations of threshold-based parallel decoding in dLLM inference. For each position i , we define the **target token** $x_0^{i,v}$ as the token with ID v ultimately decoded at position i in the final output sequence.

Figure 3 (blue line) visualizes how the confidence of $x_0^{i,v}$ evolves over the denoising process. Many tokens are repeatedly predicted and

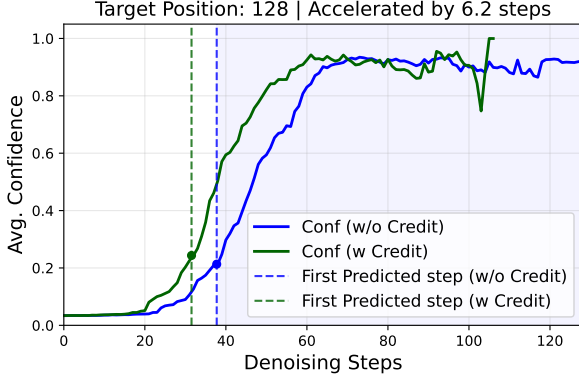


Figure 3: Impact of CreditDecoding on $x_0^{128,v}$ confidence. The shaded region marks steps where $x_0^{128,v}$ achieves top-1 confidence. CreditDecoding reduces steps by boosting confidence and entering this region earlier.

remasked long before they are ultimately decoded. This is particularly pronounced under single-token decoding schemes, which, despite this inefficiency, often lead to higher final accuracy (Feng et al., 2025).

This observation reveals two limitations of existing methods: **(i) Redundant computation.** Decisions are gated by instantaneous confidence, so correct hypotheses are repeatedly predicted and re-masked until s_t^i exceeds τ . **(ii) History-agnostic decoding.** Each step ignores past predictions; transient mispredictions can delay decoding and propagate errors to later denoising steps.

A natural idea is to *promote early decoding* by boosting the confidence of target tokens so that they cross the threshold earlier. For practicality and analytical tractability in the probability domain, we add a gain of the form $\log X$ ($X \geq 1$) to the target-token logit. Specifically, for a target token $x_0^{i,v}$, we enhance its logit $l_t^{i,v}$ as:

$$\hat{l}_t^{i,v} = l_t^{i,v} + \log X, \quad (4)$$

where $\hat{l}_t^{i,v}$ is the corresponding fused logit.

Let $p_t^{i,v} = p_{\theta}^i(v | x_t)$ denote the current probability of token v at position i . When $x_0^{i,v}$ is the predicted token at step t , we have $s_t^i = p_t^{i,v}$. It is straightforward to show that the *minimum* gain required to ensure $\hat{s}_t^i = \hat{p}_t^{i,v} \geq \tau$ is:

$$X \geq X_{\min}(p_t^{i,v}, \tau) = \frac{\tau}{1 - \tau} \cdot \left(\frac{1}{p_t^{i,v}} - 1 \right). \quad (5)$$

Eq. 5 shows that the required gain is highly sensitive to $\hat{p}_t^{i,v}$. However, early-step probabilities are unstable and the true target token is unknown.

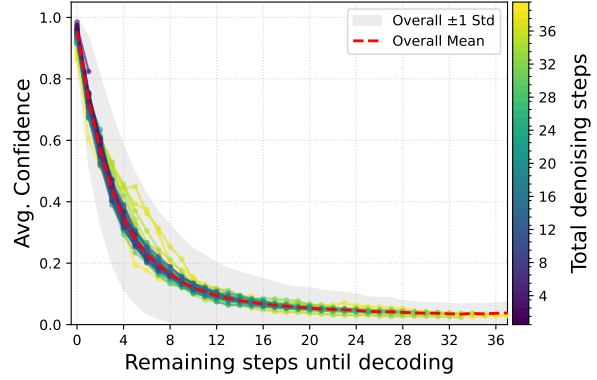


Figure 4: Temporal consistency. Results are obtained on GSM8K with threshold-based parallel decoding. Curves correspond to tokens grouped by total denoising steps.

Naively applying X_{\min} can amplify noise and lead to irreversible decoding errors. This motivates a token-level *trace credit* $C_t^{i,v}$ that aggregates historical evidence and serves as an adaptive gain for the current prediction. Intuitively, credit measures a candidate’s likelihood of converging to high confidence, enabling earlier yet safer decoding. The derivation is provided in Appendix C.1.

4.2 CreditDecoding

In this section, we introduce *CreditDecoding*, a training-free mechanism that enables earlier and safer parallel decoding.

Temporal Consistency Formally, we define the denoising trace \mathcal{T} as the ordered collection of predicted tokens and their confidence during the denoising process. Specifically, at each step t from T down to 0, all predicted tokens \tilde{x}_t and confidence scores s_t are appended to \mathcal{T} .

As shown in Figure 4, the confidence of the eventual target token $x_0^{i,v}$ exhibits a consistently increasing trend and quickly approaches 1 as it nears decoding. Moreover, this trend is largely insensitive to the length of \mathcal{T} , suggesting that the confidence of a target token is determined mainly by its *denoising stage* (i.e., the remaining steps).

Motivated by this property, we use an EMA-based credit to aggregate temporal evidence into a stable prior, predicting whether the current greedy token \tilde{x}_t^i will eventually be decoded.

Definition of Trace Credit During inference, we maintain token-level credits independently for each position. At each denoising step t , for every masked position $i \in M_t$ and token $v \in \mathcal{V}$, we maintain a credit value $C_t^{i,v} \in \mathbb{R}_{\geq 0}$ that accumulates historical evidence from \mathcal{T} for supporting token v at position i . Given the predicted token

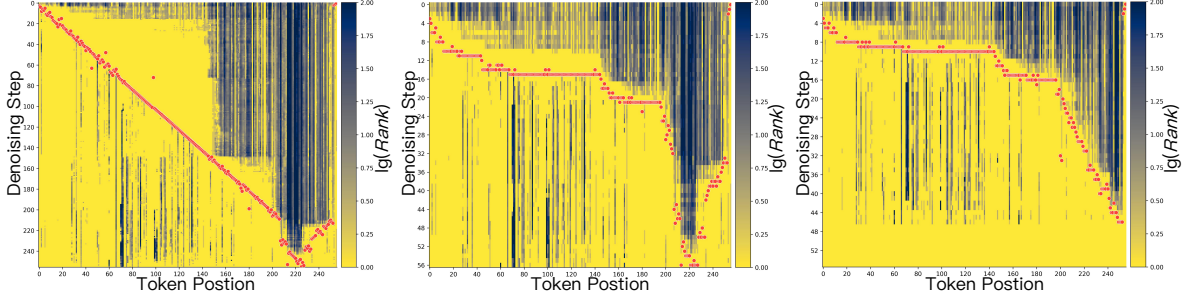


Figure 5: Confidence rank of the target token $x_0^{i,v}$ (defined in 4.1) at each position i during the denoising steps s (**Left:** LLaDA, **Middle:** Fast-dLLM, **Right:** CreditDecoding). The red dots indicate that the model decodes tokens at position i at the corresponding step s , which is also the *Decoding Boundary* in Figure C.3.

\tilde{x}_t^i , credits are updated via an EMA-style rule that reinforces \tilde{x}_t^i :

$$C_t^{i,v} = \begin{cases} \beta C_{t+1}^{i,v} + (p_t^{i,v})^\gamma, & v = \tilde{x}_t^i, \\ \beta C_{t+1}^{i,v}, & \text{otherwise.} \end{cases} \quad (6)$$

Here $\beta \in (0, 1)$ controls exponential decay, and $\gamma \in (0, 1)$ is a concave transform that up-weights low-confidence values, which is fixed in our implementation.

As illustrated in Figure 2, this update rule balances two dynamics. **(i) Global Decay:** β gradually forgets stale evidence, especially from tokens not currently predicted, suppressing early-stage confidence fluctuations. **(ii) Focused Enhancement:** for each position i , only the currently predicted token \tilde{x}_t^i receives an additional credit boost, so credit accumulates mainly on tokens that consistently rank top-1 in confidence along the denoising trace rather than on transient spikes.

For completeness, we also explore a variant that aggregates historical evidence for all tokens. Details are given in Appendix C.8. Ablations in Appendix C.9 confirm that the focused enhancement strategy ensures the optimal speed-accuracy trade-off, outperforming *Top-K Enhancement* variants.

Crucially, we fuse credit with logits to obtain a sharpened predictive distribution:

$$\tilde{l}_t^{i,v} = l_t^{i,v} + \alpha \cdot \log(C_t^{i,v} + 1) \quad (7)$$

where $\alpha > 0$ controls the strength of the credit-based prior. Eq. 7 is equivalent to applying a multiplicative gain to $p_t^{i,v}$ in the probability domain, yielding $\hat{p}_t^{i,v} = \text{Softmax}(\tilde{l}_t^i)$.

Accordingly, the confidence score becomes $\hat{s}_t^i = \max_{v \in \mathcal{V}} \hat{p}_t^{i,v}$, and the predicted token under the enhanced distribution is $\tilde{x}_t^i = \arg \max_v \hat{p}_t^{i,v}$. Tokens predicted consistently across steps thus accumulate credit, inducing a growing effective gain that

enables earlier decoding and reduces redundant remasking. Conversely, unstable candidates are down-weighted by decay, improving robustness to transient fluctuations. Importantly, CreditDecoding modifies only the output logits and seamlessly integrates with mainstream inference optimizations to achieve cumulative speedups.

