SWIFTDLM: FROM TRAJECTORY IMITATION TO SELF-EXPLORATION IN DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion large language models (dLLMs) have emerged as a promising alternative to autoregressive models due to their potential for parallel decoding. However, existing open-source dLLMs require substantial decoding steps to maintain generation quality, thereby limiting inference efficiency. While recent training-based acceleration methods reduce decoding steps, they do not explicitly enforce consistency between intermediate decoding states or explore new trajectories through reinforcement learning. To address these limitations, we propose ***SwiftDLM***, a unified two-stage training framework that substantially reduces decoding steps while maintaining or even improving generation performance. In the first stage, we employ consistency distillation on intermediate trajectory states to achieve skip-step generation. In the second stage, we leverage reinforcement learning to reward trajectories exhibiting both accuracy and efficiency, enabling self-exploration of superior decoding paths. Extensive evaluations demonstrate that our method outperforms existing dLLM acceleration approaches. When applied to LLaDA, our method achieves a 8.9× speedup on GSM8K while improving accuracy by 6 percentage points, and delivers an 11.3× speedup on MBPP while maintaining competitive performance.

1 INTRODUCTION

In recent years, autoregressive large language models (LLMs) Achiam et al. (2023); Grattafiori et al. (2024); Liu et al. (2024) have achieved remarkable success across various natural language tasks. However, the inherent left-to-right next-token prediction mechanism of autoregressive models fundamentally limits inference efficiency, creating a latency bottleneck. To address this challenge, diffusion large language models (dLLMs) have emerged as a promising alternative to autoregressive LLMs. By leveraging bidirectional attention mechanisms, dLLMs enable parallel multi-token prediction and text generation in arbitrary orders, demonstrating substantial potential for improving inference efficiency. For instance, recent closed-source dLLM models such as Mercury Khanna et al. (2025), Gemini Diffusion, and Seed Diffusion Song et al. (2025) exhibit superior performance and efficiency, with throughput exceeding 1,000 tokens per second, far surpassing autoregressive models of comparable size.

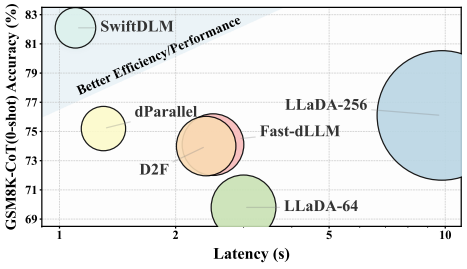


Figure 1: Efficiency–accuracy trade-off on GSM8K-CoT (LLaDA). Bubble size indicates decoding steps. LLaDA-64 and LLaDA-256 denote the vanilla LLaDA model with 64 and 256 decoding steps. SwiftDLM (top-left) demonstrates superior decoding efficiency and generation quality, significantly outperforming dParallel and Fast-dLLM.

However, current open-source dLLMs such as LLaDA Nie et al. (2025) and Dream Ye et al. (2025) have yet to achieve comparable throughput. In practice, their inference efficiency is primarily limited by two factors: (1) the inability to effectively utilize KV cache due to bidirectional attention, and (2) the requirement for substantial decoding steps, which are typically proportional to the generation sequence length, to maintain generation quality. To address these constraints, recent training-free approaches have attempted to approximate

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.

KV cache Ma et al. (2025); Hu et al. (2025); Liu et al. (2025a) and optimize sampling strategies Wu et al. (2025); Ben-Hamu et al. (2025); Li et al. (2025). Despite achieving some acceleration, these confidence-aware or entropy-bounded dynamic sampling strategies exhibit limited parallelism, typically unmasking only 3–5 tokens per step.

To enable decoding more tokens per step, recent training-based methods leverage distillation to produce more efficient models Wang et al. (2025b); Chen et al. (2025); Qian et al. (2026). For instance, dParallel Chen et al. (2025) employs certainty-forcing distillation, imposing an entropy loss on correctly predicted tokens to promote parallelism. However, such approaches inevitably trade generation quality for inference speed, failing to simultaneously achieve strong performance and high efficiency. We attribute this to two fundamental limitations of existing training-based approaches: **(1) Overlooking intermediate-state consistency.** Although d3LLM Qian et al. (2026) uses the teacher's decoding order to construct training inputs, it does not explicitly enforce consistency between predictions at different intermediate decoding states. We introduce this constraint to support skip-step generation. **(2) Constrained by teacher supervision.** Distillation-based methods rely on fixed teacher-derived supervision without directly optimizing newly sampled trajectories for correctness and efficiency.

