FILL-AND-REFINE DECODING FOR DIFFUSION LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion large language models (dLLMs) have emerged as a promising alternative to autoregressive models. However, the vanilla decoding strategy suffers from a critical limitation: once an incorrect token is accepted, it cannot be revised, causing error propagation. Existing approaches perform refinement within the fill-up process, which limits their ability to exploit posterior information. In light of this, we propose FIRE, a two-stage decoding strategy consisting of a sequence fill-up stage followed by iterative refinement, where tokens are remasked and re-decoded conditioned on the remaining context. This mechanism allows previously accepted tokens to be reconsidered and corrected. Experiments on five benchmarks spanning language understanding, code generation, and mathematics demonstrate consistent improvements under identical computational budgets. These results highlight the importance of decoding algorithms in fully unlocking the potential of diffusion language models.[1]

1 INTRODUCTION

Large language models (LLMs) (Chowdhery et al., 2022; Hurst et al., 2024; Comanici et al., 2025) have driven remarkable progress across diverse NLP domains (Zhao et al., 2023; Minaee et al., 2024). The dominant architecture behind these advances is the autoregressive (AR) transformer (Vaswani et al., 2017). While highly effective, AR decoding is inherently sequential, which limits generation parallelism (Fu et al., 2024; Xia et al., 2024).

To address this, diffusion language models (Austin et al., 2021a; Li et al., 2022) have emerged as a powerful alternative, generating sequences through iterative denoising with bidirectional attention and parallel token predictions. This paradigm offers distinct advantages over AR models (Li et al., 2025b), including accelerated inference, stronger global coherence, and controllable quality–speed trade-offs. Recent progress (Labs et al., 2025; Nie et al., 2025; Ye et al., 2025b) has further demonstrated the practicality and competitiveness of discrete diffusion large language models (dLLMs). Commercial dLLMs such as Mercury Coder (Labs et al., 2025) and Gemini Diffusion (Google DeepMind, 2025) claim to match the performance of autoregressive LLMs (Hurst et al., 2024; Team et al., 2024) while achieving up to $10\times$ faster inference on tasks like code generation (Chen et al., 2021; Austin et al., 2021b).

Despite recent advances, current dLLM decoding strategies (Israel et al., 2025; Yu et al., 2025; Wu et al., 2025) suffer from a critical limitation: once a token is accepted, it is typically fixed and cannot be modified in subsequent steps (Wang et al., 2025a; von Rütte et al., 2025). For instance, in two widely adopted open-source dLLMs, LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025b), a token is considered *accepted* if, at a specific iteration, it is unmasked and no longer remasked. Once accepted, it will serve as fixed context for all future predictions. This causes early mistakes to persist and propagate throughout the generation process (Wang et al., 2025a).

There have been some early explorations on this issue. ReMDM (Wang et al., 2025a) introduces a sampler that applies a stochastic backward remasking process for predicted tokens. RCR (He et al., 2025) tracks each token's running max confidence and remasks persistently low-confidence tokens. GIDD (von Rütte et al., 2025) trains diffusion models with a mixing schedule that interpolates between data and noise distributions to enable the remasking of predicted tokens. WINO (Hong et al.,

[1] Code and data are anonymously available.

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

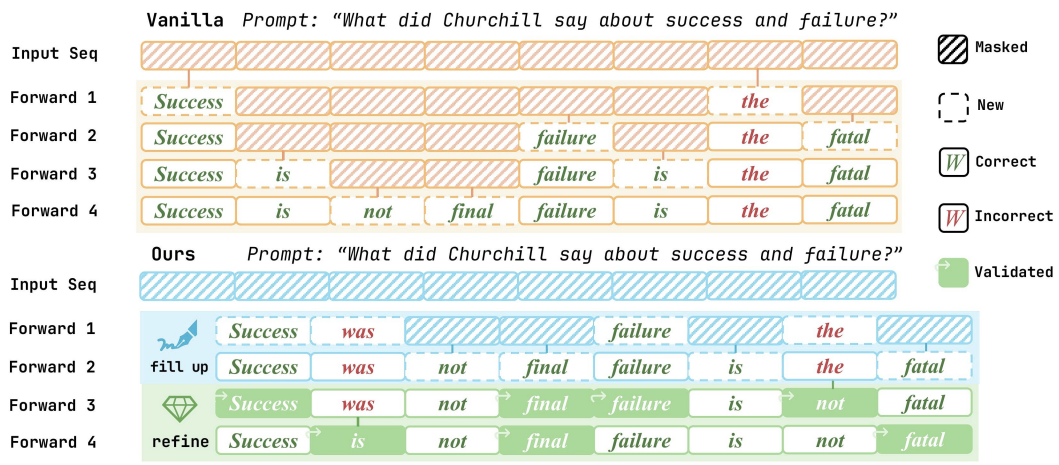


Figure 1: **Overview of FIRE.** Compared to the vanilla decoding strategy, we first fill the masked tokens with high parallelism and then iteratively refine the draft through token-level cross-validation. Here, cross-validation means tokens alternately act as the *target* and the *context* of prediction. This process allows previously accepted tokens to be revisited and corrected when necessary.

2025) proposes to draft and verify masked tokens with different thresholds and shadow all-mask context. While these works demonstrate the significance of dLLM error correction strategies, their refinement is restricted within the fill-up process. Furthermore, their improvements have not achieved ideal performance on general tasks, so the challenge remains an open problem.

To bridge this gap, we propose FIRE (**Fi**ll-and-**Re**fine), a test-time dLLM decoding method that explicitly separates generation into *two stages*: **fill-up** and **refinement**. As illustrated in Figure 1, in the first stage, we fill up the masked tokens following the vanilla dLLM decoding strategy. In the second stage, we iteratively refine this draft by remasking and decoding subsets of tokens while using the remaining as context, so that predictions are revised by cross-validating against one another. Our approach differs from existing strategies which perform refinement *within* the ongoing generation process. By explicitly decoupling fill-up and refinement into two separate phases, FIRE enables a more explicit and thorough form of token-level error correction than prior methods.

We evaluate FIRE on five standard benchmarks across language understanding (TriviaQA (Joshi et al., 2017), GPQA (Rein et al., 2024)), code generation (MBPP (Austin et al., 2021b), HumanEval (Chen et al., 2021)), and mathematics (GSM8K (Cobbe et al., 2021)). We use vanilla decoding, ReMDM (Wang et al., 2025a), and RCR (He et al., 2025) as baselines. Experimental results show that, under the same computational budget, FIRE achieves noticeable and consistent improvements over the baselines. Overall, our results suggest that decoupling dLLM generation into a draft stage followed by refinement can be an effective strategy for producing higher-quality sequences with a low compute budget.

2 Related Work

We review two threads of work most closely related: training and inference for dLLMs, and error correction during diffusion decoding.

2.1 Training and Inference of dLLMs

Substantial effort has focused on improving how dLLMs are trained and deployed. On the training side, large-scale instruction tuning (Ye et al., 2025b; Nie et al., 2025) has demonstrated that diffusion models can achieve general capabilities comparable to autoregressive LLMs. Researchers also explore refinements of the training objective: simplified masked losses (Shi et al., 2024; Sahoo et al., 2024), likelihood-based formulations (Gulrajani & Hashimoto, 2023), and variants that enhance generation robustness and reasoning (von Rütte et al., 2025; Ye et al., 2025a). Another direction focuses on

adapting reinforcement learning to diffusion, either to strengthen reasoning (Huang & Tang, 2025; Ye et al., 2024; Zhao et al., 2025; Wang et al., 2025b) or for preference optimization (Zhu et al., 2025).

Decoding is another key bottleneck for diffusion language models: parallel generation improves efficiency but often degrades quality. Adaptive Parallel Decoding (APD) (Israel et al., 2025) mitigates this trade-off by adjusting the degree of parallelism with an auxiliary autoregressive verifier, while dilated (Luxembourg et al., 2025) scheduling further accelerates inference. At the same time, KV-caching (Ma et al., 2025; Wu et al., 2025) and autoregressive-guided unmasking (Hu et al., 2025) are applied to further accelerate dLLMs. Recent works (Li et al., 2025a; Kim et al., 2025) also address flexibility by extending diffusion to variable-length and token insertion.

2.2 Error Correction in Diffusion Decoding

It is often claimed that vanilla discrete diffusion language models possess an inherent ability for error correction, since each position is repeatedly predicted as the context evolves over iterations (Wang et al., 2025a; Google DeepMind, 2025). However, this view is incomplete: once a token is *accepted*, it becomes fixed and cannot be revised. For example, LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025b) decide at every iteration whether a token should be further remasked; if it is not, the token is considered accepted and remains unchanged thereafter. As a result, any early mistake will persist and propagate through subsequent steps, limiting the reliability of diffusion generation.

Several methods have sought to address this limitation. ReMDM (Wang et al., 2025a) introduces a probabilistic remasking process that allows already revealed tokens to be re-predicted. RCR (He et al., 2025) proposes a simple confidence-based strategy that remasks uncertain tokens during inference. GIDD (von Rütte et al., 2025) modifies the corruption process with hybrid noise during training. WINO (Hong et al., 2025) proposes a decoding scheme that drafts tokens under a lenient threshold, then verifies them with a stricter threshold using a shadow all-mask context to revoke errors.

While these approaches demonstrate the feasibility of token revision or refinement, their refinement is limited within the fill-up process, which restricts their ability to exploit posterior information that only becomes available after more tokens are filled in. In contrast, our approach departs from prior work by explicitly *decoupling fill-up and refinement*. We first generate a draft following vanilla diffusion decoding, and then apply a targeted refinement stage that revisits the accepted tokens according to a cross-validation principle. This separation not only makes error correction conceptually more systematic but also delivers markedly stronger empirical gains.