In practice, we apply CreditDecoding only within the *current denoising block* for inference efficiency. Algorithm 1 provides the complete procedure.

4.3 Tuning-Free Schedule

To avoid hyperparameter tuning, we propose a step-adaptive schedule that couples credit strength to the denoising stage.

Let η_t denote the mask ratio at step t . We set $\gamma = 1$ and use step-dependent coefficients $\beta_t = \alpha_t = 1 - \eta_t$ to reflect the denoising progress, where α_t is the logit-fusion weight in Eq. 7.

In early steps, it down-weights trace credit to counteract unreliable confidence. As denoising progresses and η_t decreases, predictions stabilize and credit strength increases automatically.

5 Experiments and Analyses

5.1 Experimental Setup

Implementation We implement CreditDecoding on LLaDA-8B-Instruct (Nie et al., 2025) and LLaDA-MoE-Instruct (Zhu et al., 2025b). Inference settings differ across experiments and may deviate slightly from the original papers. We detail them in the corresponding sections. All experiments are conducted on NVIDIA H20-3e 140 GB GPUs.

In the main experiments, we set the generation length and number of steps to 256. Additional experiments with different generation lengths are

Table 1: Main benchmark results **w/ Early Stop** across eight datasets on LLaDA-8B-Instruct and LLaDA-MoE-Instruct (Gen Length=256, Block Size=64). Cells show **Score** (top, relative to LLaDA) and **TPF** (bottom, improvement over Fast-dLLM). The last row (**Avg.**) reports the mean Score and mean TPF over the eight datasets.

Benchmark	LLaDA-8B-Instruct			LLaDA-MoE-Instruct		
	Baseline	Fast-dLLM	CreditDecoding	Baseline	Fast-dLLM	CreditDecoding
MMLU <small>SCORE</small>	62.46	62.43 <small>-0.03</small>	63.78 <small>+1.32</small>	64.08	64.08 <small>0.00</small>	64.21 <small>+0.13</small>
<small>TPF</small>	1	2.86	4.57 <small>(+56%)</small>	1	2.16	2.46 <small>(+14%)</small>
SQuAD2.0 <small>SCORE</small>	91.43	91.43 <small>0.00</small>	91.71 <small>+0.28</small>	86.88	86.88 <small>0.00</small>	87.27 <small>+0.39</small>
<small>TPF</small>	1	13.55	16.84 <small>(+24%)</small>	1	7.09	9.64 <small>(+36%)</small>
DROP <small>SCORE</small>	82.86	82.74 <small>-0.12</small>	82.78 <small>-0.08</small>	80.16	80.16 <small>0.00</small>	79.72 <small>-0.44</small>
<small>TPF</small>	1	2.93	3.79 <small>(+29%)</small>	1	2.73	3.28 <small>(+20%)</small>
KorBench <small>SCORE</small>	33.12	33.20 <small>+0.08</small>	35.04 <small>+1.92</small>	36.72	36.88 <small>+0.16</small>	36.48 <small>-0.24</small>
<small>TPF</small>	1	3.72	5.03 <small>(+35%)</small>	1	2.36	3.28 <small>(+38%)</small>
HumanEval <small>SCORE</small>	34.76	34.15 <small>-0.61</small>	36.59 <small>+1.83</small>	51.22	51.22 <small>0.00</small>	51.22 <small>0.00</small>
<small>TPF</small>	1	3.82	4.69 <small>(+23%)</small>	1	4.97	6.00 <small>(+21%)</small>
LCB <small>SCORE</small>	8.15	8.15 <small>0.00</small>	7.54 <small>-0.61</small>	13.88	14.04 <small>+0.16</small>	14.37 <small>+0.49</small>
<small>TPF</small>	1	1.93	2.17 <small>(+12%)</small>	1	2.43	2.81 <small>(+16%)</small>
GSM8K <small>SCORE</small>	77.94	78.47 <small>+0.53</small>	77.18 <small>-0.76</small>	74.37	74.45 <small>+0.08</small>	74.98 <small>+0.61</small>
<small>TPF</small>	1	3.22	3.87 <small>(+20%)</small>	1	2.28	2.68 <small>(+18%)</small>
MATH <small>SCORE</small>	37.30	37.04 <small>-0.26</small>	37.24 <small>-0.06</small>	36.02	35.84 <small>-0.18</small>	36.28 <small>+0.26</small>
<small>TPF</small>	1	2.42	2.84 <small>(+17%)</small>	1	2.35	2.71 <small>(+15%)</small>
Average <small>SCORE</small>	53.50	53.45 <small>-0.05</small>	53.98 <small>+0.48</small>	55.42	55.44 <small>+0.02</small>	55.57 <small>+0.15</small>
<small>TPF</small>	1	4.31	5.48 <small>(+27%)</small>	1	3.30	4.11 <small>(+25%)</small>

reported in Section 5.5. Based on the analyses in Appendix C.4 and Section 5.3, we set the block size to 64, and the hyperparameters to $\alpha = 0.65$, $\beta = 0.7$ and $\gamma = 0.2$, which are tuned on LLaDA-8B-Instruct. All experiments are conducted w/ Early Stop for meaningful TPF and w/o KV cache, unless specified in Section 5.6

Evaluation Tasks In the main experiments, we comprehensively evaluate CreditDecoding on eight datasets spanning five categories. Specifically, we evaluate inference performance on DROP (Dua et al., 2019) and KorBench (Ma et al., 2025a), language understanding on SQuAD2.0, knowledge assessment on MMLU (Hendrycks et al., 2021a), coding ability on OpenAI HumanEval (Chen et al., 2021) and LiveCodeBench (Jain et al., 2024), and mathematical reasoning on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b). In the ablation, scaling, and other analysis experiments, due to computational constraints, we select five representative datasets across categories: MMLU, SQuAD2.0, KorBench, HumanEval, and GSM8K.

Evaluation Metrics We adopt the standard performance metrics for each evaluation dataset, as

detailed in Appendix B. In addition, to examine whether CreditDecoding can mitigate redundant computation inherent in traditional dLLMs, we utilize TPF (Tokens Per Forward) to evaluate dLLM inference efficiency. We report single-run results for each setting.

Early Stop In dLLM, early stopping changes the generated length and thus significantly affects TPF. Some studies disable it to boost TPF, as the model quickly outputs the EOS token. However, in practical applications it is usually enabled to avoid redundant outputs. We enable Early Stop by default, except in Figure 1 for better comparison.

5.2 Main Results

As shown in Table 1, we evaluate our method on eight datasets using LLaDA-8B-Instruct and LLaDA-MoE-Instruct. We use each benchmark’s default performance metric and report TPF for inference speed, with TPF of the baseline normalized to 1. TPF of CreditDecoding thus directly reflects speedup relative to the baseline.

Overall, *CreditDecoding outperforms the baseline and the SOTA Fast-dLLM in both perfor-*

mance and speed. It achieves a $5.48\times$ speedup and 0.48 performance gain on LLaDA-8B-Instruct, and a $4.10\times$ speedup on LLaDA-MoE-Instruct. Furthermore, the increase in TPF translates directly into significant *end-to-end speedups*. As detailed in Appendix C.7 and C.10, CreditDecoding consistently improves Tokens Per Second (TPS) across different acceleration scenarios on both LLaDA-8B-Instruct and LLaDA-MoE-Instruct.

Figure 5 illustrates that after applying CreditDecoding, the red line representing the decoding boundary becomes more horizontal, indicating more tokens decoded per step and higher efficiency. While the baseline requires all 256 steps, CreditDecoding completes decoding in 50 steps, consistent with the $5\times$ speedup in Table 1. Yellow and blue denote high and low confidence respectively, with their boundary marking the step where the model predicts the target token for the first time. The gap between the red line and the boundary shows the redundant steps introduced by the decoding method. Traditional dLLMs discard remasked token information, necessitating repeated predictions and creating a gap between the red line and the yellow-blue boundary.

Two observations in Figure 5 explain this acceleration: (i) The yellow-blue boundary moves upward, as trace credit acts as a temporal *low-pass filter* that suppresses confidence spikes on incorrect tokens while reinforcing those with a steadily increasing trend (Figure 4), thereby predicting the target token earlier and improving the upper limit of the red line; (ii) CreditDecoding decreases the gap between the red line and the yellow-blue boundary, because earlier decoding of correct tokens improves the intermediate context for subsequent steps, creating a positive feedback loop (Appendix C.5) that mitigates redundant steps.

5.3 Hyperparameter Ablation Study

As discussed in Section 4.2, CreditDecoding has three hyperparameters: β for global decay, α for logits fusion, and γ for concave amplification. We fix γ and perform ablation studies on α and β in the range $[0, 0.95]$ with a step size of 0.05.