To address these limitations, we propose a unified two-stage training framework that enables the model to transition from imitating teacher decoding trajectories to self-exploring optimal paths, thereby simultaneously improving both performance and efficiency. In the first stage, we leverage intermediate trajectory states to learn the sequential unmasking order and apply consistency distillation for skip-step generation. In the second stage, we employ reinforcement learning to reward trajectories exhibiting both correctness and efficiency, enabling the model to self-explore superior decoding paths beyond the teacher's constraints. The two stages are structurally coupled and synergistic: Stage I establishes a policy prior with skip-step capability that reduces the rollout cost of Stage II RL training, while Stage II in turn transcends the performance ceiling imposed by teacher trajectories—a limitation that distillation alone cannot overcome.

We validate our method on two representative open-source dLLMs: LLaDA (trained from scratch) and Dream (initialized from an autoregressive LLM). As shown in fig. 1, experiments across multiple benchmarks demonstrate that our method substantially reduces decoding steps while maintaining or even improving generation performance, outperforming existing dLLM acceleration approaches. For instance, on LLaDA, our method requires only 10% of the vanilla model's sampling steps on the mathematical reasoning benchmark GSM8K Cobbe et al. (2021), achieving an 8.9× speedup while improving accuracy by 6 percentage points. On the code generation benchmark MBPP Austin et al. (2021b), it uses merely 8% of the sampling steps, delivering a 11.3× speedup while maintaining competitive performance.

In summary, our main contributions are as follows:

- We propose ***SwiftDLM***, a two-stage training framework that accelerates dLLMs through trajectory imitation followed by self-exploration. Its first stage applies consistency distillation to intermediate states of teacher trajectories, capturing the teacher's unmasking order and enabling skip-step generation.
- We develop a reinforcement learning stage that rewards correct outputs generated with fewer decoding steps, enabling exploration beyond the teacher trajectories.
- Experiments on LLaDA and Dream across four mathematical reasoning and code generation benchmarks demonstrate a better efficiency–accuracy trade-off than the evaluated acceleration baselines.

2 Related Work

Diffusion Large Language Models (dLLMs). Diffusion models Ho et al. (2020); Song et al. (2020) have recently achieved success in generative modeling, particularly in continuous domains such as images Rombach et al. (2022). However, their application to discrete data such as text remains challenging. Masked Diffusion Models Austin et al. (2021a); Shi et al. (2024); Campbell et al. (2022); Zheng et al. (2024); Nie et al. (2025) have emerged as a promising approach, enabling parallel decoding by iteratively predicting masked tokens while leveraging bidirectional attention for holistic context modeling. Building on this paradigm, recent work has explored integrating diffusion with

existing LLM architectures at the billion-parameter scale. For instance, Dream Ye et al. (2025) converts pretrained autoregressive models into diffusion models, while LLaDA Nie et al. (2025) trains diffusion-based LLMs from scratch. Both models demonstrate competitive performance comparable to state-of-the-art autoregressive models Dubey et al. (2024). However, dLLMs suffer from inference inefficiency, as increasing decoding parallelism degrades generation quality, while their bidirectional nature renders them incompatible with KV caching.

Acceleration of dLLMs. To improve the inference efficiency of dLLMs while maintaining generation quality, existing acceleration methods can be divided into three categories. First, some approaches Ma et al. (2025); Liu et al. (2025a); Hu et al. (2025) leverage the high similarity of hidden states across consecutive denoising steps, employing approximate caching mechanisms to reduce the computational overhead per denoising step. Second, other works Ben-Hamu et al. (2025); Li et al. (2025); Wu et al. (2025) reduce the total number of denoising steps through optimized sampling strategies, such as confidence-aware or entropy-bounded approaches. Third, unlike the training-free methods above, some training-based approaches leverage distillation to reduce decoding steps or restructure the generation trajectory of dLLMs. For instance, dParallel Chen et al. (2025) leverages certainty-forcing distillation to enable more tokens to reach high certainty in parallel at each steps, while D2F Wang et al. (2025b) refurbishes dLLMs into an AR-diffusion hybrid paradigm through asymmetric distillation.