3 Methodology

3.1 Preliminaries

Decoding in Discrete dLLMs. We consider the decoding process of discrete diffusion large language models (Ye et al., 2025b; Nie et al., 2025). Specifically, let $x_i^{(t)} \in \mathcal{V}$ denote the token at position $i \in \{1, \dots, L\}$ and time step $t \in \{0, \dots, T\}$, where $\mathcal{V}$ is the vocabulary, L is the sequence length, and T is the total number of forward steps. At inference time, the sequence is initialized with

$$x^{(0)} = (c_1, \dots, c_m, \underbrace{\texttt{[MASK]}_{m+1}, \dots, \texttt{[MASK]}_L}_{L-m}) \in \mathcal{V}^L,$$

where c_1 to c_m are prompt tokens and the remaining $L - m$ positions are masked tokens. At each time step, discrete diffusion large language models output the logits of all masked tokens and decode them by sampling, where $y_i^{(t)} \sim p_\theta(\cdot \mid x^{(t)}, t)$, and p_θ is the conditional distribution parameterized by the dLLMs. A deterministic rule then decides whether to accept or remask each decoded token. Specifically, the next sequence is constructed as

$$x_i^{(t+1)} = \begin{cases} y_i^{(t)}, & \text{accepted}, \\ \texttt{[MASK]}, & \text{remasked}, \end{cases} \quad \text{for } i \notin \mathcal{I}_t,$$

$$x_j^{(t+1)} = x_j^{(t)} \quad \text{for } j \in \mathcal{I}_t.$$

where $\mathcal{I}_t \subseteq \{1, \dots, L\}$ is the index set of tokens already accepted at step t. In the vanilla setup, each step accepts approximately $\lfloor L/T \rfloor$ tokens, which are selected based on criteria like model

confidence or entropy. Different dLLMs may adopt alternative decoding strategies. For example, semi-autoregressive decoding (Nie et al., 2025) only proceeds to the next block once all tokens in the current block have been accepted. Our study focuses on the widely adopted setup that decodes a fixed number of tokens with the lowest entropy.

Limitations of Conventional Decoding. In the conventional decoding setup, masked positions are iteratively unmasked and decoded, while accepted tokens become fixed and remain unchanged. Formally, once a position index i enters the visible set $\mathcal{I}_t$, we have $i \in \mathcal{I}_{t'}$ and $x_i^{(t')} = x_i^{(t)}$ for all $t' > t$. As a result, an early error at position $j \in \mathcal{I}_t$ is permanently preserved and enters the context for all future predictions $p_\theta(x_i^{(t')} \mid x^{(t'-1)}, t'-1)$, where $i \notin \mathcal{I}_{t'-1}$. Such errors cannot be revised and may propagate through the decoding process as persistent noise, ultimately degrading the quality of the generated sequence.

3.2 Method Overview

To overcome this limitation, we propose FiRe (**Fi**ll-and-**Re**fine), which moves beyond the traditional view of decoding as a single, progressive unmasking trajectory, and instead reframes it as a two-stage process of **fill-up** and **refinement** to better utilize the posterior information in decoding.

Stage I (Sequence Fill-Up). In the fill-up stage, the diffusion language model produces a coarse draft by filling masked positions following the vanilla dLLM decoding strategy, providing a complete but imperfect hypothesis of the output.

Stage II (Cross-Validation Refinement). In the refinement stage, our iterative procedure follows a token-level cross-validation principle, where tokens alternately act as validator and as validation targets. This alternating role improves the overall consistency of the sequence.

3.3 Sequence Fill-Up

The sequence fill-up stage is largely based on the vanilla dLLM decoding procedure. However, to facilitate the refinement stage, we introduce one modification: the logit penalty on the End-of-Text (EoT) token.

EoT Penalty. Since the refinement stage can correct errors, we prefer longer and more informative drafts to overly short completions. To this end, we apply an *EoT penalty* (Bai et al., 2021; Laban et al., 2020), which discourages the generation of EoT tokens in the fill-up stage. Concretely, we penalize the logit of the EoT token by a factor $\lambda_{\text{eot}} > 1$ before softmax, using the sign-aware form of the standard repetition penalty (Keskar et al., 2019). While this adjustment does not directly improve draft quality, it effectively prevents early termination and produces drafts that are better suited for subsequent refinement. Formally, let z_v be the unnormalized logit for token v at position i and time step t. The penalized distribution is

$$\tilde{p}_\theta(v \mid x^{(t)}, t) \propto \begin{cases} \exp(z_v/\lambda_{\text{eot}}), & \text{if } v = \texttt{[EoT]},\ z_v \geq 0 \\ \exp(\lambda_{\text{eot}} z_v), & \text{if } v = \texttt{[EoT]},\ z_v < 0 \\ \exp(z_v), & \text{otherwise.} \end{cases}$$

Finally, the fill-up stage produces a sequence consisting of the prompt tokens and model predictions for previously masked positions:

$$x^{(\rho T)} = \left(c_1, \ldots, c_m,\ x_{m+1}^{(\rho T)}, \ldots, x_L^{(\rho T)}\right) \in \mathcal{V}^L,$$

where $x_i^{(\rho T)} \neq \texttt{[MASK]}$ for all $i > m$. Here $\rho \in (0, 1)$ controls the split between the two stages.

3.4 Cross-Validation Refinement

The refinement stage corrects errors in the draft with a token-level cross-validation principle, where tokens alternately act as validator and as validation targets. In each iteration, a different subset of tokens is *sampled*, *remasked* and *decoded* conditioned on the preserved context, progressively reducing mistakes and improving coherence.

Iterative Refinement. At each iteration k, we remask a random subset $S^{(k)} \subseteq \{m+1, \ldots, L\}$ of non-prompt positions, sampled at rate γ_k so that $|S^{(k)}| = \lfloor \gamma_k (L-m) \rfloor$.

$$x_i^{(k)} = \begin{cases} \texttt{[MASK]}, & i \in S^{(k)} \\ x_i^{(k)}, & \text{otherwise.} \end{cases}$$

The sequence for the next iteration is then obtained by predicting the masked tokens:

$$x_i^{(k+1)} = \begin{cases} y_i^{(k)}, & i \in S^{(k)} \\ x_i^{(k)}, & \text{otherwise,} \end{cases}$$

where $y_i^{(k)} \sim p_\theta(\cdot \mid x^{(k)}, k)$.

In each iteration, a subset of generated tokens is held fixed as context, while the remaining tokens are remasked and decoded to better align with the fixed context tokens. Iterating this process gradually improves the coherence of the entire sequence.

Annealed Refinement Rate. To improve the stability of refinement steps, we anneal the refinement rate γ_k over time. Higher refinement rates in early iterations encourage broader corrections of initial errors, while lower rates in later iterations help stabilize the predictions. We adopt a cosine annealing schedule with both upper and lower bounds:

$$\gamma_k = \gamma_{\min} + \tfrac{1}{2}(\gamma_{\max} - \gamma_{\min}) \left(1 + \cos\left(\tfrac{\pi k}{K}\right)\right),$$

where k is the current refinement iteration and K is the total number of refinement steps.

4 Experiment

4.1 Experimental Setup

We describe the backbones, benchmarks and metrics, baselines, and evaluation configurations used in our experiments. Unless otherwise stated, all methods are compared under the same prompting and the same total forward step budget.

Models. Following previous studies (Ma et al., 2025; Israel et al., 2025; Wu et al., 2025; He et al., 2025), we evaluate our method on the instruction-tuned variants of two widely used open-source discrete dLLMs: **Dream-v0-Instruct-7B** (Ye et al., 2025b) and **LLaDA-8B-Instruct** (Nie et al., 2025). Both are state-of-the-art representatives of open-source discrete dLLMs.

Datasets & metrics. To assess the general effectiveness of our method, we evaluate it on three representative tasks with five standard benchmarks: (i) language understanding with **TriviaQA** (Joshi et al., 2017) and **GPQA** (Rein et al., 2024), (ii) code generation with **HumanEval** (Chen et al., 2021) and **MBPP** (Austin et al., 2021b), and (iii) mathematical reasoning with **GSM8K** (Cobbe et al., 2021). We use pass@1 for HumanEval and MBPP, and accuracy otherwise.

Baselines. We compare our method against the vanilla decoding strategy and two training-free baselines that propose to revise the accepted tokens. (i) **Vanilla** follows the standard dLLM decoding procedure: once a token is accepted, it remains fixed throughout the generation process and cannot be revised. (ii) **ReMDM** (Wang et al., 2025a) introduces a stochastic sampler that applies a backward remasking process for predicted tokens. (iii) **RCR** (He et al., 2025) records each token's running max confidence and remasks persistently low-confidence tokens. We do not include GIDD (von Rütte et al., 2025) (training required) and WINO (Hong et al., 2025) (focusing on small block-size block-attention; the number of forward steps is not precisely controllable), as they are not directly applicable in our fixed-budget, training-free evaluation protocol.

Table 1: **Main Experiment Results (Dream)**. Performance of **Dream** across five standard benchmarks under different numbers of forward steps, with improvement (blue) and degradation (red) relative to vanilla performance. Highest values for each task are **bold**. ± denotes the standard deviation over three random seeds.

Model	Method	TriviaQA						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
Dream (0-shot)	Vanilla	23.08±0.01	23.22±0.02	23.16±0.03	23.24±0.02	23.51±0.01	28.08±0.03	29.32±0.03
	ReMDM	22.11±1.17	22.94±0.28	22.87±0.10	22.94±0.16	23.27±0.32	27.98±0.41	29.26±0.37
	RCR	15.63±0.23	14.53±0.12	15.02±0.13	17.68±0.13	18.92±0.27	26.81±0.34	36.64±0.42
	FIRE	**27.78**±0.29	**31.61**±0.11	**33.76**±0.19	**35.98**±0.16	**40.61**±0.16	**42.46**±0.22	**40.47**±0.16
Model	**Method**	**GPQA**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
Dream (5-shot)	Vanilla	26.35±0.64	30.50±0.39	31.90±0.52	31.02±1.18	33.70±0.88	32.40±0.76	32.90±0.91
	ReMDM	25.41±1.11	30.18±0.98	32.09±0.17	31.83±0.84	34.12±1.19	32.84±0.68	33.54±0.72
	RCR	24.80±0.25	28.42±0.16	31.53±0.14	31.42±0.33	33.81±0.29	32.61±0.38	33.21±0.47
	FIRE	**32.80**±1.12	**32.30**±1.04	**33.40**±0.18	**35.70**±1.38	**35.40**±1.15	**34.30**±1.82	**34.80**±1.19
Model	**Method**	**HumanEval**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
Dream (0-shot)	Vanilla	**8.13**±0.35	13.41±0.00	11.79±0.35	12.80±0.61	26.02±0.35	37.80±0.61	**50.61**±0.00
	ReMDM	2.03±0.70	9.35±0.35	12.20±0.00	13.21±0.35	27.03±0.70	38.82±0.70	50.20±0.93
	RCR	1.22±0.24	8.54±0.31	8.54±0.31	22.56±0.45	30.49±0.37	26.22±0.28	26.22±0.28
	FIRE	4.88±1.06	**17.89**±1.27	**27.03**±2.54	**30.89**±1.37	**33.03**±2.21	**40.24**±0.81	47.56±0.61
Model	**Method**	**MBPP**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
Dream (4-shot)	Vanilla	**14.40**±0.20	14.73±0.12	17.00±0.20	25.00±0.40	31.07±0.12	45.13±0.31	**56.93**±0.83
	ReMDM	8.80±0.53	14.67±0.31	15.93±0.12	26.00±0.20	33.13±0.70	45.27±0.64	56.60±0.35
	RCR	4.80±0.84	10.40±0.71	23.60±0.55	29.00±0.29	36.00±0.36	42.60±0.43	41.73±0.12
	FIRE	10.53±1.01	**25.13**±0.64	**35.00**±0.80	**41.07**±2.91	**44.40**±1.20	**48.47**±1.55	51.53±0.76
Model	**Method**	**GSM8K**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
Dream (0-shot)	Vanilla	9.22±0.46	23.07±0.04	26.79±0.12	35.36±0.09	50.11±0.00	65.38±0.12	**73.10**±0.06
	ReMDM	11.02±0.64	18.12±0.59	27.98±0.00	35.91±0.09	47.81±0.09	61.66±0.16	70.00±0.44
	RCR	3.79±0.25	6.90±0.27	21.46±0.33	42.00±0.41	48.90±0.38	46.55±0.29	36.47±0.35
	FIRE	**14.40**±0.59	**24.92**±1.22	**35.96**±1.45	**47.66**±0.18	**62.80**±0.90	**68.99**±0.92	72.61±0.46