Figure 6 shows that performance fluctuates slightly with α and β , peaking around $\beta = 0.5$ and $\beta = 0.7$, while α remains stable within $[0.2, 0.65]$. **Larger values of both parameters increase TPF by accumulating more trace credit, but may reduce accuracy.** We select $\alpha = 0.65$ and $\beta = 0.7$ as they provide a good trade-off, yielding strong perfor-

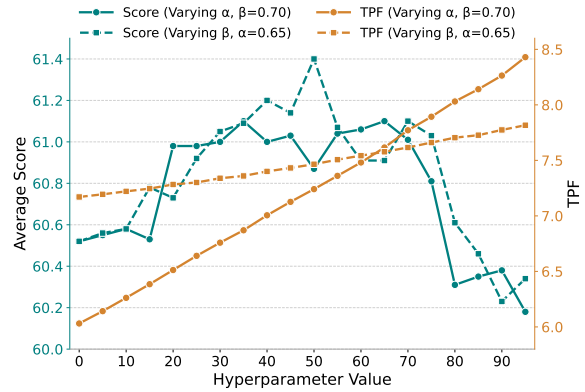


Figure 6: Hyperparameter Ablation 1. Green: average score; Yellow: average TPF. **Solid:** varying α with fixed $\beta = 0.70$; **Dashed:** varying β with fixed $\alpha = 0.65$.

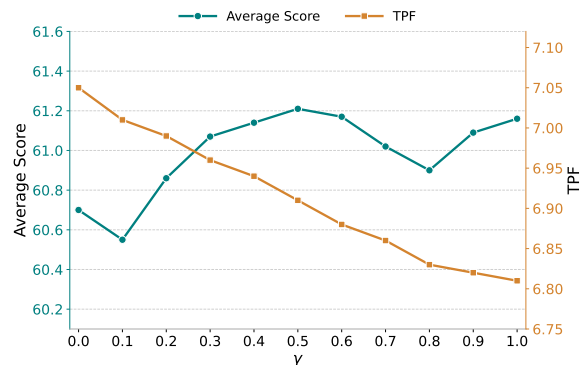


Figure 7: Hyperparameter Ablation 2. Green: average score; Yellow: average TPF. **Solid:** varying γ with fixed $\beta = 0.70$ and $\alpha = 0.65$.

mance and high TPF across five datasets, though optimal values vary by dataset.

Furthermore, we investigate the impact of γ . As detailed in Figure 7, TPF exhibits a strict negative correlation with γ . Meanwhile, similar to the trend observed in Figure 6, the generation score follows a non-linear trajectory, peaking at $\gamma = 0.5$ before showing a slight recovery as γ approaches 1.0. Consequently, we decouple γ by fixing it at a relatively small value of 0.2 in our main evaluations to ensure a robust acceleration effect.

5.4 Generalizability

In this section, we demonstrate the generalizability of CreditDecoding by conducting experiments across models characterized by diverse paradigms and scales. As shown in Table 2, in addition to the LLaDA-8B-Ins and LLaDA-MoE-Ins models discussed in the main results, we extend our evaluation to include SDAR-8B-Chat-b32 (Cheng et al., 2025), LLaDA2-Mini (Bie et al., 2025) and LLaDA2-Flash (Bie et al., 2025). Collectively, these experiments encompass various training paradigms

Table 2: Performance comparison across different model paradigms and scales. Results are reported as the average **Score / TPF** over the evaluated datasets with default hyperparameters. CD refers to CreditDecoding.

Paradigm	Arch.	Model	Params	Fast-dLLM	CD	CD-Adaptive
Pure Diffusion	Dense	LLaDA-8B-Ins	8B	60.38 / 5.49	60.86 / 7.00	60.23 / 7.74
	MoE	LLaDA-MoE-Ins	7B-A1B	62.70 / 3.77	<u>62.83 / 4.81</u>	63.74 / 5.27
Block Diffusion	Dense	SDAR-Chat-b32	8B	65.37 / 1.72	<u>65.51 / 1.91</u>	66.37 / 1.83
	MoE	LLaDA2-Mini	16B-A1B	76.23 / 2.01	<u>76.69 / 2.26</u>	76.92 / 2.15
	MoE	LLaDA2-Flash	100B-A6B	84.89 / 2.48	<u>84.53 / 2.79</u>	84.97 / 2.68

and architectures, spanning parameter scales from 8B to 100B. We employ dInfer (Ma et al., 2025b) for inference, configuring the generation length to 2048 and the block size to 32. Due to computational constraints, we limit our comparison to Fast-dLLM, CreditDecoding, and the CreditDecoding-Adaptive, which adapts to the step mask ratio introduced in Section 4.3, omitting weaker baselines that are incompatible with parallel decoding.

Results in Table 2 yield two critical insights. First, *CreditDecoding strategy delivers more substantial efficiency gains in Pure Diffusion training paradigms* due to its larger attention context window. Second, and most significantly, *CreditDecoding-Adaptive serves as a universal, tuning-free solution*. Addressing the dataset-dependent nature of denoising traces \mathcal{T} (Appendix C.6), it eliminates the need for task-specific hyperparameter tuning, thereby reducing deployment overhead. It yields superior generation scores and remains significantly faster than Fast-dLLM, balancing performance and usability with minimal efficiency cost.

5.5 Scalability

Current research on dLLMs typically evaluates generation lengths of 128 or 256, with a few studies extending to 512. However, the primary advantage of dLLMs over autoregressive models—parallel decoding—is largely underutilized at such short lengths. To address this, in this section we scale the generation length to 1024 and 4096, aiming to provide insights into the potential of dLLMs for long-text generation.

We fix the number of steps equal to the generation length and set the block size to 64. Figure 8 shows that both the baseline and CreditDecoding peak at length 512 and gradually degrade as length increases. Notably, *CreditDecoding consistently outperforms the baseline and scales better with generation length*, demonstrating stronger robustness in long-context scenarios.

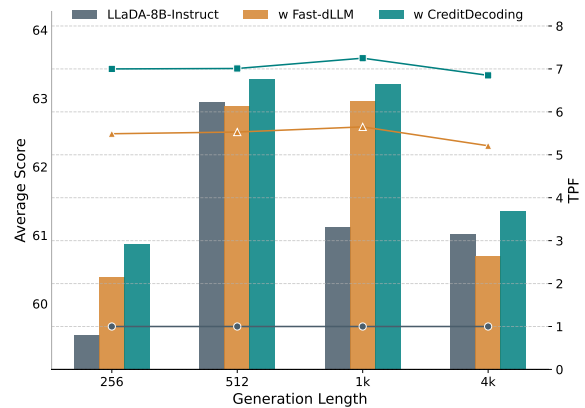


Figure 8: Scalability of CreditDecoding on LLaDA. Lines indicate average TPF; bars show average score.

Crucially, the method sustains a consistent $7\times$ TPF speedup across settings. As text length grows, *this relative speedup translates into increasingly substantial absolute latency reductions*, highlighting its practical value in long-context scenarios.

5.6 Orthogonality

CreditDecoding operates purely on the model logits and does not interfere with the *sample* or *select* procedures. This design makes it a *plug-and-play* post-processing module that is naturally orthogonal to both (i) system-level inference optimizations such as compiler-level acceleration (Jain et al., 2023), and (ii) algorithmic acceleration strategies for dLLMs such as threshold decoding, KV cache variants, and EOS early stopping.

In all cases, we keep the hyperparameters and selection rules of the baseline methods unchanged, and only apply CreditDecoding on top. As demonstrated in Figure 1, our experiments confirm this orthogonality. When CreditDecoding is combined with these methods, we observe that the speedup is preserved while the performance drop is partially mitigated, showing that historical trace credit acts as a stabilizing prior under aggressive compression.

For algorithmic acceleration, threshold parallel decoding (Fast-dLLM w/o KV cache (Wu et al.,

Table 3: Performance and efficiency comparison of CreditDecoding across different decoding strategies.

Decoding Algorithm	Score	Score (w/ Credit)	TPF	TPF (w/ Credit)
Top-2	57.59	57.73 (+0.14)	2.00	2.00 (+0.00)
Top-4	52.92	53.79 (+0.87)	4.00	4.00 (+0.00)
Top-8	46.94	47.18 (+0.24)	8.00	8.00 (+0.00)
Top- K Margin	60.44	61.20 (+0.76)	5.49	6.27 (+0.78)
Threshold (0.95)	60.38	60.86 (+0.48)	5.49	7.00 (+1.51)

2025)) typically accelerates but at the cost of lower accuracy. For EOS early stopping, CreditDecoding complements the strategy by improving robustness of token selection in earlier steps. Adding CreditDecoding further boosts both speed and accuracy, with $4.7\times$ acceleration over the baseline with an additional $+0.63$ improvement in average score. Similarly, CreditDecoding alleviates the accuracy loss of threshold decoding and complements KV cache methods by reducing redundant iterations within cached segments. These results demonstrate that *CreditDecoding can be seamlessly integrated with existing optimizations to deliver consistent performance gains*.

Appendix C.7 provides details of our acceleration methods and orthogonality experimental results for CreditDecoding on LLaDA and LLaDA-MoE, including *inference throughput (TPS)*.

5.7 Versatility

To comprehensively evaluate CreditDecoding across diverse mainstream decoding schemes, we supplement our study with *Top- K* decoding experiments ($K \in \{2, 4, 8\}$), where a strictly fixed number of highest-confidence tokens are decoded at each step.