Reinforcement Learning (RL) for dLLMs Recent works have explored reinforcement learning to enhance the reasoning capabilities of dLLMs. VRPO (LLaDA-1.5) Zhu et al. (2025) adapts DPO Rafailov et al. (2023) to masked diffusion models via variance-reduced Monte Carlo estimation of the ELBO. Jazbec et al. (2025) trains a separate unmasking policy using GRPO Shao et al. (2024), while keeping the underlying dLLM fixed. Methods like d1 Zhao et al. (2025) and wd1 Tang et al. (2025) approximate per-token log-probabilities through a single unmasking step, which reduces overhead but introduces approximation bias by neglecting intermediate decoding states. Our method evaluates token probabilities in the intermediate contexts recorded along sampled decoding trajectories and uses task rewards together with decoding-step feedback to update the generation model.

3 PRELIMINARY

Notation. In this paper, we denote a sequence of tokens as $x = (x_1, \ldots, x_L) \in \mathcal{V}^L$, where L represents the sequence length and $\mathcal{V}$ is the vocabulary of size V. $\text{Cat}(\boldsymbol{x}; \boldsymbol{p})$ represents the categorical distribution over $\boldsymbol{x}$ with probabilities $\boldsymbol{p}$. In the context of dLLMs, we assume the vocabulary includes a special mask token $\mathbf{M}$.

3.1 DIFFUSION LARGE LANGUAGE MODELS

The most promising discrete diffusion approach for language modeling is Masked Diffusion Language Models (MDLMs). MDLMs employ a forward noising process that progressively replaces the clean text with a special [mask] token, corrupting it into x_t over a continuous timestep $t \in [0, 1]$. The forward process independently corrupts tokens via the following Markov chain:

$$q_{t|0}(\boldsymbol{x}_t^i \mid \boldsymbol{x}_0^i) = \text{Cat}(\boldsymbol{x}_t^i; \alpha_t \boldsymbol{x}_0^i + (1 - \alpha_t)\mathbf{M}), \tag{1}$$

where $\boldsymbol{x}^i$ denotes the i-th element of $\boldsymbol{x}$, and $\boldsymbol{x}_t$ represents the corrupted data at timestep t. The noise schedule α_t is a monotonically decreasing function in the interval $[0, 1]$, such as $\alpha_t = 1 - t$.

The reverse process utilizes a neural network, denoted as π_θ, to iteratively predict the original tokens x_0 from the noisy data x_t. Specifically, we parameterize the reverse transition $p_\theta(\boldsymbol{x}_s \mid \boldsymbol{x}_t)$ for $s < t$ as the posterior distribution $q(\boldsymbol{x}_s \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$, where $\boldsymbol{x}_0$ is replaced by the model prediction $\pi_\theta(\cdot \mid \boldsymbol{x}_t)$:

$$p_\theta(\boldsymbol{x}_s \mid \boldsymbol{x}_t) = \begin{cases} \text{Cat}(\boldsymbol{x}_s; \boldsymbol{x}_t), & \boldsymbol{x}_t \neq \mathbf{M}, \\ \text{Cat}\left(\boldsymbol{x}_s; \dfrac{(1 - \alpha_s)\mathbf{M} + (\alpha_s - \alpha_t)\pi_\theta(\cdot \mid \boldsymbol{x}_t)}{1 - \alpha_t}\right), & \boldsymbol{x}_t = \mathbf{M}. \end{cases} \tag{2}$$

To learn this reverse process, the training objective is to minimize the Negative Evidence Lower Bound (NELBO). Under the assumption of absorbing transitions, the NELBO simplifies to the fol-

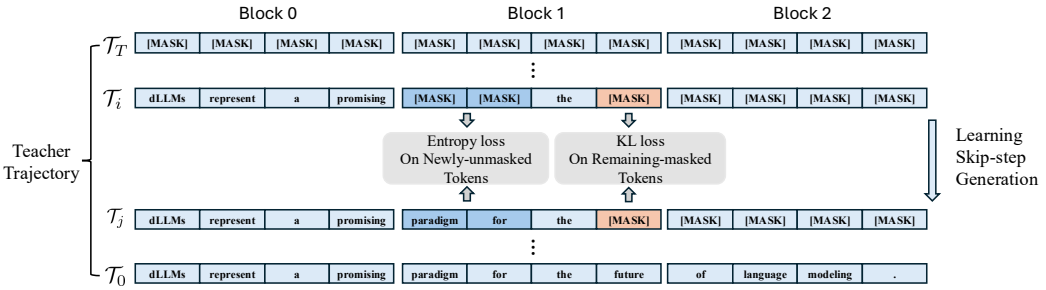


Figure 2: Illustration of trajectory compression via consistency distillation, where we align state $\mathcal{T}_i$ with subsequent state $\mathcal{T}_j$ within the same block. Cross-entropy loss is applied to newly-unmasked tokens and KL divergence to remaining-masked tokens, enabling skip-step generation.