Configurations. For fairness, all methods are evaluated with the same dLLM backbones in the few-shot scenario. We align our few-shot setting with LLaDA (Nie et al., 2025) and Dream (Ye et al., 2025b), using task-specific numbers (see Table 1 and 2) of in-context examples. We equalize the computational cost between baselines and our method by allocating the same total forward step budget. Specifically, for our method, we set the allocation ratio ρ between sequence fill-up and refinement to 0.5. However, it is worth noting that our method itself has no restriction on how steps are allocated. This constraint is introduced solely to ensure a fair comparison.

For our method, we adopt a cosine annealing scheduler for the refinement rate with $\gamma_{\max} = 0.8$ and $\gamma_{\min} = 0.4$, and increase the EoT penalty λ_{eot} from 1.0 to 1.3 as the number of forward steps T grows. For baselines, we use the grid-searched hyperparameters for ReMDM ($t_{\text{on}} = 0.60$, $t_{\text{off}} = 0.40$, $\alpha_{\text{on}} = 0.95$; Appendix G) and use the linear remasking scheduling function for RCR, which is reported to be optimal (He et al., 2025).

We follow the default prompts from OpenCompass (Contributors, 2023) (GPQA, HumanEval for LLaDA) and LM-Eval (Gao et al., 2024) (all the remaining) evaluation framework and fix the generation length L at 256, except for HumanEval where we set L =512 to allow complete code generation. We vary the total number of forward steps T over $\{4, 8, 16, 32, 64, 128, 256\}$, covering the scenarios from highly parallel to fully sequential decoding. All experiments are run on 8 NVIDIA H200 GPUs, and each data point is evaluated with three random seeds to ensure statistical significance.

Table 2: **Main Experiment Results (LLaDA)**. Performance of **LLaDA** across five standard benchmarks under different numbers of forward steps, with improvement (blue) and degradation (red) relative to vanilla performance. Highest values for each task are **bold**. ± denotes the standard deviation over three random seeds.

Model	Method	TriviaQA						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
LLaDA (0-shot)	Vanilla	0.19±0.02	0.65±0.03	2.13±0.01	4.63±0.02	9.36±0.06	16.25±0.02	22.76±0.01
	ReMDM	0.25±0.02	0.43±0.01	1.08±0.02	1.82±0.03	3.05±0.06	5.43±0.03	8.24±0.02
	RCR	0.09±0.01	0.80±0.01	**4.44**±0.01	**8.62**±0.01	**16.08**±0.02	**24.04**±0.01	**29.30**±0.01
	FIRE	**0.99**±0.01	**1.86**±0.08	3.52±0.09	6.19±0.06	10.94±0.10	16.46±0.09	19.72±0.14
Model	**Method**	**GPQA**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
LLaDA (5-shot)	Vanilla	28.90±1.39	29.20±1.58	27.78±1.58	31.84±1.67	31.31±0.92	26.26±0.52	29.80±0.46
	ReMDM	26.50±0.92	29.90±0.15	26.60±1.04	31.33±0.97	32.33±0.71	29.90±1.12	29.60±0.11
	RCR	28.40±0.42	28.10±0.16	27.50±0.11	30.23±0.25	30.23±0.17	28.83±0.15	28.00±0.14
	FIRE	**30.30**±1.52	**32.80**±1.38	**28.90**±1.51	**32.83**±1.24	**33.83**±1.62	**31.31**±1.96	**30.81**±1.41
Model	**Method**	**HumanEval**						
		#F=8	#F=16	#F=32	#F=64	#F=128	#F=256	#F=512
LLaDA (0-shot)	Vanilla	0.90±0.32	1.22±0.38	4.27±1.15	14.02±1.31	26.22±1.48	34.15±2.42	**46.17**±0.33
	ReMDM	0.42±1.18	0.92±0.12	4.30±1.34	14.50±1.19	27.13±3.12	35.21±0.76	43.01±3.15
	RCR	0.21±0.31	0.84±0.97	4.42±0.38	14.30±0.32	28.32±0.39	35.11±0.81	44.22±0.28
	FIRE	**1.00**±1.47	**1.83**±0.39	**6.10**±1.32	**15.85**±0.98	**31.10**±0.82	**37.80**±0.73	45.12±5.84
Model	**Method**	**MBPP**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
LLaDA (4-shot)	Vanilla	**9.53**±0.81	14.40±0.40	13.40±0.69	17.73±0.12	24.07±0.64	31.27±0.81	37.87±0.61
	ReMDM	1.53±0.31	2.33±0.58	4.53±0.42	10.53±0.12	17.27±0.64	23.33±1.72	35.47±0.90
	RCR	0.60±0.40	3.47±0.46	10.00±0.20	13.33±0.12	15.93±0.46	22.27±0.31	26.73±0.12
	FIRE	5.53±1.03	**16.00**±0.87	**22.73**±1.03	**25.60**±0.69	**29.27**±0.42	**33.87**±0.81	**38.53**±1.50
Model	**Method**	**GSM8K**						
		#F=4	#F=8	#F=16	#F=32	#F=64	#F=128	#F=256
LLaDA (4-shot)	Vanilla	10.24±0.21	19.94±0.32	34.87±0.64	46.78±0.48	50.11±0.55	54.42±0.79	69.81±0.61
	ReMDM	12.00±0.23	20.34±1.02	35.63±0.46	45.23±0.58	55.26±0.28	56.82±0.19	69.22±0.87
	RCR	11.23±0.29	17.53±0.76	30.23±0.31	47.30±0.51	52.32±0.69	54.16±0.35	68.39±0.92
	FIRE	**17.70**±0.47	**32.33**±0.84	**37.30**±1.12	**53.75**±1.21	**65.81**±0.27	**72.02**±0.59	**72.78**±1.11

4.2 MAIN RESULTS

Table 1 and Table 2 summarize the main results of FIRE and baselines under matched forward step budgets across five benchmarks. Overall, FIRE improves over vanilla decoding significantly in the highly-parallel setting (e.g., on Dream at #F=8, TriviaQA increases from 23.22 to 31.61, surpassing vanilla at #F=256), and remains competitive as $\#F$ increases (e.g., on LLaDA at #F=256, GSM8K improves from 69.81 to 72.78). This supports our motivation that errors committed early in parallel decoding tend to persist, and that a dedicated refinement phase can effectively correct them.

Effect of $\#F$. Across tasks and backbones, the benefit of FIRE is most pronounced when $\#F$ is small, where each forward step updates many tokens and early mistakes are more likely to be accepted and propagated. As $\#F$ grows and decoding approaches the fully sequential setting, vanilla performance typically increases and the marginal room for improvement shrinks; correspondingly, FIRE's gains narrow and in a few settings become comparable to strong baselines.

Task-level observations. On mathematics (**GSM8K**), FIRE delivers strong and stable improvements over vanilla across $\#F$, with large gains in most of cases (e.g., from 26.79 to 35.96 at $\#F$=16). On language understanding (**TriviaQA** and **GPQA**), FIRE often yields large absolute gains at small and moderate $\#F$ (e.g., TriviaQA from 23.24 to 35.98 at $\#F$=32). On code genera-

Table 3: **Performance under different fill-up steps with and without cross-validation refinement.** We sweep the number of fill-up forward steps from 16 to 128 and compare vanilla decoding (#R=0) with FIRE using a fixed refinement budget (#R=16). Results are reported for both Dream and LLaDA on TriviaQA and GSM8K.

Model	Task	Fill-up Steps=16		Fill-up Steps=32		Fill-up Steps=64		Fill-up Steps=128	
		#R=0	#R=16	#R=0	#R=16	#R=0	#R=16	#R=0	#R=16
Dream	TriviaQA	23.16	$35.98_{+12.82}$	23.24	$35.88_{+12.64}$	23.51	$35.93_{+12.42}$	28.08	$38.09_{+10.01}$
Dream	GSM8K	26.79	$48.77_{+21.98}$	32.75	$55.57_{+22.82}$	35.36	$67.32_{+31.96}$	65.38	$69.06_{+3.68}$
LLaDA	TriviaQA	2.13	$6.19_{+4.06}$	4.63	$5.64_{+1.01}$	9.36	$16.34_{+6.98}$	16.25	$18.90_{+2.65}$
LLaDA	GSM8K	34.87	$39.87_{+5.00}$	46.78	$51.58_{+4.80}$	50.11	$59.34_{+9.23}$	54.42	$67.54_{+13.12}$

tion (**HumanEval** and **MBPP**), FIRE improves performance most reliably in the small $\#F$ range (e.g., HumanEval from 12.80 to 30.89 at $\#F$=32).

Comparison with revision baselines. Among training-free methods that revise accepted tokens, ReMDM and RCR can help in selective settings, but their performance is less consistent across tasks and $\#F$. In contrast, FIRE provides the most reliable improvements in the scenarios where parallel decoding is most challenging, suggesting that explicitly decoupling *fill-up* and *refinement* enables more effective token-level correction than revising tokens within the ongoing generation process.

4.3 ABLATION STUDIES

To analyze the effect of different components in our decoding strategy, we conduct ablation studies on (i) token-level cross-validation refinement, (ii) EoT Penalty, and (iii) refinement rate annealing.

Cross-Validation Refinement. We examine cross-validation refinement under a fixed refinement budget by varying the number of forward steps in the fill-up stage for parallel decoding. Specifically, we sweep the fill-up steps from 16 to 128 and compare two decoding settings for both Dream and LLaDA: vanilla decoding without refinement (#R=0) and FIRE with a fixed number of refinement steps (#R=16). The refinement procedure uses the same remask-and-decode strategy across all groups. Results on GSM8K and TriviaQA are reported in Table 3.