It should be noted that CreditDecoding primarily accelerates decoding by accumulating historical trajectories into the trace credit, enabling high-confidence tokens to reach the decoding threshold earlier. This acceleration is directly reflected in the improvement of TPF. However, *Top- K* decoding intrinsically fixes the token count per step, which inherently locks the TPF and overall decoding speed. Consequently, the TPF remains identical across all decoding methods under this setting, structurally neutralizing, the inference speedup that acceleration techniques would otherwise provide. Furthermore, consistent with prior studies, fixing the token count per step often leads to substantial performance degradation compared to threshold-based methods, which typically offer superior performance and lower latency. This phenomenon is

also clearly reflected in our empirical results.

To demonstrate the robustness of CreditDecoding across different selection rules, we additionally incorporate the *Top- K margin* decoding method. The comparative results are detailed in Table 3.

As illustrated above, while the fixed decoding rate of *Top- K* fundamentally nullifies the potential for acceleration (TPF remains constant), CreditDecoding still yields consistent generation performance improvements. This demonstrates that leveraging the historical information stored in the trace credit enhances prediction accuracy even when speedups are bottlenecked. More importantly, under flexible decoding settings such as *Top- K margin* and threshold-based decoding, the integration of CreditDecoding consistently delivers substantial enhancements in both generation quality and inference efficiency.

6 Conclusion

The history-agnostic decoding process of diffusion language models inherently suffers from substantial computational redundancy. To address this, we propose CreditDecoding, a training-free decoding strategy orthogonal to mainstream acceleration methods. Specifically, it assigns a trace credit to each candidate token by accumulating historical logit predictions from remasked steps, and subsequently fuses this credit into the current logits.

By fully leveraging the model’s past predictions on remasked tokens, CreditDecoding reduces the complexity of current inference steps. This leads to simultaneous improvements in performance and decoding efficiency, achieving a $5.48\times$ speedup and an average accuracy gain of $+0.48$ compared to standard LLaDA-8B-Instruct.

Ultimately, CreditDecoding aims to approach the theoretical limit of acceleration (indicated by the yellow-blue boundary in Figure 5). As more powerful base models emerge, the acceleration ceiling of CreditDecoding will rise, leading to further efficiency gains in model inference.

Limitations

Although CreditDecoding provides consistent acceleration and accuracy gains, it also has several limitations.

First, it relies on the accumulation of historical evidence along the denoising trace, and thus requires a certain number of denoising steps to build effective trace credit. For pure-diffusion dLLMs that adopt small block sizes to trade inference speed for better performance, and for dLLMs that are explicitly trained for aggressive parallel decoding whose logit distributions are sharp with fast confidence convergence, our approach may not have enough denoising steps to accumulate sufficient trace credit. In these cases, the acceleration benefit becomes limited. Experiments in Appendix C.4 also show that increasing the block size generally allows stronger credit accumulation and leads to larger speedup.

Second, due to variations in dataset characteristics, such as data pattern or sample difficulty, and inherent model behavior, the optimal setting for α , β and γ may differ across tasks and models. Although we propose a tuning-free adaptive variant in Sec. 4.3 that yields stable and consistent improvements across model architectures, benchmarks, and parameter scales, it does not always achieve the theoretically optimal performance–efficiency trade-off. In practice, obtaining the best benefit still requires lightweight hyperparameter search on a small validation set.

Third, we observe a slight performance degradation on reasoning-intensive tasks, such as mathematical reasoning and complex code generation. These tasks inherently possess strong causal dependencies and unstable confidence convergence, which can interfere with the trace credit accumulation during highly parallel decoding. A detailed failure mode analysis regarding these tasks is provided in Appendix C.11.

Acknowledgements

We thank the anonymous reviewers and the area chair for their constructive comments. This work was supported by Ant Group Research Fund, and supported in part by the National Natural Science Foundation of China (Grant No. 62595733, 62561160155, 62325109), the Shanghai 'The Belt and Road' Young Scholar Exchange Grant (Grant No. 24510742000), and the Key R&D Program of Zhejiang (Grant No. 2025C01104).

References

- Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, and 12 others. 2025. [Llada2.0: Scaling up diffusion language models to 100b](#). *Preprint*, arXiv:2512.15745.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *Preprint*, arXiv:2107.03374.
- Xinhua Chen, Sitao Huang, Cong Guo, Chiyue Wei, Yintao He, Jianyi Zhang, Hai "Helen" Li, and Yiran Chen. 2025. [Dpad: Efficient diffusion language models with suffix dropout](#). *Preprint*, arXiv:2508.14148.
- Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. 2025. [Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation](#). *Preprint*, arXiv:2510.06303.
- 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](#). *Preprint*, arXiv:2110.14168.
- Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. [Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2368–2378. Association for Computational Linguistics.
- Guhao Feng, Yihan Geng, Jian Guan, Wei Wu, Liwei Wang, and Di He. 2025. [Theoretical benefit and limitation of diffusion language model](#). *Preprint*, arXiv:2502.09622.
- Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025a. [Scaling diffusion language models via adaptation from autoregressive models](#). *Preprint*, arXiv:2410.17891.
- Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jitao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. 2025b. [Diffucoder: Understanding and improving masked diffusion models for code generation](#). *Preprint*, arXiv:2506.20639.

- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. [Measuring massive multitask language understanding](#). *Preprint*, arXiv:2009.03300.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. [Measuring mathematical problem solving with the math dataset](#). *Preprint*, arXiv:2103.03874.
- Pengcheng Huang, Shuhao Liu, Zhenghao Liu, Yukun Yan, Shuo Wang, Zulong Chen, and Tong Xiao. 2025. [Pc-sampler: Position-aware calibration of decoding bias in masked diffusion models](#). *Preprint*, arXiv:2508.13021.
- Animesh Jain, Shunting Zhang, Edward Yang, and 1 others. 2023. [Pytorch 2.0: Our next generation 2.0 release](#).
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. [Live-codebench: Holistic and contamination free evaluation of large language models for code](#). *Preprint*, arXiv:2403.07974.
- Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngatiaoco, Sitan Chen, and Michael Albergo. 2025a. [Any-order flexible length masked diffusion](#). *Preprint*, arXiv:2509.01025.
- Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. 2025b. [Train for the worst, plan for the best: Understanding token ordering in masked diffusions](#). *Preprint*, arXiv:2502.06768.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with pagedattention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626. Association for Computing Machinery.
- Xiaoran Liu, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025a. [Longllada: Unlocking long context capabilities in diffusion llms](#). *Preprint*, arXiv:2506.14429.
- Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. 2025b. [d1lm-cache: Accelerating diffusion large language models with adaptive caching](#). *Preprint*, arXiv:2506.06295.
- Kaijing Ma, Xinrun Du, Yunran Wang, Haoran Zhang, Zhoufutu Wen, Xingwei Qu, Jian Yang, Jiaheng Liu, Minghao Liu, Xiang Yue, Wenhao Huang, and Ge Zhang. 2025a. [Kor-bench: Benchmarking language models on knowledge-orthogonal reasoning tasks](#). *Preprint*, arXiv:2410.06526.
- Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, and 4 others. 2025b. [dinfer: An efficient inference framework for diffusion language models](#). *Preprint*, arXiv:2510.08666.
- 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](#). *Preprint*, arXiv:2502.09992.
- Fred Zhangzhi Peng, Zachary Bezemek, Sawan Patel, Jarrid Rector-Brooks, Sherwood Yao, Avishek Joey Bose, Alexander Tong, and Pranam Chatterjee. 2025. [Path planning for masked diffusion model sampling](#). *Preprint*, arXiv:2502.03540.
- Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, and 3 others. 2025. [Seed diffusion: A large-scale diffusion language model with high-speed inference](#). *Preprint*, arXiv:2508.02193.
- Wen Wang, Bozhen Fang, Chenchen Jing, Yongliang Shen, Yangyi Shen, Qiuyu Wang, Hao Ouyang, Hao Chen, and Chunhua Shen. 2025. [Time is a feature: Exploiting temporal dynamics in diffusion language models](#). *Preprint*, arXiv:2508.09138.
- Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Dongrui Liu, and Linfeng Zhang. 2025. [Accelerating diffusion large language models with slowfast sampling: The three golden principles](#). *Preprint*, arXiv:2506.10848.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. 2025. [Fast-d1lm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding](#). *Preprint*, arXiv:2505.22618.
- Ling Yang, Ye Tian, Bowen Li, Xinchun Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. 2025. [Mmada: Multimodal large diffusion language models](#). *Preprint*, arXiv:2505.15809.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025. [Dream 7b: Diffusion large language models](#).
- Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. 2025. [Llada-v: Large language diffusion models with visual instruction tuning](#). *Preprint*, arXiv:2505.16933.
- Runpeng Yu, Xinyin Ma, and Xinchao Wang. 2025. [Dimple: Discrete diffusion multimodal large language model with parallel decoding](#). *Preprint*, arXiv:2505.16990.

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

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tiejuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, and 7 others. 2025b. *Llada-moe: A sparse moe diffusion language model*. *Preprint*, arXiv:2509.24389.