lowing form Ou et al. (2024), with the detailed derivation provided in Appendix A:

$$\mathcal{L}(\theta) = -\mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{x}_t}\left[\frac{1}{t}\sum_{i=1}^{L}\mathbf{1}\left[\boldsymbol{x}_t^i = \mathbf{M}\right]\log \pi_\theta(\boldsymbol{x}_0^i \mid \boldsymbol{x}_t)\right]. \tag{3}$$

3.2 Parallel Decoding in dLLMs

Parallel decoding in dLLMs simultaneously unmasks tokens at multiple positions within a single step. Models such as LLaDA and Dream unmask a fixed number of tokens per step based on confidence scores $c_t^i := \max_{v\in\mathcal{V}} p_t^i(v)$ or entropy values $h_t^i := -\sum_{v\in\mathcal{V}} p_\theta(v \mid x_t^i)\log p_\theta(v \mid x_t^i)$. However, under the conditional independence assumption, such parallel sampling disrupts token dependencies, degrading generation quality. To address this, dynamic decoding strategies selectively unmask tokens exceeding a confidence threshold or falling below an entropy threshold, and we adopt this as our default sampling method:

$$\mathcal{U}_t := \{i \in \mathcal{M}_t \mid c_t^i > \lambda_{\text{conf}}\} \text{ or } \{i \in \mathcal{M}_t \mid h_t^i < \lambda_{\text{ent}}\}. \tag{4}$$

While these training-free methods alleviate dependency disruption, they exhibit constrained parallelism (typically unmasking 3–5 tokens per step), which we attribute to the misalignment between pre-training and the confidence- or entropy-based inference paradigm. To bridge this gap, we incorporate intermediate trajectory states from dynamic sampling into the training process, aligning the model with this inference paradigm and thereby unlocking greater parallel potential while enhancing generation performance.

4 Methodology

We propose a two-stage framework that jointly improves decoding efficiency and generation quality. Stage I uses consistency distillation to learn the teacher's decoding order and compress trajectories, reducing the rollout cost of subsequent reinforcement learning (RL). Stage II uses RL to explore decoding paths beyond the teacher trajectories, rewarding correct outputs generated with fewer steps.

4.1 Trajectory Compression via Consistency Distillation

Intermediate states of teacher trajectories capture the sequential unmasking order. We leverage these states and explicitly enforce consistency between their predictions to enable skip-step generation. To this end, we first utilize the block-wise decoding method Nie et al. (2025), where a low-confidence remasking strategy is used in each block to collect teacher decoding trajectories.

Teacher Trajectory Generation. Specifically, given a prompt x with generation length L and block size L_B, we record the unmasking process starting from a fully masked state, yielding the trajectory $\{\mathcal{T}_T, \ldots, \mathcal{T}_1\}$, where $\mathcal{T}_i \in \mathbb{R}^L$. The intermediate state $\mathcal{T}_i$ can be spatially partitioned into K blocks, satisfying $L = L_B \cdot K$. Correspondingly, the trajectory sequence is temporally segmented into K

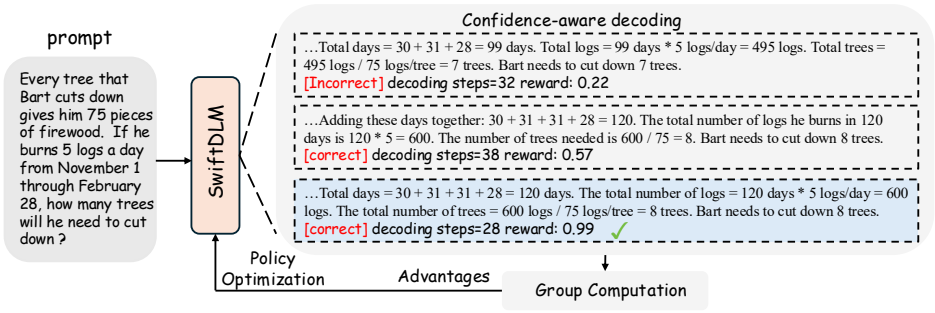


Figure 3: Illustration of self-exploration via reinforcement learning. We assign higher rewards to trajectories that exhibit both accuracy and efficiency within the group, enabling the model to self-explore superior decoding paths.

phases $\{\mathcal{B}_1, \ldots, \mathcal{B}_K\}$, during which the unmasking operation is localized to a single specific block, with each phase having a duration of T_B. Here, the strategy of unmasking one token per step implies that $T = L$ and $T_B = L_B$.