Across all evaluated fill-up step settings, enabling cross-validation refinement consistently improves performance over the no-refinement baseline for both models. The gains are particularly pronounced at smaller fill-up steps, where the initial draft is relatively coarse and early token errors are more likely to persist; adding a lightweight refinement phase provides effective token-level error correction and yields substantial accuracy improvements (e.g., on GSM8K with fill-up steps = 16, from 26.79 to 48.77 for Dream). As fill-up steps increase, the draft becomes stronger and the absolute gap narrows, yet refinement remains beneficial throughout the sweep (e.g., on TriviaQA at #F=128, from 22.17 to 28.5, aggregated over the two models). Overall, these results indicate that cross-validation refinement is a robust plug-in: even with a small and fixed refinement budget, it delivers consistent gains across a wide range of fill-up step counts.

EoT penalty. To isolate the impact of the EoT penalty, we fix the fill-up and refinement configurations and vary only the penalty coefficient λ_{eot}. Specifically, we vary λ_{eot} from 1.0 to 1.3 while keeping the number of forward steps T fixed at 32 and 128. We find that applying non-trivial λ_{eot} consistently improves generation quality, with notable gains at $\lambda_{\text{eot}} = 1.1$, 1.2, and 1.3 (+23.2%, +28.4%, +23.9% relatively). This is because the EoT penalty encourages longer fill-up sequence: although these drafts may not always be fully correct, they tend to contain more information overall. During refinement, the useful content can be preserved and amplified while the incorrect parts are likely to be corrected.

Refinement Rate Annealing. In cross-validation refinement, we found that the ratio between sampled predicted tokens serving as validators and those serving as validation targets (referred to as the *refinement rate* here) is critical. A rate that is too small makes it difficult to effectively revise validated tokens, while a rate that is too large limits the available context and lowers refinement quality. Thus, choosing an appropriate refinement rate is essential for stable performance. To achieve this

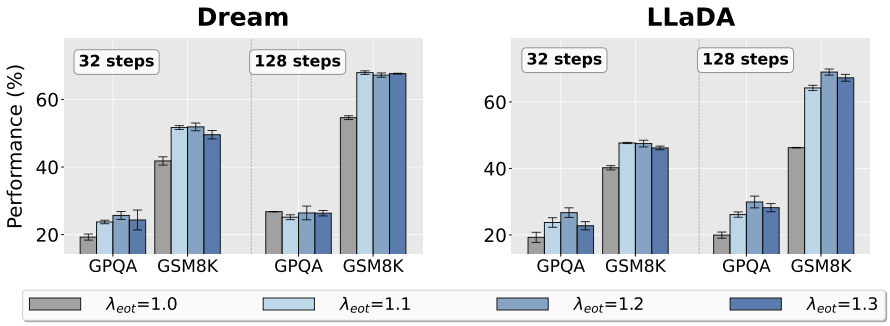


Figure 2: **Ablation Studies of EoT Penalty.** We fix the fill-up and refinement configurations while varying λ_{eot} from 1.0 to 1.3, with results shown for 32 and 128 forward steps T. Across most tasks, introducing an appropriate EoT penalty substantially improves generation quality. The precise numerical values are reported in Appendix A.

Table 4: **Effect of refinement rate annealing.** Annealing consistently improves accuracy on GSM8K and TriviaQA across forward step budgets.

Model / Task	**Anneal**	$\#F$				
		16	32	64	128	256
Dream / GSM8K	×	33.98	47.27	64.84	67.97	70.31
	✓	36.20	50.00	62.76	69.53	74.09
LLaDA / GSM8K	×	33.98	52.73	60.55	65.23	68.75
	✓	35.55	52.73	65.49	70.18	70.31
Dream / TriviaQA	×	17.19	19.92	19.53	21.88	23.44
	✓	19.14	20.70	26.17	29.30	24.22
LLaDA / TriviaQA	×	1.17	0.78	4.30	10.94	15.23
	✓	1.56	3.52	8.98	12.50	16.02

balance, we introduce cosine annealing of the refinement rate, which provides sufficient momentum in the early steps of refinement while ensuring stability in the later steps.

To evaluate the effectiveness of this design, we compare the performance of Dream and LLaDA on GSM8K and TriviaQA with and without refinement rate annealing across different numbers of forward steps (Table 4). For LLaDA, annealing consistently delivers an average improvement of +2.1 points across all step settings on GSM8K. For Dream, while an exception occurs at 64 steps, the overall effect is still an average improvement of +1.3 points on this dataset. On TriviaQA, refinement rate annealing remains effective across most forward steps, delivering average gains of +2.8 on LLaDA and +1.3 on Dream. These results demonstrate that our cosine annealing mechanism of refinement rate is an effective strategy.

Qualitative Cases. Furthermore, Appendix F presents several case studies that qualitatively illustrate the error correction process of our method.

5 Conclusion

We revisited a key limitation of discrete dLLMs: once accepted, tokens cannot be revised, causing early mistakes to propagate. We proposed FIRE, a training-free strategy that decouples decoding into fill-up and refinement, and uses token-level cross-validation to systematically correct errors.

Experiments across five benchmarks spanning language understanding, code, and math show consistent gains over baselines under matched forward step budgets, highlighting that decoding strategy is a crucial component of dLLM performance.

AI USE STATEMENT

In this work, we used generative AI tools to create and edit software code: Claude Code (Anthropic, 2025) served as a coding assistant to write and debug experimental scripts, and to convert the manuscript to the ICLR format. We also used GPT-5 (OpenAI, 2025) to edit the manuscript to improve readability. We have not used generative AI tools to propose or refine hypotheses, design research methodology or experiments, or interpret results. We reviewed all AI-assisted work: generated code was verified and tested by the authors, and all AI-edited text was checked and revised where necessary. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. It does not involve human subjects or the collection of new data. All datasets used (TriviaQA (Joshi et al., 2017), GPQA (Rein et al., 2024), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b)) are publicly available academic benchmarks without personally identifiable or sensitive information. Our method improves decoding in diffusion language models but does not remove the risks inherent to such models, which may generate incorrect or misleading content and insecure or faulty code; caution is warranted when deploying them in high-stakes scenarios.

REPRODUCIBILITY STATEMENT

Section 4.1 specifies the backbones, benchmarks, metrics, prompts, generation lengths, forward-step budgets, and hyperparameters of FIRE and the baselines. Algorithm 1 (Appendix B.1) gives the complete refinement procedure, and Appendix G documents the hyperparameter search for the baselines. Exact numerical results with standard deviations over three seeds are reported in Tables 1 and 2 and in Appendix A, and the settings of all additional studies are stated in Appendices B and C. Code and data are available through the anonymous link in the abstract.

REFERENCES

Anthropic. Claude code | claude. https://claude.com/product/claude-code, 2025. Accessed: 2025-09-24.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021a. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

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

He Bai, Peng Shi, Jimmy Lin, Luchen Tan, Kun Xiong, Wen Gao, Jie Liu, and Ming Li. Semantics of the unwritten: The effect of end of paragraph and sequence tokens on text generation with GPT2. In Jad Kabbara, Haitao Lin, Amandalynne Paullada, and Jannis Vamvas (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pp. 148–162, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-srw.16. URL https://aclanthology.org/2021.acl-srw.16/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, and Greg Brockman. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, and Sebastian Gehrmann. PaLM:

Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, and Reiichiro Nakano. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, and Evan Rosen. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

OpenCompass Contributors. OpenCompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, and Alain Le Noac'h. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Google DeepMind. Gemini diffusion - google deepmind, 2025. URL https://deepmind.google/models/gemini-diffusion/. Accessed: 2025-09-19.

Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.

Haoyu He, Katrin Renz, Yong Cao, and Andreas Geiger. Mdpo: Overcoming the training-inference divide of masked diffusion language models, 2025. URL https://arxiv.org/abs/2508.13148.

Feng Hong, Geng Yu, Yushi Ye, Haicheng Huang, Huangjie Zheng, Ya Zhang, Yanfeng Wang, and Jiangchao Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms, 2025. URL https://arxiv.org/abs/2507.18578.

Zhanqiu Hu, Jian Meng, Yash Akhauri, Mohamed S. Abdelfattah, Jae sun Seo, Zhiru Zhang, and Udit Gupta. Accelerating diffusion language model inference via efficient kv caching and guided diffusion, 2025. URL https://arxiv.org/abs/2505.21467.

Chihan Huang and Hao Tang. Ctrldiff: Boosting large diffusion language models with dynamic block prediction and controllable generation, 2025. URL https://arxiv.org/abs/2505.14455.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, and Alec Radford. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Daniel Israel, Guy Van den Broeck, and Aditya Grover. Accelerating diffusion llms via adaptive parallel decoding, 2025. URL https://arxiv.org/abs/2506.00413.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation, 2019.

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. Any-order flexible length masked diffusion, 2025. URL https://arxiv.org/abs/2509.01025.

Philippe Laban, Andrew Hsi, John Canny, and Marti A. Hearst. The summary loop: Learning to write abstractive summaries without examples. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5135–5150, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.460. URL https://aclanthology.org/2020.acl-main.460/.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, and Akash Palrecha. Mercury: Ultra-fast language models based on diffusion, 2025. URL https://arxiv.org/abs/2506.17298.

Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models, 2025a. URL https://arxiv.org/abs/2508.00819.

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. A survey on diffusion language models, 2025b. URL https://arxiv.org/abs/2508.10875.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Omer Luxembourg, Haim Permuter, and Eliya Nachmani. Plan for speed: Dilated scheduling for masked diffusion language models, 2025. URL https://arxiv.org/abs/2506.19037.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models, 2025. URL https://arxiv.org/abs/2505.15781.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models, 2025. URL https://arxiv.org/abs/2502.09992.

OpenAI. Introducing GPT-5. https://openai.com/index/introducing-gpt-5/, 2025. Accessed: 2025-09-24.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

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

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

Jamba Team, Barak Lenz, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, and Dan Padnos. Jamba-1.5: Hybrid transformer-mamba models at scale, 2024. URL https://arxiv.org/abs/2408.12570.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Dimitri von Rütte, Janis Fluri, Yuhui Ding, Antonio Orvieto, Bernhard Schölkopf, and Thomas Hofmann. Generalized interpolating discrete diffusion, 2025. URL https://arxiv.org/abs/2503.04482.

Guanghan Wang, Yair Schiff, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Remasking discrete diffusion models with inference-time scaling, 2025a. URL https://arxiv.org/abs/2503.00307.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models, 2025b. URL https://arxiv.org/abs/2509.06949.

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

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7655–7671, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.456. URL https://aclanthology.org/2024.findings-acl.456/.