A Statements

In this section, we provide clarifications on issues that require explicit statements. The Ethics Statement is omitted, as this work does not raise any ethical concerns.

A.1 Reproducibility.

We employ four widely adopted and advanced dLLM models, LLaDA-8B-Instruct, LLaDA-MoE-Instruct, LLaDA2-Mini and LLaDA2-Flash, along with their publicly available weights. Detailed experimental configurations for each setup are provided, and the specific metric configurations for score evaluation are included in Appendix B. We utilize TPF, a metric that is relatively robust to confounding factors, as it is not influenced by the number of GPUs, hardware model, or inference framework. Therefore, we argue that the results reported in this paper are highly reproducible.

A.2 Use of LLMs

In this study, LLMs were used only to check and improve the accuracy, coherence, and academic quality of the writing. We confirm that LLMs were not employed to generate any experimental data or substantive content. All data originate from our own experiments, and all analyses and conclusions represent the authors’ original work.

B Evaluation Config

We employ OpenCompass to assist in the evaluation process, ensuring a standardized and systematic assessment. For each benchmark (LCB for LiveCodeBench), we use the specific metrics presented in Table 4, which allow for a consistent and comprehensive comparison across different tasks. Regarding in-context learning (ICL) configurations, all benchmarks were evaluated in a zero-shot setting, except for Drop and SQuAD2, which were

Table 4: Benchmarks, their corresponding evaluation metrics, and In-Context Learning (ICL) setup.

Benchmark	Metric	ICL
GSM8K	Accuracy	0-shot
Math	Accuracy	0-shot
SQuAD2.0	Score	1-shot
DROP	Score	2-shot
MMLU	Weighted Average	0-shot
KorBench	Naive Average	0-shot
LCB OC Code Generation v6	Score	0-shot
OpenAI HumanEval	Pass@1	0-shot

evaluated in two-shot and one-shot settings respectively.

C CreditDecoding Supplement

In this section, we provide additional details and experiments related to CreditDecoding that were not presented in the main body of the paper.

C.1 Deriving the Minimum Logit Gain

In this subsection, we derive the minimum gain X induced by adding $\log X$ to a token logit, such that the boosted token probability exceeds the decoding threshold.

Consider a fixed position i at denoising step t with logits $\{l_t^{i,u}\}_{u \in \mathcal{V}}$ and the corresponding softmax distribution

$$p_t^{i,u} = \frac{e^{l_t^{i,u}}}{\sum_{z \in \mathcal{V}} e^{l_t^{i,z}}}. \quad (8)$$

Let $v \in \mathcal{V}$ be the token to be boosted. We apply a logit gain:

$$\hat{l}_t^{i,u} = \begin{cases} l_t^{i,u} + \log X, & u = v, \\ l_t^{i,u}, & u \neq v, \end{cases} \quad X \geq 1, \quad (9)$$

and define the enhanced distribution by softmax:

$$\hat{p}_t^{i,u} = \frac{e^{\hat{l}_t^{i,u}}}{\sum_{z \in \mathcal{V}} e^{\hat{l}_t^{i,z}}}. \quad (10)$$

Substituting Eq. 9 into Eq. 10, we obtain:

$$\begin{aligned} \hat{p}_t^{i,v} &= \frac{e^{l_t^{i,v} + \log X}}{e^{l_t^{i,v} + \log X} + \sum_{u \neq v} e^{l_t^{i,u}}} \\ &= \frac{X e^{l_t^{i,v}}}{X e^{l_t^{i,v}} + \sum_{u \neq v} e^{l_t^{i,u}}} \\ &= \frac{X p_t^{i,v}}{(1 - p_t^{i,v}) + X p_t^{i,v}}. \end{aligned} \quad (11)$$

Algorithm 1 CreditDecoding

```

1: Input: denoiser  $f_\theta$ , threshold  $\tau$ , fusion strength  $\alpha$ , decay
    $\beta$ , boost exponent  $\gamma$ 
2: Initialize  $x_T \leftarrow ([\text{MASK}], \dots, [\text{MASK}])$ 
3: for each decoding block  $[b_s, b_e]$  do
4:   Initialize credit tensor  $C \leftarrow \mathbf{0}$   $\triangleright$  Per-block credit
5:   for  $t = T, T-1, \dots, 1$  do  $\triangleright$  Denoising
6:      $x_t^{\text{blk}} \leftarrow x_t[b_s:b_e]$ 
7:      $m \leftarrow \mathbb{I}[x_t^{\text{blk}} = [\text{MASK}]]$ 
            $\triangleright$  Prediction and Greedy selection
8:      $\ell \leftarrow f_\theta(x_t)_{b_s:b_e}$ 
9:      $p \leftarrow \text{Softmax}(\ell)$ 
10:     $(p^{\max}, v^{\max}) \leftarrow \max_{v \in \mathcal{V}} p(\cdot)$ 
11:     $\Delta \leftarrow (p^{\max})^\gamma \odot m$ 
            $\triangleright$  Credit Update
12:     $C \leftarrow \beta \cdot C$   $\triangleright$  Global decay
13:     $C \leftarrow C + \text{ScatterAdd}(C, v^{\max}, \Delta)$   $\triangleright$  Focused
           enhancement
14:     $\hat{\ell} \leftarrow \ell + \alpha \cdot \log(1 + C)$   $\triangleright$  Logits fusion
            $\triangleright$  Threshold-based Decoding
15:     $\hat{p} \leftarrow \text{Softmax}(\hat{\ell})$ 
16:     $(s^{\max}, \tilde{x}) \leftarrow \max_{v \in \mathcal{V}} \hat{p}(\cdot)$   $\triangleright$  Predicted tokens
           and confidence
17:     $\text{idx} \leftarrow (s^{\max} \geq \tau) \wedge m$   $\triangleright$  Positions to decode
            $\triangleright$  Update block state
18:     $x_{t-1}[b_s:b_e] \leftarrow \text{where}(\text{idx}, \tilde{x}, x_t^{\text{blk}})$ 
19:  end for
20: end for
21: return  $x_0$ 

```

To decode token v under threshold $\tau \in (0, 1)$, we require $\hat{p}_t^{i,v} \geq \tau$. By Eq. 11:

$$X \geq X_{\min}(p_t^{i,v}, \tau) = \frac{\tau}{1-\tau} \cdot \frac{1-p_t^{i,v}}{p_t^{i,v}}. \quad (12)$$

Consequently, for under-confident tokens (e.g., $p_t^{i,v} \approx 1/|\mathcal{V}|$), the required credit X becomes prohibitively large, validating that naive gain application is risky in early denoising steps with unstable probabilities.

C.2 Algorithm

To accurately illustrate the workflow of the CreditDecoding algorithm, we present its detailed steps in Algorithm 1. Here, T denotes the total number of decoding steps, and r, s, t correspond to the mask ratios of the previous, current, and subsequent steps, respectively. Lines 4–12 describe the process of trace credit accumulation, where historical trace credit are decayed by a factor β and the token with the highest confidence is assigned additional trace credit. Lines 13–14 show the application of trace credit, in which past information is integrated into the current step by fusing it with the original logits. Finally, Lines 16–23 represent the mainstream approach to parallel decoding, namely the thresholding method, whose effectiveness has been demonstrated in Fast-dLLM (Wu et al., 2025).

C.3 Ideal Decoding Boundary

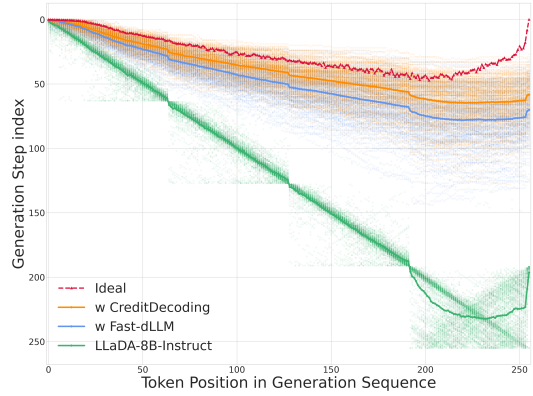
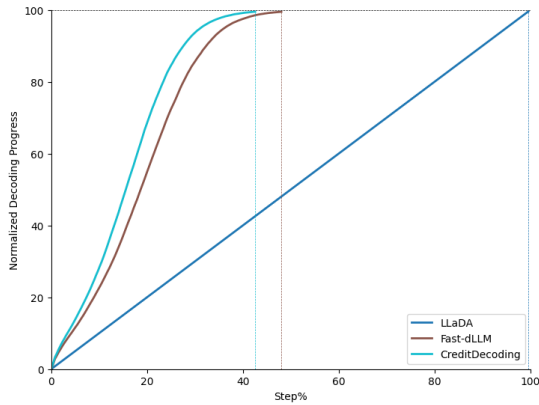


Figure 9: Decoding Boundary between token stabilization and final decoding. Evaluated on GSM8K using LLaDA-8B-Instruct, the *red* curve represents the theoretical decoding bound, while others show actual decoding bound. The persistent gap highlights the computational redundancy where correct tokens are repeatedly remasked despite early stabilization.