Consistency Distillation. Subsequently, we fine-tune the dLLM on the collected teacher trajectories by adapting Consistency Distillation–initially proposed for continuous domains like image synthesis—to the discrete diffusion process, thereby enabling the model to capture the transition dynamics between distinct trajectory states. To align with the block-wise decoding paradigm, we constrain the distillation training objective within a single block k ($1 \le k \le K$). Within the trajectory segment corresponding to this block, we randomly sample two states $\mathcal{T}_i$ and $\mathcal{T}_j$ satisfying $(k-1)T_B < j < i < kT_B$. The training objective then enforces consistency by aligning the initial state $\mathcal{T}_i$ with the subsequent state $\mathcal{T}_j$. We focus on the tokens that are masked in the initial state $\mathcal{T}_i$ and partition them into two subsets based on $\mathcal{T}_j$: newly-unmasked tokens and remaining-masked tokens, as illustrated in fig. 2. Formally, we denote the index set of newly-unmasked tokens within block k as $\mathcal{U} = \{l \in \mathcal{B}_k \mid \mathcal{T}_i[l] = \mathbf{M}, \mathcal{T}_j[l] \neq \mathbf{M}\}$ and that of remaining-masked tokens as $\mathcal{R} = \{l \in \mathcal{B}_k \mid \mathcal{T}_i[l] = \mathbf{M}, \mathcal{T}_j[l] = \mathbf{M}\}$, where $\mathcal{T}_i[l]$ represents the l-th token of state $\mathcal{T}_i$.

To learn the transition dynamics between trajectory states, we apply a cross-entropy loss to the newly-unmasked tokens $\mathcal{U}$:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{|\mathcal{U}|} \sum_{l \in \mathcal{U}} \log p_\theta(\mathcal{T}_j[l] \mid \mathcal{T}_i), \tag{5}$$

where $\mathcal{T}_j[l]$ denotes the ground-truth token at index l within state $\mathcal{T}_j$. Unlike previous random masking Chen et al. (2025); Nie et al. (2025) strategies, training on the newly-unmasked tokens $\mathcal{U}$ aligns the model with the sequential unmasking order inherent in teacher trajectories. However, $\mathcal{L}_{\text{CE}}$ provides no supervision for the tokens indexed by $\mathcal{R}$ in the initial state $\mathcal{T}_i$. Consequently, to ensure temporal consistency between the states, we introduce a consistency loss targeting these remaining-masked tokens:

$$\mathcal{L}_{\text{Consistency}} = \frac{1}{|\mathcal{R}|} \sum_{l \in \mathcal{R}} \text{KL}\left(\text{sg}(p_\theta(\cdot \mid \mathcal{T}_j)[l]) \parallel p_\theta(\cdot \mid \mathcal{T}_i)[l]\right), \tag{6}$$

where $\text{sg}(\cdot)$ indicates the stop-gradient operator, used to stabilize the training target.

The overall training objective is formulated as:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \beta \mathcal{L}_{\text{Consistency}} \tag{7}$$

where β is a hyperparameter balancing the two terms. This formulation can be also viewed as a combination of black-box distillation ($\mathcal{L}_{\text{CE}}$) and white-box distillation ($\mathcal{L}_{\text{Consistency}}$).

4.2 Self-Exploration via Reinforcement Learning

While the first stage enables skip-step generation through consistency distillation, it confines the model to the teacher's trajectory, limiting further improvements in both efficiency and performance.

To overcome this constraint, we employ reinforcement learning to enable the student model to self-explore optimal decoding paths.