Jiacheng Ye, Shansan Gong, Liheng Chen, Lin Zheng, Jiahui Gao, Han Shi, Chuan Wu, Xin Jiang, Zhenguo Li, and Wei Bi. Diffusion of thought: Chain-of-thought reasoning in diffusion language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 105345–105374. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/be30024e7fa2c29cac7a6dafcbb8571f-Paper-Conference.pdf.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning, 2025a. URL https://arxiv.org/abs/2410.14157.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models, 2025b. URL https://arxiv.org/abs/2508.15487.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding, 2025. URL https://arxiv.org/abs/2505.16990.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, and Zican Dong. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2), 2023.

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

A Experimental Details

A.1 Ablation Studies

Similarly, we present the exact numerical results our further analysis on EoT penalty in tables below.

Table 5: **Performance with different values of the EoT penalty coefficient λ_{eot}, with forward steps $T = 32$.** Evaluated on GPQA and GSM8K with Dream and LLaDA. Mean ($\pm$ standard deviation) over 3 seeds.

Model	Task	$\lambda_{\text{eot}} = 1.0$	1.1	1.2	1.3
Dream	GPQA	19.27 ±0.90	23.74 ±0.52	25.67 ±1.18	24.33 ±2.95
Dream	GSM8K	41.80 ±1.23	51.68 ±0.57	51.88 ±1.14	49.56 ±1.25
LLaDA	GPQA	19.27 ±1.52	23.74 ±1.44	26.71 ±1.45	22.77 ±1.24
LLaDA	GSM8K	40.21 ±0.64	47.66 ±0.18	47.49 ±1.01	46.17 ±0.55

Table 6: **Performance with different values of the EoT penalty coefficient λ_{eot}, with forward steps $T = 128$.** Evaluated on GPQA and GSM8K with Dream and LLaDA. Mean ($\pm$ standard deviation) over 3 seeds.

Model	Task	$\lambda_{\text{eot}} = 1.0$	1.1	1.2	1.3
Dream	GPQA	26.79 ±0.00	25.15 ±0.72	26.41 ±2.03	26.34 ±0.80
Dream	GSM8K	54.61 ±0.56	67.93 ±0.53	67.20 ±0.64	67.63 ±0.13
LLaDA	GPQA	19.94 ±0.93	26.12 ±0.80	29.91 ±1.77	28.20 ±1.23
LLaDA	GSM8K	46.25 ±0.08	64.22 ±0.79	68.99 ±0.92	67.27 ±1.03

B Additional Experiments

This appendix collects additional empirical studies that complement the main experiments. The studies cover (i) the cross-validation refinement pseudocode, (ii) a blockwise versus full-draft refinement ablation, (iii) a fair EoT-penalty comparison applied to all baselines, (iv) sensitivity to the fill-up/refinement allocation ratio ρ, (v) scaling to generation lengths up to 1024, (vi) compatibility with Fast-dLLM, (vii) per-step latency measurements, (viii) batched-inference profiling, and (ix) a confidence-guided remask sampling study.

B.1 Cross-Validation Refinement Pseudocode

We provide the complete pseudocode for the cross-validation refinement procedure in Algorithm 1. It details the remask-and-resample loop together with the cosine annealing of the refinement rate γ_k.

B.2 Blockwise vs. Full-Draft Refinement

A natural alternative to applying refinement after a full draft is to refine each block immediately as it is produced. We compare five configurations on LLaDA-8B-Instruct (HumanEval, 0-shot, $\#F = 64$, $\rho = 0.5$, subset size $= 256$): vanilla full-attention decoding, vanilla blockwise decoding (block length $\in \{16, 32\}$), and three FiRe variants—full-draft refinement, per-block refinement, and blockwise fill-up followed by a single full-draft refinement.

Table 7 reports pass@1. Refining intermediate blockwise drafts is unstable, with per-block refinement even degrading performance at block length 16 (-1.2). In contrast, refining the full draft yields large and consistent gains regardless of how the draft is produced ($+10.3$ at block length 16 and $+6.7$ at block length 32). This supports the central design choice of FiRe: refinement should operate on a completed draft so that cross-validation across tokens becomes feasible.

Algorithm 1 Cross-Validation Refinement

Require: mask predictor p_θ, prompt p_0, draft $x^{(\rho T)}$, refinement steps K, $\gamma_{\max}$, $\gamma_{\min}$

$L' \leftarrow$ length of generated tokens in $x^{(\rho T)}$
for $k \leftarrow 1$ to K **do**
  $\gamma_k \leftarrow \gamma_{\min} + \frac{1}{2}(\gamma_{\max} - \gamma_{\min})(1 + \cos(\pi k/K))$ ▷ Cosine annealing
  $n_{\text{mask}} \leftarrow \lfloor \gamma_k \cdot L' \rfloor$
  $S^{(k)} \leftarrow$ randomly sample n_{mask} positions from generated tokens
  for $i \leftarrow 1$ to L' **do**
    if $i \in S^{(k)}$ **then**
      $x_i^{(k)} \leftarrow [\text{MASK}]$ ▷ Remask selected positions
    end if
  end for
  $\hat{x}^{(k)} \leftarrow \arg\max p_\theta(\cdot \mid p_0, x^{(k)})$ ▷ Full forward pass
  for $i \in S^{(k)}$ **do**
    $x_i^{(k+1)} \leftarrow \hat{x}_i^{(k)}$ ▷ Replace with new predictions
  end for
end for
return $x^{(K+1)}$

Table 7: **Blockwise vs. full-draft refinement.** Pass@1 on HumanEval (LLaDA-8B-Instruct, 0-shot, $\#F = 64$, $\rho = 0.5$, subset size $= 256$). Refining the full draft yields large gains under both block lengths; refining per block is unstable.

Method	Block	pass@1 (%)
Full	N/A	17.7
Full + FIRE	N/A	18.9 (+1.2)
Block	16	11.0
Block + per-block refine	16	9.8 (-1.2)
Block + full-draft refine	16	**21.3 (+10.3)**
Block	32	13.4
Block + per-block refine	32	15.2 (+1.8)
Block + full-draft refine	32	**20.1 (+6.7)**

B.3 FAIR EOT-PENALTY COMPARISON

In the main paper the EoT penalty is reported only for FIRE. To rule out the possibility that the penalty disproportionately favours our method, we sweep $\lambda_{\text{eot}} \in \{1.0, 1.1, 1.2, 1.3\}$ for all baselines as well, on LLaDA (GSM8K, 0-shot) and Dream (MBPP, 4-shot) with $\#F = 64$ and subset size $= 256$. Tables 8 and 9 report the results.

Two observations follow. First, the EoT penalty benefits all methods, indicating that it is not a FIRE-specific trick. Second, even when each baseline is allowed its own best λ_{eot}, FIRE remains the strongest method on both benchmarks (68.36 vs. 61.72 on GSM8K, 29.55 vs. 25.55 on MBPP).

Table 8: **EoT penalty sweep on LLaDA, GSM8K (0-shot, $\#F = 64$, subset size $= 256$).** Columns are λ_{eot}.

Method	1.0	1.1	1.2	1.3
Vanilla	42.58	56.25	54.69	53.91
ReMDM	43.36	56.25	53.91	53.52
RCR	50.00	61.72	57.81	59.77
FIRE	55.08	64.45	**68.36**	66.02

Table 9: **EoT penalty sweep on Dream, MBPP (4-shot, $\#F = 64$, subset size $= 256$).** Columns are λ_{eot}.

Method	1.0	1.1	1.2	1.3
Vanilla	24.07	25.41	24.98	25.55
ReMDM	17.27	19.33	22.21	24.43
RCR	15.93	16.72	19.44	22.95
FIRE	24.55	26.75	28.10	**29.55**

B.4 SENSITIVITY TO THE ALLOCATION RATIO ρ

FIRE requires an allocation ratio ρ between fill-up and refinement steps. We sweep $\rho \in \{0.1, \ldots, 0.9\}$ on Dream-Instruct (GSM8K, fill-up EoT penalty 1.1, subset size $= 256$) at total step budgets $\#F \in \{16, 32, 64, 128, 256\}$. Figure 3 shows that performance is robust across a wide range of ρ, with a broad plateau between $\rho = 0.3$ and $\rho = 0.7$. This justifies the simple default $\rho = 0.5$ used throughout the main paper.

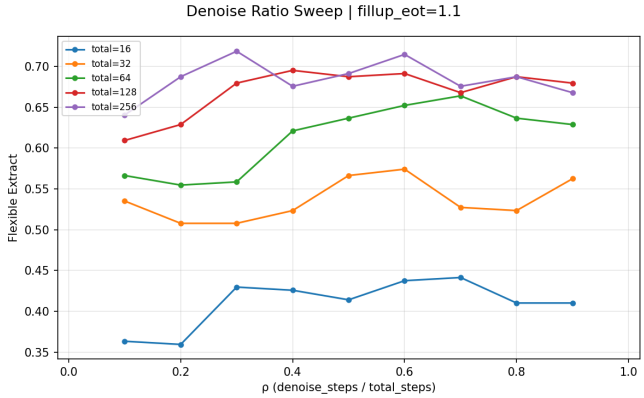


Figure 3: **Sensitivity to the allocation ratio** ρ (Dream-Instruct, GSM8K). Performance is robust across $\rho \in [0.3, 0.7]$ for all total step budgets.

B.5 SCALING TO LONGER GENERATION LENGTHS

We evaluate FIRE on substantially longer generation lengths than those used in the main paper. Table 10 reports results on LLaDA (GSM8K, 0-shot, subset size $= 128$) for generation lengths $L \in \{128, 256, 512, 1024\}$ under two constrained budgets, $\#F = L/2$ and $\#F = L/4$. FIRE achieves the highest accuracy in all eight configurations, and its advantage over vanilla decoding *grows* with sequence length (from $+12.5$ at $L = 128$ to $+35.2$ at $L = 1024$ under $\#F = L/4$).

Table 10: **Scaling to longer generation lengths under constrained budgets** (LLaDA, GSM8K, 0-shot, subset size $= 128$).

L	$\#F$	Vanilla	ReMDM	RCR	FIRE
$\#F = L/2$					
128	64	55.47	50.78	56.25	**74.22**
256	128	54.69	52.34	51.56	**70.31**
512	256	49.22	50.78	56.25	**75.00**
1024	512	46.09	48.44	54.69	**75.00**
$\#F = L/4$					
128	32	47.66	47.66	50.78	**60.16**
256	64	42.97	43.75	54.69	**66.41**
512	128	46.88	42.19	48.44	**72.66**
1024	256	44.53	42.19	45.31	**79.69**

To further verify that FIRE does not rely on constrained budgets, we additionally evaluate the regime $\#F = L$ (one forward step per token), which corresponds to the highest-quality generation setting for dLLMs. Table 11 shows that FIRE still consistently outperforms all baselines at every generation length and that its advantage widens with L.