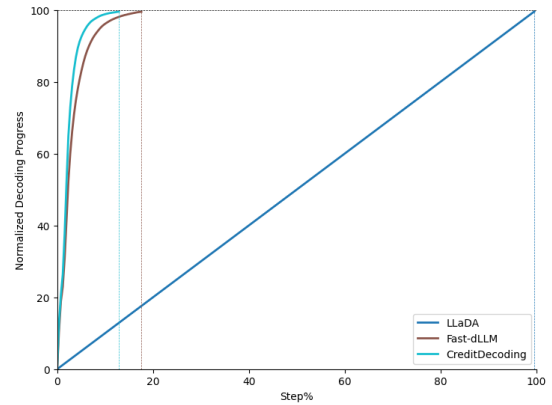
In Figure 9, we illustrate the decoding boundaries of three methods, LLaDA, Fast-dLLM, and CreditDecoding, within a single plot. Figure 5 further presents the decoding boundaries (highlighted by red lines) from the perspective of confidence rank, along with an analysis of the ideal decoding boundary. For a more intuitive comparison, the ranges of the plots in Figure 9 are aligned, whereas the vertical scales of the three subplots in Figure 5 differ to emphasize the detailed variations introduced by thresholding. The line connecting the first appearance of yellow points marks where target tokens are initially predicted.

It is worth noting that, as shown in Figure 5, various methods yield distinct ideal decoding boundaries. This variation arises because each method receives distinct inputs at every step, resulting in different confidence distributions. Importantly, stronger models or more effective methods tend to produce an ideal decoding boundary that shifts upward. Our CreditDecoding approach, however, is orthogonal to most existing methods and can further improve their ideal decoding boundaries, bringing the actual decoding boundary closer to the ideal one.

In Figure 10, we visualize the accumulated decoded tokens per step for LLaDA, Fast-dLLM, and CreditDecoding on specific datasets. Two key observations emerge: First, the speedup of the same method varies across datasets. Second, CreditDecoding consistently outperforms Fast-dLLM, especially on SQuAD2.0, where it reduces the denois-



(a) Normalized Decoding Progress on GSM8K



(b) Normalized Decoding Progress on SQuAD2.0

Figure 10: Normalized Decoding Progress **w/o Early Stop** on **GSM8K** and **SQuAD2.0**. We demonstrate the *decoding progress* through visualizing the accumulated number of decoded tokens per step, using **LLaDA-8B-Instruct** with *generation length* = 256 and *block size* = 64. In order to ensure comparability of the decoding progress, we did not use early stopping. The vertical dashed lines in the figure mark the denoising step at which each method completes, and the ratio of these steps represents the speedup.

ing steps by an additional 5% compared to Fast-dLLM’s 20% reduction. As an orthogonal method, CreditDecoding offers greater improvements with higher speedups.

C.4 Block Length Ablation

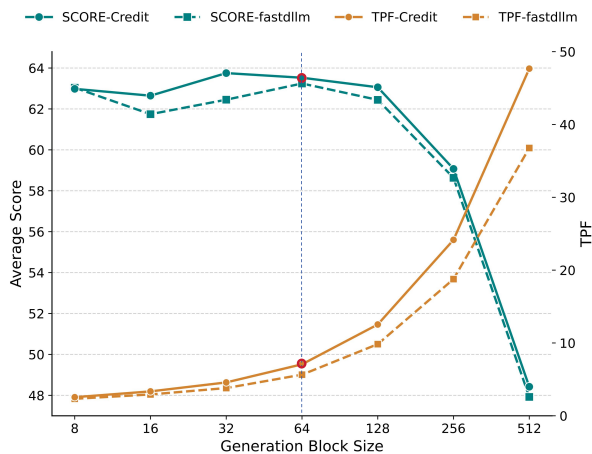


Figure 11: Block Length Ablation Study

In the parallel decoding framework of generative models, the choice of generation block length directly determines the balance between system performance and efficiency. Through block length ablation experiments analyzing Fast-dLLM and CreditDecoding, as shown in Figure 11, it can be observed that when the block length is set to 64, both methods demonstrate optimal balance. Under this configuration, both CreditDecoding and Fast-dLLM achieve higher average scores, indicating that this scale fully unleashes the potential of

block-level generation mechanisms without significantly sacrificing generation quality. In contrast, smaller block lengths (such as 32) can maintain high performance levels but result in significantly lower inference speeds, making it difficult to meet the high-throughput requirements of practical deployment. Meanwhile, larger block lengths (such as 256 and above) significantly improve TPF but lead to a sharp performance decline. Therefore, a block length of **64** is an ideal choice considering both accuracy and practicality.

We also observe that the performance degradation with increasing block size is caused by the parallel decoding mechanism itself, as dLLMs inevitably suffer from prematurely decoded tokens. This issue also appears in other training-free methods like Top- K or entropy-based thresholding. Nevertheless, CreditDecoding significantly alleviates this degradation. By capturing *the temporal consistency of confidence* during denoising, CreditDecoding effectively filters out noise from premature predictions, thereby better maintaining performance while accelerating compared to Fast-dLLM.

A closer comparison in Figure 11 reveals that CreditDecoding demonstrates greater advantages in overall performance. In terms of performance, CreditDecoding reaches its peak at a block length of 32 and consistently performs on par with or better than Fast-dLLM across medium and small scales. Even under large block lengths where both methods decline, CreditDecoding still exhibits bet-

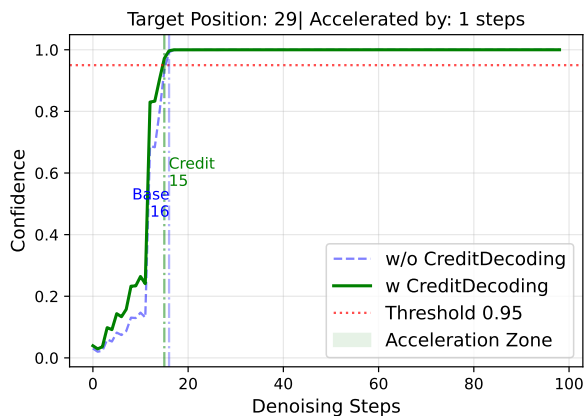


Figure 12: A case study comparing the confidence evolution of a target token under Fast-dLLM and CreditDecoding. It illustrates the direct impact of trace credit on a single token. While this local gain appears minor, it iteratively improves the context for subsequent tokens, which ultimately yields a 21.3% reduction in total inference steps.

ter error control. Regarding speed, CreditDecoding shows higher TPF across all scales, and the performance gap widens as block length increases, reflecting its architectural advantages in parallel scenarios.

In summary, CreditDecoding’s core value lies in its superior generation performance compared to Fast-dLLM, along with higher overall efficiency. Future work could explore adaptive block sizing or hybrid decoding strategies to dynamically adjust block length and further optimize the balance between performance and speed across different sequence lengths.

C.5 Contextual Cumulative Effect

Notably, CreditDecoding does not merely accelerate the decoding of an individual token; by decoding some tokens earlier, it improves the context for subsequent denoising steps and can increase the confidence of other still-masked positions as well. Figure 12 provides a representative example: although CreditDecoding reduces the number of denoising steps, the confidence trajectory of the target token remains largely unchanged, suggesting that the speedup mainly stems from earlier decoding enabled by improved intermediate context, rather than altering the model’s intrinsic predictions.

C.6 Dataset-Dependent Denoising Traces

Further analysis in Figure 13 compares GSM8K and HumanEval. GSM8K requires more context from prior steps, leading to delayed but sharp confidence gains, while HumanEval shows earlier, grad-

ual confidence increases. With CreditDecoding, HumanEval retains its confidence trend while reducing inference steps, whereas GSM8K experiences minor fluctuations that contribute to the performance drop reported in Table 1.

In Figures 13 (c) and (d), we observe the confidence convergence of four specific tokens. On GSM8K, the confidence increase is delayed for tokens later in the sequence, indicating that later tokens rely on earlier tokens to build confidence. Once earlier tokens are decoded, confidence increases rapidly. On HumanEval, confidence increases earlier in the sequence but at a slower rate.

In Figures 13 (e) and (f), after applying CreditDecoding, we observe that for HumanEval, CreditDecoding retains the original confidence trends while reducing the total inference steps, leading to faster inference without loss of accuracy. For GSM8K, early predictions introduce slight confidence fluctuations, contributing to the performance degradation observed in Table 1.

CreditDecoding essentially reduces the inference complexity by leveraging past judgments, improving both accuracy and efficiency. Its goal is to approach the yellow-blue boundary in Figures (a) and (b), representing the theoretical limit of acceleration. As more powerful base models emerge, the acceleration ceiling of CreditDecoding will increase, leading to further efficiency gains in model inference.

C.7 Orthogonality

In Section 5.6, we demonstrate the orthogonality and compatibility of CreditDecoding through experiments combining it with several acceleration techniques. Results show that CreditDecoding consistently improves both speed and performance across all tested methods.

In this section, we provide brief introductions to the acceleration methods discussed in Section 5.6 and include additional TPS results to better illustrate CreditDecoding’s effectiveness, particularly on system-level accelerations that mainly improve TPS. We also extend our orthogonality analysis to LLaDA-MoE, with detailed results presented below.