Here we employ Group Relative Policy Optimization (GRPO) Shao et al. (2024) as our training framework, which eliminates the need for a separate critic model by deriving advantages from group statistics, significantly reducing training resources. As illustrated in fig. 3, for each prompt q, we sample G trajectories of varying lengths $\{x_0^g, \ldots, x_{S^g}^g\}_{g=1}^G$ via block-wise confidence-threshold sampling, along with their associated unmasked positions $\{u_1^g, \ldots, u_{S^g}^g\}_{g=1}^G$, where $u_s^g \in \{0,1\}^L$. We then compute the reward R^g for each trajectory, assigning higher rewards to trajectories that produce correct outputs with fewer decoding steps. After obtaining rewards, we define the group-relative advantage A^g by omitting the standard deviation normalization Liu et al. (2025b): $A^g := R^g - \text{mean}(R_{1:G})$.

We further optimize the generation model on newly sampled trajectories using feedback on both response quality and decoding efficiency. To this end, we introduce two key improvements.

First, during the rollout phase, we employ block-wise confidence-threshold dynamic sampling to generate trajectories of varying lengths. Specifically, we divide the sequence into blocks and generate them left-to-right, decoding only tokens that exceed the confidence threshold within each block, similar to the Fast-dLLM. This sampling strategy facilitates the subsequent self-exploration, where we assign higher rewards to trajectories that yield correct outputs with fewer sampling steps.

Second, we evaluate token probabilities conditioned on the intermediate states recorded during generation. At each decoding step s, we retain the newly unmasked positions u_s^g and compute the probability of each selected token given its preceding state x_{s-1}^g:

$$p_\theta(x_s^g[i] \mid x_{s-1}^g) := \pi_\theta(x_s^g[i] \mid x_{s-1}^g), \qquad \text{for } i \text{ such that } u_s^g[i] = 1. \tag{8}$$

This computation retains the intermediate decoding context for each selected token. The trajectory-level reward provides a shared group-relative advantage for policy updates across decoding steps. We optimize the resulting trajectory-conditioned token-level policy using the following clipped surrogate objective:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}_{q\sim\mathcal{D}}\left[\frac{1}{G}\sum_{g=1}^{G}\frac{1}{S^g}\sum_{s=1}^{S^g}\frac{1}{\|u_s^g\|_1}\sum_{i:\, u_s^g[i]=1}\min\left\{r_s^{g,i}A^g, \text{clip}(r_s^{g,i}, 1-\epsilon, 1+\epsilon)A^g\right\}\right] \tag{9}$$

where $r_s^{g,i} = \frac{p_\theta(x_s^g[i]|x_{s-1}^g)}{p_{\theta_{\text{old}}}(x_s^g[i]|x_{s-1}^g)}$ is the token-level likelihood ratio evaluated at the same recorded state and position. Within each policy update, we reuse the intermediate states and unmasking positions recorded in the corresponding rollout to recompute token probabilities, providing a local approximation to trajectory-level policy optimization. The detailed training algorithm is provided in Appendix B.

Balancing Efficacy and Efficiency. To endow the model with both efficacy and efficiency, it is essential to design a reward mechanism that jointly optimizes answer correctness and decoding steps. A common practice involves assigning distinct weights to individual reward components, yielding a weighted sum formulation, such as $r = w_1 r_{\text{correct}} + w_2 r_{\text{format}} + w_3 r_{\text{step}}$, where $r_{\text{step}} = -\frac{S}{N_{\text{step}}}$. In our experiments, the tested weighted-sum formulation reduces decoding steps at the expense of answer accuracy (Appendix F.4). Motivated by multiplicative reward designs Jazbec et al. (2025), we instead scale the combined correctness and format reward by an efficiency factor: $r = (r_{\text{correct}} + r_{\text{format}})(1 - \frac{S}{N_{\text{step}}})^\alpha$. This formulation couples response quality with decoding efficiency, with α controlling the strength of the efficiency incentive. Empirically, it achieves a better accuracy–efficiency trade-off than the tested weighted-sum formulation.

Policy Entropy Dynamics. Prior work on autoregressive models Cui et al. (2025) reveals that the policy entropy $\mathcal{H}$ and performance exhibit a relationship of $R = -a\exp(\mathcal{H}) + b$, demonstrating that the policy trades entropy for rewards. In dLLMs, policy entropy can be employed to characterize the uncertainty of the model's decoding trajectory, defined as follows:

$$\mathcal{H}_{\text{PE}} := \frac{1}{S}\sum_{s=1}^{S}\frac{1}{|u_s|}\sum_{\ell\in u_s}\mathcal{H}\left(\pi_\theta(x_s[l] \mid x_{s-1})\right), \tag{10}$$

Table 1: Comparison of SwiftDLM and baselines on LLaDA-8B-Instruct. All methods use block-wise sampling with generation length 256 and block size 32. Confidence-based methods use a threshold of 0.9. d3LLM is evaluated with the shared decoder. We report decoding steps, latency, speedup, and accuracy.