Table 11: **Performance under the $\#F = L$ regime** (LLaDA-8B-Instruct, GSM8K, 0-shot, subset size $= 256$). Δ is the gain of FIRE over Vanilla.

$L = \#F$	Vanilla	ReMDM	RCR	FIRE	Δ
128	56.25	56.25	56.25	**69.53**	+13.28
256	53.52	53.52	54.69	**72.66**	+19.14
512	51.17	49.22	50.78	**71.48**	+20.31
1024	50.78	49.22	52.34	**78.52**	+27.74

B.6 COMPATIBILITY WITH FAST-DLLM

FIRE is orthogonal to inference-acceleration techniques such as Fast-dLLM (Wu et al., 2025), which speeds up the fill-up phase via KV caching and parallel decoding. We combine Fast-dLLM with FIRE on LLaDA (GSM8K, 5-shot, subset size $= 64$, temperature $= 0$). Table 12 shows that adding FIRE on top of Fast-dLLM further improves accuracy at both block lengths (e.g., $73.44 \rightarrow 81.25$ at block length 16), confirming that FIRE can be stacked with acceleration techniques without conflict.

Table 12: **Compatibility with Fast-dLLM** (LLaDA, GSM8K, 5-shot, subset size $= 64$, temperature $= 0$). *blk*: block length; *thr*: confidence threshold; *fill*/*refine*: number of fill-up/refinement forward steps.

Method	blk	thr	fill	refine	Acc.
Vanilla	N/A	N/A	80	N/A	46.48
Vanilla + FIRE	N/A	N/A	40	40	63.28
Fast-dLLM	32	0.9	$\sim$80	0	68.75
+ FIRE	32	0.8	$\sim$60	20	**71.88**
Fast-dLLM	16	0.9	$\sim$80	0	73.44
+ FIRE	16	0.8	$\sim$60	20	**81.25**

B.7 PER-STEP LATENCY MEASUREMENT

To confirm that the number of forward steps $\#F$ is a faithful cost proxy and that FIRE does not introduce hidden overhead, we measure per-forward-step wall-clock latency on a single NVIDIA GH200 GPU (120GB). For each configuration we run 50 instances with the first 5 discarded as warm-up. For FIRE, we report the fill-up and refinement stages separately.

Table 13 reports LLaDA latencies (in ms / forward step). All methods are within 2% of each other at the same $\#F$, and FIRE's refinement stage is not slower than vanilla decoding. This validates the use of $\#F$ as a fair cost proxy across methods.

Table 13: **Per-forward-step latency (ms)** on LLaDA-8B (NVIDIA GH200). All methods are within 2% of each other.

$\#F$	Vanilla	ReMDM	RCR	FIRE fill	FIRE refine
32	33.1	33.2	33.5	33.0	33.1
64	33.1	33.2	33.6	33.1	32.9
128	33.0	33.1	33.5	32.9	27.7

B.8 Batched Inference and GPU Utilization

Unlike autoregressive models, dLLMs do not benefit from batched inference. We measure wall-clock time, GPU utilization, and peak memory at different batch sizes on GSM8K with $\#F = 256$, running 64 instances per configuration. Table 14 confirms that increasing the batch size *slows down* dLLM inference and increases memory consumption, because the forward pass already saturates GPU compute at batch size 1. We elaborate on the underlying mechanism in Appendix D.5.

Table 14: **Batched-inference profiling** (GSM8K, $\#F = 256$, 64 instances). BS: batch size. Wall-clock time *increases* with batch size.

Model	BS	Time (s)	Util. (%)	Mem (MiB)
Dream	1	209.1	95.8	22,473
Dream	8	351.7 (1.68×)	100	33,415
LLaDA	1	309.3	100	39,743
LLaDA	16	420.4 (1.36×)	100	63,967

B.9 Confidence-Guided Remask Sampling

A natural variant of FiRe is to guide the remask subset by token confidence rather than sampling uniformly. We introduce a confidence update rate that interpolates between uniform sampling (rate 1.0) and fully confidence-prioritized sampling (rate 0.2), and sweep this rate on Dream-Instruct (GSM8K, 4-shot, $\rho = 0.5$, subset size $= 256$) across refinement step budgets $\in \{2, 4, 8, 16, 32, 64\}$. Figure 4 shows that performance is essentially flat across the swept update rates at every step budget. Confidence guidance neither improves nor degrades results, which is why we adopt uniform sampling as the default for simplicity.

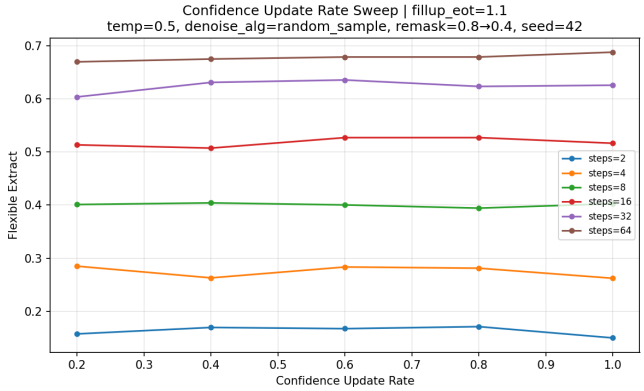


Figure 4: **Confidence-guided remask sampling** (Dream-Instruct, GSM8K). Performance is stable across all confidence update rates; uniform sampling (rate $= 1.0$) is a simple and effective default.

C Additional Controlled Studies

This appendix reports three further studies: (i) a full controlled comparison of all methods with and without the EoT penalty, (ii) an early-stopping criterion for the refinement stage, and (iii) a sign-aware formulation of the EoT penalty.

C.1 Controlled Comparison With and Without the EoT Penalty

The main tables apply the EoT penalty to FiRe only, and Appendix B.3 extends it to the baselines in two settings. To attribute the gains of FiRe to refinement rather than to the penalty, we evaluate Vanilla, ReMDM, RCR, and FiRe both without and with the EoT penalty on both backbones, four

Table 15: **Controlled EoT comparison on Dream.** Each cell reports the score (%) without / with the EoT penalty. Bold marks the best method separately under each condition.

Benchmark	$\#F$	Vanilla	ReMDM	RCR	FIRE
GSM8K	4	3.91 / 4.69	**16.41** / 17.19	7.03 / 8.59	**16.41 / 17.97**
	8	3.12 / 7.81	26.56 / **31.25**	13.28 / 9.38	**35.16** / 25.78
	16	7.81 / 7.03	32.03 / **43.75**	22.66 / 26.56	**35.94** / 41.41
	32	14.84 / 12.50	42.97 / 53.91	46.09 / **54.69**	**51.56** / 53.91
	64	21.09 / 27.34	**53.91** / 65.62	50.00 / **68.75**	51.56 / 67.97
	128	45.31 / 37.50	**71.88** / 78.12	52.34 / 72.66	51.56 / **79.69**
	256	41.41 / 59.38	**75.78** / **82.81**	40.62 / 77.34	48.44 / 78.12
TriviaQA	4	6.25 / 7.03	11.72 / 7.03	9.38 / 7.81	**16.41 / 11.72**
	8	8.59 / 5.47	13.28 / 6.25	9.38 / 8.59	**15.62 / 11.72**
	16	11.72 / 7.03	11.72 / 7.03	5.47 / 8.59	**19.53 / 17.19**
	32	12.50 / 10.94	11.72 / 7.03	10.94 / 9.38	**23.44 / 17.97**
	64	14.84 / 10.94	11.72 / 7.03	12.50 / 7.81	**21.88 / 21.09**
	128	17.97 / 10.94	14.06 / 13.28	15.62 / 12.50	**21.09 / 21.09**
	256	19.53 / 14.84	14.06 / 12.50	**23.44** / 16.41	17.97 / **23.44**
HumanEval	4	0.00 / 3.12	7.81 / **10.16**	2.34 / 7.03	**8.59** / 9.38
	8	0.78 / 1.56	9.38 / 16.41	7.81 / 11.72	**17.19 / 21.88**
	16	4.69 / 2.34	9.38 / 17.19	12.50 / 25.00	**27.34 / 36.72**
	32	7.03 / 3.91	14.06 / 23.44	20.31 / 33.59	**30.47 / 40.62**
	64	9.38 / 7.03	21.88 / 34.38	26.56 / 40.62	**42.97 / 47.66**
	128	18.75 / 24.22	28.12 / 38.28	28.12 / 50.78	**39.06 / 53.91**
	256	24.22 / 34.38	35.16 / 42.97	28.91 / **52.34**	**37.50** / 44.53
MBPP	4	0.78 / 2.34	10.16 / 8.59	5.47 / 7.81	**11.72 / 10.16**
	8	0.78 / 3.12	17.19 / 20.31	12.50 / 11.72	**29.69 / 25.78**
	16	1.56 / 5.47	16.41 / 19.53	21.09 / 22.66	**37.50 / 38.28**
	32	6.25 / 4.69	20.31 / 28.91	31.25 / 36.72	**37.50 / 46.88**
	64	11.72 / 14.84	36.72 / 38.28	35.94 / 46.09	**44.53 / 46.88**
	128	22.66 / 29.69	43.75 / 46.88	39.84 / 47.66	**48.44 / 50.00**
	256	30.47 / 49.22	**56.25** / 57.03	40.62 / **58.59**	54.69 / 57.03

benchmarks, and all forward-step budgets of the main tables (56 settings in total). All methods use the same total $\#F$. Each benchmark is evaluated on a 128-example subset with seed 42; for every method, λ_{eot} is selected from $\{1.1, 1.2, 1.3\}$ under the same search space.

Tables 15 and 16 report the results. Without the EoT penalty, FIRE is best or tied in 40/56 settings; with the EoT penalty applied to every method, FIRE remains best or tied in 32/56 settings. The gains of FIRE therefore persist when the EoT intervention is removed and cannot be attributed solely to the penalty.

C.2 Early Stopping for Refinement

The refinement stage runs for a fixed number of iterations because it does not converge naturally (Appendix E). We evaluate a simple early-stopping criterion. During refinement, we keep a per-position counter of consecutive identical re-predictions: a position whose token is reproduced k times in a row is frozen and excluded from later remasking, and refinement terminates once every position is frozen or the original iteration cap is reached. The rule adds no extra forward passes.

We sweep k on both backbones over GSM8K, HumanEval, and MBPP (0-shot, 32 fill-up steps plus at most 32 refinement iterations, 256-instance subsets, full HumanEval). Table 17 reports accuracy together with the average number of refinement iterations actually used; $k = \infty$ disables the rule.