We evaluate its orthogonality on four representative acceleration techniques, as illustrated in Figure 1.

Early Stop: Early Stop terminates decoding when the current token is <EOS> and all previous tokens are finalized, effectively reducing redundant

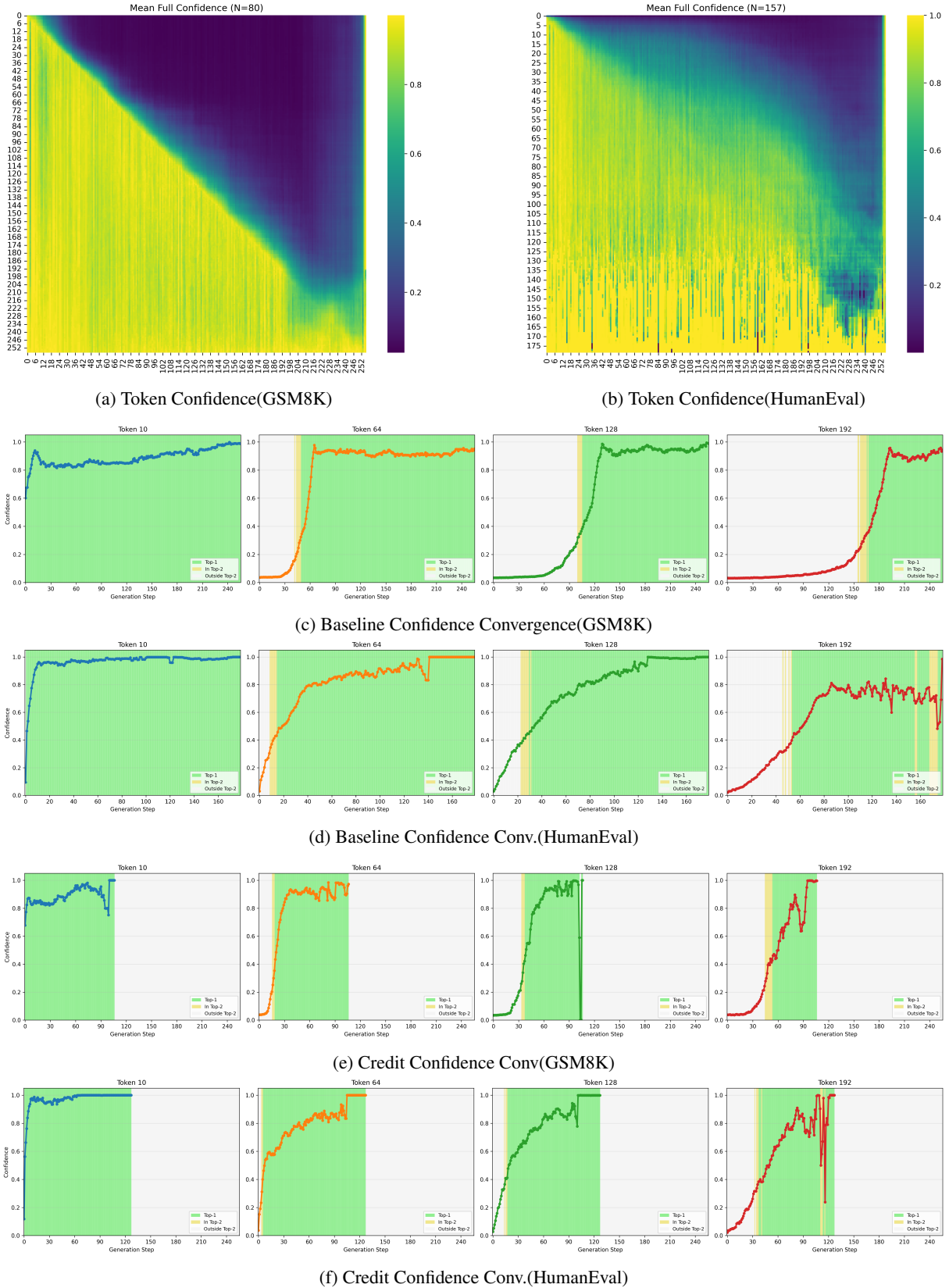


Figure 13: Token confidence across datasets. Figures (a) and (b) present the average confidence of N tokens at each position during inference. Figures (c)–(f) illustrate the convergence behavior of four representative tokens located at positions 10, 64, 128, and 192, respectively. Results in (a), (c), and (e) are sampled from GSM8K, while those in (b), (d), and (f) are from HumanEval. Panels (c) and (d) show baseline convergence, and (e) and (f) show convergence with CreditDecode.

Table 5: Orthogonality analysis of CreditDecoding (CD) combined with various optimizations. The baseline setting is **w/o Early Stop**. Values in parenthesis/subscript denote the improvement brought by CD.

Method	TPS	TPF	Score
<i>LLaDA-8B-Instruct</i>			
Standard Inference	7.95	1.00	60.12
+ <i>CreditDecoding</i>	34.90 _{+339%}	15.39 _{+1439%}	60.73 _{+0.61}
Fast-dLLM (w/o KV)	28.90	12.64	60.11
+ <i>CreditDecoding</i>	34.90 _{+21%}	15.39 _{+22%}	60.73 _{+0.62}
Fast-dLLM (w/ KV)	39.38	4.42	58.51
+ <i>CreditDecoding</i>	51.40 _{+31%}	14.00 _{+217%}	58.63 _{+0.12}
Early Stop	9.70	1.00	59.94
+ <i>CreditDecoding</i>	37.33 _{+285%}	6.98 _{+598%}	60.75 _{+0.81}
PyTorch Compiler	9.03	1.00	60.26
+ <i>CreditDecoding</i>	39.51 _{+337%}	15.41 _{+1441%}	60.43 _{+0.17}
<i>LLaDA-MoE-Instruct</i>			
Standard Inference	3.53	1.00	62.73
+ <i>CreditDecoding</i>	15.26 _{+333%}	14.80 _{+1380%}	62.97 _{+0.24}
Fast-dLLM (w/o KV)	13.30	13.08	62.74
+ <i>CreditDecoding</i>	15.26 _{+15%}	14.80 _{+13%}	62.97 _{+0.23}
FP8 Quantization	2.72	1.00	62.47
+ <i>CreditDecoding</i>	11.87 _{+336%}	14.45 _{+1345%}	62.58 _{+0.11}
Early Stop	5.11	1.00	62.66
+ <i>CreditDecoding</i>	16.99 _{+233%}	4.77 _{+377%}	62.92 _{+0.26}
PyTorch Compiler	2.42	1.00	63.29
+ <i>CreditDecoding</i>	10.72 _{+343%}	14.88 _{+1388%}	62.99 _{-0.3}

generation and improving decoding efficiency.

Fast-dLLM (Wu et al., 2025): A state-of-the-art acceleration method consisting of threshold-based parallel decoding and KV cache. Since the KV cache significantly increases TPS at the cost of performance, we mainly compare with Fast-dLLM (w/o KV).

PyTorch Compiler (Jain et al., 2023): PyTorch Compiler leverages graph-level optimizations for runtime acceleration without altering decoding behavior. For MoE architectures, the compiler can fuse MoE kernels, which significantly accelerates the inference process.

FP8 Quantization (Kwon et al., 2023): FP8 quantization is a technique that reduces the precision of floating-point numbers to 8-bit, aiming to accelerate deep learning models by lowering storage and computation costs while maintaining sufficient accuracy.

Table 5 presents detailed results corresponding to Figure 1, including TPS comparisons and reports results under the same settings on LLaDA-MoE, further including FP8 quantization.

Table 6: Main benchmark results w/ **Early Stop** across eight datasets on **LLaDA-8B-Instruct** (Gen Length=256, Block Size=64). Cells show **Score** (top, relative to LLaDA) and **TPF** (bottom, improvement over Fast-dLLM).

Benchmark	$\frac{\text{Score}}{\text{TPF}}$	Fast-dLLM	CD	CD [†]
MMLU	62.43 _{-0.03}	2.86	63.78 _{+1.32}	63.66 _{+1.20}
SQuAD2.0	91.43 _{+0.00}	13.55	91.71 _{+0.28}	91.48 _{+0.05}
DROP	82.74 _{-0.12}	2.93	82.78 _{-0.08}	82.70 _{-0.16}
KorBench	33.20 _{+0.08}	3.72	35.04 _{+1.92}	33.92 _{+0.80}
HumanEval	34.15 _{-0.61}	3.82	36.59 _{+1.83}	37.80 _{+3.04}
LCB	8.15 _{+0.00}	1.93	7.54 _{-0.61}	7.71 _{-0.44}
GSM8K	78.47 _{+0.53}	3.22	77.18 _{-0.76}	77.48 _{-0.46}
MATH	37.04 _{-0.26}	2.42	37.24 _{-0.06}	37.18 _{-0.12}
Average	53.45 _{-0.05}	4.31	53.98 _{+0.48}	53.99 _{+0.49}

C.8 Full-Distribution Trace Credit

In Sec. 4.2, for each position, trace credit is accumulated only on the top-1 candidate token at each step. This focused strategy concentrates reinforcement and yields stronger acceleration, but it may ignore useful probability mass on other plausible candidates.

To examine the generality of credit accumulation beyond the top-1 hypothesis, we also consider a full-distribution variant that updates all tokens:

$$C_t^{i,v} = \beta C_{t+1}^{i,v} + (p_t^{i,v})^\gamma, \quad \forall v \in \mathcal{V}. \quad (13)$$

By aggregating historical evidence over the entire predictive distribution, this variant becomes less sensitive to transient noise and confidence oscillation. Intuitively, it can achieve slightly higher accuracy but weaker acceleration compared to the standard version, forming a more conservative trade-off option.

We evaluate this full-distribution variant (denoted as **CD[†]**) alongside the standard CreditDecoding (**CD**). Tables 6 and 7 report results on LLaDA-8B-Instruct and LLaDA-MoE-Instruct across eight benchmarks.

Overall, **CD[†]** delivers slightly larger accuracy improvements than **CD** (+0.49 vs. +0.48 on LLaDA-8B and +0.35 vs. +0.15 on LLaDA-MoE),

Table 7: Main benchmark results w/ **Early Stop** across eight datasets on **LLaDA-MoE-Instruct**. (Gen Length=256, Block Size=64). Cells show **Score** (top, relative to LLaDA-MoE) and **TPF** (bottom, improvement over Fast-dLLM).

Benchmark	Score _{TPF}	Fast-dLLM	CD	CD [†]
MMLU	64.08 _{2.16} ^{+0.00}	64.21 _{2.46} ^{+0.13}	63.94 _{2.33} ^{-0.14}	
SQuAD2.0	86.88 _{7.09} ^{+0.00}	87.27 _{9.64} ^{+0.39}	87.35 _{8.41} ^{+0.47}	
DROP	80.16 _{2.73} ^{+0.00}	79.72 _{3.28} ^{-0.44}	79.87 _{2.92} ^{-0.29}	
KorBench	36.88 _{2.36} ^{+0.16}	36.48 _{3.28} ^{-0.24}	36.64 _{2.73} ^{-0.08}	
HumanEval	51.22 _{4.97} ^{+0.00}	51.22 _{6.00} ^{+0.00}	53.05 _{5.45} ^{+1.83}	
LCB	14.04 _{2.43} ^{+0.16}	14.37 _{2.81} ^{+0.49}	14.65 _{2.55} ^{+0.77}	
GSM8K	74.45 _{2.28} ^{+0.08}	74.98 _{2.68} ^{+0.61}	74.37 _{2.42} ^{+0.00}	
MATH	35.84 _{2.35} ^{-0.18}	36.28 _{2.71} ^{+0.26}	36.26 _{2.48} ^{+0.24}	
Average	55.44 _{3.30} ^{+0.02}	55.57 _{4.11} ^{+0.15}	55.77 _{3.66} ^{+0.35}	

while incurring around 10% lower speedup due to its weaker reinforcement strength. Nevertheless, **CD[†]** still achieves substantial acceleration, showing that aggregating credit over the full predictive distribution remains effective relative to the baseline.

These results provide two important insights. First, the comparable (and sometimes higher) accuracy of **CD[†]** indicates that credit accumulation is fundamentally general and does not rely on reinforcing only the token that will be decoded. Second, **CD** achieves stronger acceleration, indicating that assigning credit only to the top-1 token is sufficient. Since the target tokens typically maintains the highest confidence for several consecutive steps, this *focused reinforcement* is both stable and computationally efficient.

Unlike **CD**, the full-distribution variant requires maintaining credits for every token in the vocabulary and updating them at each denoising step, which introduces non-trivial memory and computational overhead. Therefore, **CD** provides a better efficiency–accuracy trade-off and remains the recommended default in practice.

Table 8: Performance and TPF across different credit enhancement strategies on top- K candidate tokens and **CD[†]**, where top-1 denotes the default focus-enhancement mechanism applied in **CD**.

Enhancement Strategy	Score	TPF
Top-1 (CD)	53.98	5.47
Top-2	53.57	5.18
Top-3	53.96	5.06
Top-4	53.71	5.00
Top-5	53.63	4.95
Top- <i>all</i> (CD [†])	53.99	4.77

C.9 Robustness of the Focused Enhancement

For each denoising step, the focus enhancement mechanism strictly selects only the top-1 candidate token ranked by confidence to update the trace credit. To validate the robustness of this design choice, we evaluate a variant that updates trace credit for the top- K candidate tokens ranked by confidence at each position, rather than only the top-1 token.

First, we investigate incorporating broader information into the credit matrix. As detailed in Appendix C.8, we test a "Full-Distribution Trace Credit" approach, which records probabilities across all token positions regardless of their confidence levels. Furthermore, we conducted experiments using a *top- K enhancement strategy*, varying K from 1 to 5. The results, along with the performance of our extended variant **CD[†]**, are summarized in Table 8. Based on these results, we outline the following key observations:

- Incorporating more tokens into the trace credit consistently degrades TPF. This occurs because the accumulated credit distribution becomes less "peaky," resulting in a more conservative denoising process.
- The score does not exhibit a reliable upward trend as K increases. We hypothesize that increasing K introduces noisy or detrimental tokens that interfere with the denoising trajectory.
- Interestingly, as shown in Appendix C.8, the **CD[†]** variant achieves a slight performance gain (+0.2) on the LLaDA-MoE model. This suggests that tokens beyond the top-1 can theoretically assist denoising, provided they are

rigorously filtered. Considering this alongside our second observation, we conclude that selecting tokens based simply on confidence rank may not be an effective strategy.

Overall, updating the trace credit exclusively with the top-1 token remains the optimal configuration to balance inference efficiency and accuracy. Incorporating additional tokens sacrifices speed without guaranteeing stable performance improvements.

C.10 End-to-End Efficiency

To supplement the theoretical TPF metrics presented in the main text, we provide the detailed end-to-end inference speed, measured in Tokens Per Second (TPS), for all eight benchmarks on LLaDA-8B-Instruct. The evaluation is conducted under the same settings as Table 1.

Table 9: TPS comparison on **LLaDA-8B-Instruct** (Gen Length=256, Block Size=64). For our method, the cells show the absolute **TPS**, along with the speedup multipliers over the Baseline (superscript) and Fast-dLLM (subscript).

Benchmark	Baseline	Fast-dLLM	CD ^{vs. Baseline} _{vs. Fast-dLLM}
MMLU	4.95	12.49	17.30 ^{+249%} _{+38.4%}
SQuAD2.0	1.61	10.57	12.94 ^{+706%} _{+22.4%}
DROP	5.61	15.02	18.95 ^{+238%} _{+26.2%}
KorBench	6.33	12.40	14.56 ^{+130%} _{+17.4%}
HumanEval	16.04	59.25	67.69 ^{+322%} _{+14.2%}
LCB	4.55	8.58	9.54 ^{+110%} _{+11.2%}
GSM8K	18.84	58.37	69.95 ^{+271%} _{+19.8%}
MATH	18.20	43.35	50.81 ^{+179%} _{+17.2%}
Average	9.52	27.50	32.72 ^{+244%} _{+19.0%}

As shown in Table 9, CreditDecoding consistently improves TPS across all benchmarks, achieving an average gain of +19.0% over Fast-dLLM and +244% over the baseline. These TPS results further demonstrate that the efficiency improvements indicated by TPF translate directly into significant end-to-end speedups, confirming the practical efficacy of CreditDecoding in real-world deployments.

C.11 Failure Mode Analysis

While CreditDecoding maintains or improves performance on datasets with high parallel decoding potential, we observe slight performance drops on a few reasoning-intensive tasks, as shown in Table 1, such as mathematical reasoning and complex code generation. These datasets exhibit lower TPF and require longer denoising trajectories. In contrast, all datasets with $TPF > 4$ maintain or improve performance. We attribute this to the following factors.

Strong Causal Dependency. As illustrated in Figure 13a, the predictive distribution for later tokens in these tasks is highly dependent on preceding ones, resulting in high entropy. This forces the model to behave in a strictly autoregressive-like manner during denoising. Consequently, when the context is not yet fully formed—especially under aggressive parallel decoding—prediction errors in earlier steps are easily propagated and amplified for later tokens.

Unstable Confidence Convergence. As shown in Figure 4, tokens requiring a larger number of total denoising steps (the yellow curve) exhibit significantly less stable confidence convergence compared to those with higher TPF (the blue curve). When the temporal consistency of confidence weakens, the credit accumulation process struggles to effectively distinguish true convergence signals from noise under fixed hyperparameter configurations.

Nevertheless, we find that this vulnerability is fundamentally tied to the intrinsic capability of the base model. As shown in Table 1, these performance drops are significantly mitigated on the stronger model *LLaDA-MoE*. This suggests that more capable models exhibit stronger and more stable temporal consistency during the denoising trajectory, making the trace credit accumulation process inherently more robust to complex reasoning tasks.

Furthermore, the hyperparameters in Table 1 were selected to maximize average score across all eight datasets. While this may lead to suboptimal performance on certain reasoning-heavy edge cases, it establishes a robust baseline for general dLLM acceleration. Future work could explore dynamic or task-adaptive hyperparameter scheduling to further bridge the gap in these specialized domains.