With $k = 5$, accuracy remains comparable in five of six settings while the average number of refinement iterations drops by 9%–38%, with 35%–84% of instances stopping before the cap. The exception is LLaDA/HumanEval, although $k = 8$ and $k = 3$ match the no-stop baseline. Smaller k degrades mathematical reasoning (about 7–9 points on GSM8K at $k = 3$), whereas code tasks are more tolerant.

Table 16: **Controlled EoT comparison on LLaDA.** Each cell reports the score (%) without / with the EoT penalty. Bold marks the best method separately under each condition.

Benchmark	$\#F$	Vanilla	ReMDM	RCR	FIRE
GSM8K	4	5.47 / 11.72	7.81 / 14.06	5.47 / **27.34**	**14.06** / 19.53
	8	9.38 / 17.19	**23.44** / 21.09	12.50 / **42.97**	20.31 / 26.56
	16	21.88 / 41.41	32.03 / 37.50	**32.81** / **52.34**	26.56 / 37.50
	32	38.28 / 57.03	**46.88** / 57.81	45.31 / **61.72**	40.62 / 55.47
	64	42.19 / 64.06	41.41 / 59.38	50.00 / 66.41	**52.34** / **67.19**
	128	53.12 / 66.41	51.56 / 67.19	55.47 / 67.19	**63.28** / **71.88**
	256	54.69 / 64.84	49.22 / 65.62	57.03 / 67.19	**66.41** / **75.78**
TriviaQA	4	0.00 / 0.00	0.00 / 0.00	0.00 / 0.00	**0.78** / **0.78**
	8	0.00 / **0.00**	0.00 / **0.00**	0.00 / **0.00**	**1.56** / **0.00**
	16	0.00 / **0.78**	0.00 / **0.78**	**1.56** / **0.78**	0.78 / **0.78**
	32	1.56 / 0.78	3.12 / 0.78	**3.91** / **1.56**	3.12 / **1.56**
	64	0.78 / 0.78	0.78 / 1.56	3.91 / 1.56	**8.59** / **3.12**
	128	2.34 / 2.34	3.12 / 2.34	**7.03** / 2.34	**7.03** / **7.81**
	256	9.38 / 5.47	**10.94** / 5.47	**10.94** / 6.25	10.16 / **7.81**
HumanEval	8	**17.19** / 10.94	11.72 / 6.25	10.94 / **13.28**	12.50 / 12.50
	16	16.41 / 14.84	21.88 / 16.41	10.16 / **19.53**	**26.56** / 18.75
	32	20.31 / 18.75	19.53 / 15.62	17.19 / **30.47**	**26.56** / 23.44
	64	21.88 / 18.75	24.22 / 18.75	20.31 / **25.00**	**26.56** / 20.31
	128	**26.56** / 21.88	18.75 / 17.97	25.00 / **27.34**	23.44 / 23.44
	256	**29.69** / **30.47**	25.78 / 23.44	22.66 / 24.22	28.91 / 18.75
	512	27.34 / 25.78	25.00 / 22.66	21.88 / **28.91**	**28.91** / 25.78
MBPP	4	8.59 / 7.03	3.91 / 3.91	**10.16** / **14.84**	9.38 / 8.59
	8	**19.53** / 12.50	5.47 / 10.16	7.03 / **18.75**	**19.53** / 14.84
	16	17.19 / 16.41	18.75 / 14.84	13.28 / 19.53	**25.00** / **25.00**
	32	20.31 / 17.97	19.53 / 17.19	**25.00** / **31.25**	**25.00** / 27.34
	64	26.56 / 25.00	26.56 / 26.56	26.56 / 35.94	**28.91** / **36.72**
	128	32.81 / 35.16	33.59 / 34.38	24.22 / **45.31**	**34.38** / 39.06
	256	**35.94** / 38.28	34.38 / **39.06**	30.47 / 36.72	33.59 / 37.50

Table 17: **Early stopping for refinement.** Each cell reports accuracy (%) with the average number of refinement iterations used in parentheses (cap 32). $k = \infty$ disables early stopping.

Model	Task	$k = \infty$	$k = 10$	$k = 8$	$k = 5$	$k = 3$	$k = 2$
LLaDA	GSM8K	66.41 (32.0)	64.06 (31.9)	70.31 (31.1)	66.02 (24.8)	57.81 (12.7)	55.08 (7.2)
LLaDA	HumanEval	18.29 (32.0)	18.90 (32.0)	18.90 (31.7)	12.80 (29.2)	18.90 (15.6)	15.85 (8.2)
LLaDA	MBPP	24.22 (32.0)	28.12 (31.9)	27.73 (31.7)	24.22 (29.0)	28.52 (17.0)	21.88 (8.3)
Dream	GSM8K	64.45 (32.0)	65.23 (31.8)	62.89 (31.3)	62.11 (25.3)	57.42 (12.4)	51.95 (7.2)
Dream	HumanEval	40.85 (32.0)	42.07 (31.5)	39.63 (29.9)	42.07 (22.1)	39.02 (10.8)	37.80 (6.6)
Dream	MBPP	41.41 (32.0)	44.53 (30.6)	46.88 (28.2)	46.88 (19.9)	44.92 (10.7)	38.28 (6.8)

C.3 SIGN-AWARE EOT PENALTY

Dividing the EoT logit by $\lambda_{\text{eot}} > 1$ reduces the EoT probability only when the logit is positive; for a negative logit, the division moves it towards zero. Following the standard repetition-penalty implementation (Keskar et al., 2019), we therefore use the sign-aware form in Section 3.3: $z_{\text{eot}}/\lambda_{\text{eot}}$ if $z_{\text{eot}} \geq 0$ and $\lambda_{\text{eot}} z_{\text{eot}}$ otherwise.

We compare the direct-division and sign-aware penalties on a GSM8K subset with LLaDA under the same random seed (Table 18). The scores are identical, and manual inspection confirms instance-level matches. Logging the sign of the EoT logit explains this: negative EoT logits occur in 0.00%–0.01% of fill-up positions and 0.03%–0.13% of refinement positions. This is consistent with our motivation that the EoT token is generally over-competitive during generation, so its logit is rarely negative.

Table 18: **Direct-division vs. sign-aware EoT penalty** (LLaDA, GSM8K subset, same seed). Right: fraction of positions with a negative EoT logit.

$\#F$	λ_{eot}	Direct division	Sign-aware
32	1.0	40.62	40.62
32	1.1	55.47	55.47
32	1.2	50.00	50.00
32	1.3	54.69	54.69
64	1.0	52.34	52.34
64	1.1	67.19	67.19
64	1.2	67.19	67.19
64	1.3	63.28	63.28

$\#F$	$z_{\text{eot}} < 0$ (fill-up)	$z_{\text{eot}} < 0$ (refinement)
32	0.00%	0.05%–0.09%
64	0.00%–0.01%	0.03%–0.13%

D Discussion

D.1 Why our method can be training-free?

Our method does not require additional training, yet still achieves significant improvements, because the refinement stage closely matches the training objective of dLLMs. During training, these models reconstruct randomly masked tokens given the visible context, and the loss is computed uniformly over all masked positions. This is exactly what our refinement stage does: in each iteration, a subset of tokens is remasked and predicted conditioned on the others, so that every token can in turn act as target or context under the same conditional distribution the model was optimized for.

Vanilla decoding, by contrast, is not fully consistent with this training setup. At inference it accepts a small set of the most confident tokens (based on logit features). The predictions are evaluated only on those high-confidence positions, while other masked tokens contribute nothing and simply get remasked—despite the fact that, during training, losses were computed on them as well.

D.2 Why our strategy is good for large parallel sizes?

Our method achieves better improvements in parallel decoding scenarios, i.e., when the forward step is smaller than the sequence length and multiple tokens are decoded in each forward step.

One key reason may lie in the visibility constraint during parallel decoding: tokens generated within the same step cannot attend to each other, which often leads to local inconsistencies. This phenomenon is even more noticeable with larger parallel sizes. Our token-level cross-validation process helps to mitigate this issue. During cross-validation, tokens filled up in the same step can be validated such that one serves as context (or validator) while another serves as the validation target. This mechanism enables tokens that were originally invisible to each other to interact directly—for example, when validating token A, token B (from the same step) can now be used as part of the context. Such interactions promote coherence among simultaneously decoded tokens. By repeating this process across multiple rounds, inconsistencies introduced by parallel decoding are progressively reduced, resulting in more coherent sequences overall.

In contrast, when the forward step equals the sequence length (i.e., non-parallel decoding with one token per step), every token naturally conditions on all previously accepted tokens. Since there is no within-step invisibility, the inconsistency problem does not arise, and thus the potential benefit of our method is relatively limited in this scenario.

D.3 Conceptual Distinction from Prior Remasking Methods

A natural question is whether FiRe is essentially a re-timing of remasking-based decoding methods such as ReMDM (Wang et al., 2025a) and RCR (He et al., 2025), i.e., applying remasking operations to a fully completed draft rather than to intermediate, partially masked states. While this is one observable difference, we emphasize that FiRe differs from prior remasking strategies in two deeper, conceptually distinct ways.

Cross-validation mechanism. Because FIRE operates on a fully decoded draft, a selected subset of tokens is remasked and re-predicted using *the remaining tokens as validators*. Each token can serve as both a validation target and a validator across iterations—a property that cannot be realized by previous remasking methods, which always condition on partially masked contexts during refinement. This token-level cross-validation enables coherence among tokens that were originally decoded in parallel and thus could not attend to one another.

Refinement rate annealing. FIRE introduces a cosine annealing schedule for the refinement rate γ_k, providing stronger revision in early iterations and gradual stabilization in later ones. This design is absent in conventional remasking strategies, which adopt fixed or task-agnostic remasking probabilities.

These distinctions are not merely implementation differences. They reflect a fundamentally different design philosophy: explicitly decoupling fill-up and refinement into two separate stages, rather than interleaving partial revision within a single denoising trajectory.

D.4 Why Generation Length Affects dLLMs Differently from AR Models

Unlike autoregressive models, where the causal mask ensures that increasing the generation length does not affect the quality of previously generated tokens, dLLMs attend to and denoise all tokens simultaneously. As a result, increasing the generation length does not guarantee improved accuracy (Li et al., 2025a; Kim et al., 2025).

In our setup, longer sequences lower the signal-to-noise ratio at early denoising steps, as each token attends to a much larger set of mostly masked positions. More erroneous tokens are committed early and accumulate throughout denoising. However, FIRE does not exhibit this degradation: by explicitly decoupling generation into fill-up and refinement stages, it affords substantially more opportunities to correct early errors rather than allowing them to accumulate through subsequent denoising steps. This mechanism helps explain why FIRE's advantage over vanilla decoding grows with generation length, as observed in the scaling experiments reported in Appendix B.

D.5 Why Batching Does Not Accelerate dLLM Inference

In autoregressive models, batched inference provides near-free speedup because at each decoding step only a length-1 query is computed against the KV cache. The GPU compute units are largely underutilized (memory-bound), and packing multiple examples effectively turns matrix–vector operations into matrix–matrix operations without additional wall-clock cost.

In diffusion LLMs, however, every forward pass already processes the full sequence for attention and feed-forward computation, which saturates GPU utilization even at batch size 1. Increasing the batch size therefore does not unlock idle compute; it simply increases memory consumption with no—and often even negative—speedup, as we empirically observe in Appendix B. This compute-bound nature of dLLM inference has also been noted in recent work.

E Limitations

Format Stability. Although our method achieves consistent improvements across benchmarks, the gains are relatively smaller on code generation tasks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b). These tasks are highly format-sensitive, where even slight deviations in syntax or structure can render an otherwise correct solution invalid. Because our refinement process operates purely at the token level and does not enforce explicit structural constraints, it can occasionally disrupt the formatting of well-formed code. This indicates a limitation of our approach when applied to domains that demand strict output adherence. A similar limitation has also been noted in methods such as RCR (He et al., 2025), which require heavier remasking than vanilla generation and can likewise compromise sequence formatting.

Natural Convergence. The cross-validation refinement in our method can be viewed as an iterative self-mapping process, where the input and output spaces are identical. Ideally, such a process should terminate naturally when, after a certain iteration, the sequence reaches a stable fixed

point—subsequent updates would then map the sequence onto itself, eliminating the need to predefine a fixed number of steps. In practice, however, our current approach does not exhibit this kind of natural convergence. Even after many refinement steps, while the number of tokens edited by the model tends to decrease, it never converges to zero. Instead, the process continues to oscillate.

F Qualitative Examples

In this section, we present a qualitative case study showing how cross-validation refinement performs token-level error correction. Consider the GSM8K (Cobbe et al., 2021) query: "*Marilyn's first record sold 10 times as many copies as Harald's. If they sold 88,000 copies combined, how many copies did Harald sell?*"

Figure 5 shows LLaDA (Nie et al., 2025) with 16 fill-up forward steps without refinement. Different colors indicate the decoding order. Early steps are syntactically and semantically reasonable, but by step 16 (orange) the output degrades—failing to carry out precise arithmetic (e.g., producing "88,000000") and losing grammatical consistency (e.g., "the number the number").

Cross-validation refinement then iteratively edits inconsistent tokens and inserts more appropriate ones. Figure 5 shows results after 1, 8, and 16 refinement steps, respectively. After one step, redundant tokens like "the number the number" are removed and spurious digits are trimmed (e.g., "88,000000" $\rightarrow$ "88,000"), though minor issues (e.g., fragments like "Har Harald") may remain. As refinement proceeds, the text becomes well-formed and the arithmetic is corrected, ultimately converging to the correct answer: 88,000.

This example indicates that dLLMs already contain the knowledge needed to solve the problem since initial generations include relevant fragments such as "88" and "00/0000", but benefit from a more stable mechanism for parallel language modeling. Cross-validation refinement provides this mechanism, enabling correction over iterations.

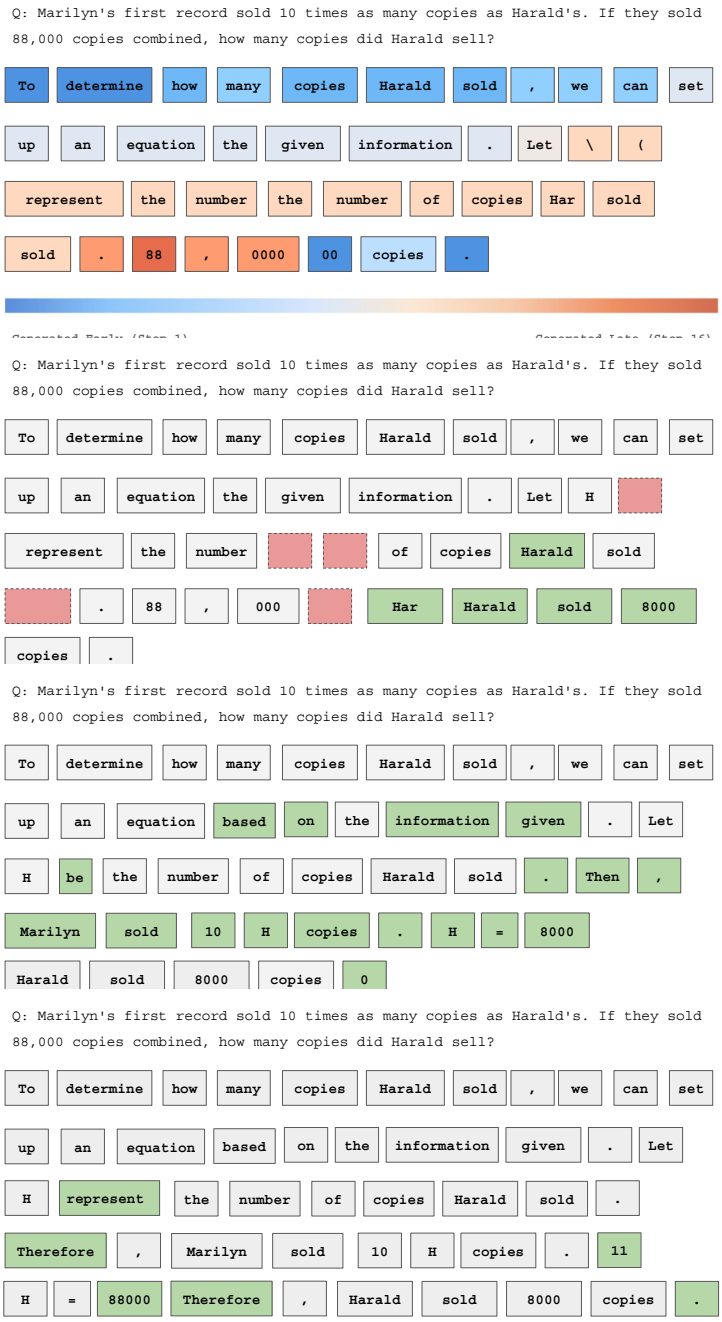


Figure 5: **Decoding snapshots from fill-up to refinement. (a) Fill-up output:** token colors fade from blue to red to indicate generation time order; we use 16 fill-up steps and 16 refinement steps, and omit special tokens such as [EoT]. **(b,c,d) After 1, 8, and 16 refinement iterations:** red dashed boxes mark deleted tokens and green boxes mark newly added tokens.

G HYPERPARAMETER SETTINGS FOR BASELINES

Table 19: Grid search results for ReMDM across tasks, models, and α_{on}. Each column corresponds to a $(t_{\text{off}}, t_{\text{on}})$ pair, and rows report performance under different settings of α_{on} for Dream and LLaDA.

Task	Model	α_{on}	$t_{\text{off}} = 0.05$ $t_{\text{on}} = 0.25$	$t_{\text{off}} = 0.20$ $t_{\text{on}} = 0.40$	$t_{\text{off}} = 0.40$ $t_{\text{on}} = 0.60$	$t_{\text{off}} = 0.60$ $t_{\text{on}} = 0.80$	$t_{\text{off}} = 0.75$ $t_{\text{on}} = 0.95$	$t_{\text{off}} = 0.05$ $t_{\text{on}} = 0.45$
MBPP	Dream	0.95	32.81	31.25	32.81	31.25	23.44	32.81
		0.90	32.81	31.25	32.81	31.25	28.12	31.25
	LLaDA	0.95	9.38	21.88	9.38	18.75	15.62	6.25
		0.90	9.38	9.38	6.25	18.75	21.88	0.00
GSM8K	Dream	0.95	35.94	35.94	35.94	35.94	35.94	35.94
		0.90	35.94	35.94	35.94	35.94	35.94	35.94
	LLaDA	0.95	28.12	46.88	40.62	46.88	40.62	37.50
		0.90	28.12	28.12	46.88	40.62	43.75	28.12

Task	Model	α_{on}	$t_{\text{off}} = 0.20$ $t_{\text{on}} = 0.60$	$t_{\text{off}} = 0.40$ $t_{\text{on}} = 0.80$	$t_{\text{off}} = 0.55$ $t_{\text{on}} = 0.95$	$t_{\text{off}} = 0.05$ $t_{\text{on}} = 0.65$	$t_{\text{off}} = 0.20$ $t_{\text{on}} = 0.80$	$t_{\text{off}} = 0.35$ $t_{\text{on}} = 0.95$
MBPP	Dream	0.95	32.81	32.81	25.00	31.25	31.25	29.69
		0.90	31.25	32.81	28.12	31.25	31.25	28.12
	LLaDA	0.95	12.50	15.62	18.75	6.25	12.50	21.88
		0.90	18.75	12.50	9.38	6.25	12.50	15.62
GSM8K	Dream	0.95	35.94	35.94	37.50	35.94	35.94	35.94
		0.90	35.94	37.50	39.06	35.94	35.94	45.31
	LLaDA	0.95	34.38	28.12	43.75	40.62	31.25	37.50
		0.90	34.38	53.12	34.38	40.62	34.38	37.50

To ensure fairness and reproducibility, we performed a systematic hyperparameter search for the baseline methods. Below we describe the details.

ReMDM (Wang et al., 2025a). ReMDM involves four hyperparameters in total: t_{on}, t_{off}, α_{on}, and the *remasking strategy*. For the strategy, we adopt the best-performing variant reported in the original work, *ReMDM-loop*. The original paper uses the configuration $t_{\text{on}} = 0.65$, $t_{\text{off}} = 0.05$, and $\alpha_{\text{on}} = 0.9$. Building on this configuration, we conducted an additional grid search. Specifically, we evaluated candidate hyperparameters on two models (LLaDA-8B-Instruct (Nie et al., 2025) and Dream-v0-Instruct-7B (Ye et al., 2025b)) and two tasks (MBPP (Austin et al., 2021b) and GSM8K (Cobbe et al., 2021)). For this procedure, we used subsets of 64 examples for Dream and 32 examples for LLaDA. We use a context length of 256 and forward step T of 32. We then selected the hyperparameters that achieved the best aggregate performance across both models and tasks. The resulting configuration is $t_{\text{on}} = 0.60$, $t_{\text{off}} = 0.40$, and $\alpha_{\text{on}} = 0.95$ (see Table 19).

RCR (He et al., 2025). This method does not involve tunable hyperparameters, and thus no hyperparameter search was required.