Benchmark	Method	Steps ↓	Latency↓	Speedup ↑	Accuracy↑
GSM8K-CoT (0-shot)	LLaDA-8B-Instruct	256	9.8s	1.0×	75.6%
	Fast-dLLM	58	2.5s	3.9×	73.6%
	d3LLM	31	1.3s	7.5×	73.1%
	D2F	54	2.4s	4.1×	73.5%
	dParallel	30	1.3s	7.5×	74.7%
	Ours	**25**	**1.1s**	**8.9**×	81.6%
MATH (4-shot)	LLaDA-8B-Instruct	256	24.2s	1.0×	37.9%
	Fast-dLLM	96	5.1s	4.7×	36.6%
	d3LLM	50	4.8s	5.0×	30.4%
	D2F	78	4.6s	5.3×	30.8%
	dParallel	48	4.7s	5.1×	35.7%
	Ours	**43**	**4.3s**	**5.6**×	39.8%
HumanEval (0-shot)	LLaDA-8B-Instruct	256	11.7s	1.0×	40.8%
	Fast-dLLM	90	3.7s	3.2×	37.8%
	d3LLM	34	1.7s	6.9×	39.6%
	D2F	45	2.2s	5.3×	38.0%
	dParallel	31	1.6s	7.3×	40.1%
	Ours	**24**	**1.1s**	**10.6**×	40.4%
MBPP (3-shot)	LLaDA-8B-Instruct	256	24.9s	1.0×	41.8%
	Fast-dLLM	68	3.3s	7.5×	39.0%
	d3LLM	26	2.6s	9.6×	40.8%
	D2F	33	2.3s	10.8×	37.6%
	dParallel	24	2.5s	10.0×	40.8%
	Ours	**21**	**2.2s**	**11.3**×	42.4%

where $x_s[l]$ denotes the l-th token of the s-th intermediate state in the trajectory. In dLLMs, we similarly observe that as model performance and efficiency improve, the uncertainty of the inference trajectory progressively decreases. Recent work on dLLMs, specifically dParallel Chen et al. (2025), leverages entropy loss for distillation-based acceleration. However, it focuses solely on the entropy at individual intermediate states while neglecting the entropy across the entire trajectory. Furthermore, it applies entropy loss to all correctly predicted tokens at a given state, including inherently high-entropy tokens (e.g., "therefore") Qian et al. (2025), which can degrade model performance.

5 Experiments

5.1 Experimental Setup

Implementation Details We conduct experiments on two representative dLLMs: LLaDA-8B-Instruct Nie et al. (2025) and Dream-7B-Instruct Ye et al. (2025). We apply a two-stage training paradigm to both base model. To ensure a fair comparison with dParallel Chen et al. (2025), we adopt the same dataset configuration. Specifically, we utilize approximately 92k samples for LLaDA and 122k samples for Dream, which are sourced from the training sets of GSM8K Cobbe et al. (2021) and PRM12K Lightman et al. (2023), as well as subsets of Numina-Math Li et al. (2024) and AceCode Zeng et al. (2025). Complete training configurations are provided in Appendix C.

In the first stage, teacher trajectories are collected using block-wise sampling ($L_b = 32$) with intermediate states preserved. Subsequently, we fine-tune the model on these trajectories using LoRA Hu et al. (2022) for 5 epochs, with a learning rate of $2e-5$ and $\beta = 1.0$. In the second stage, we train separate models on the math dataset (the GSM8K training set) and code dataset (the AceCode subset) using LoRA , with a learning rate of 1×10^{-6}, rollouts per prompt $G = 6$, and $\alpha = 2.0$. Specifically, the rollouts are generated via the block-wise confidence-threshold decoding strategy.

Table 2: Comparison of SwiftDLM and baselines on Dream-7B-Instruct. All methods use block-wise sampling with generation length 256 and block size 32. Confidence-based methods use a threshold of 0.9. d3LLM is evaluated with the shared decoder. We report decoding steps, latency, speedup, and accuracy.

Benchmark	Method	Steps ↓	Latency↓	Speedup ↑	Accuracy↑
GSM8K-CoT (0-shot)	Dream-7B-Instruct	256	11.7s	1.0×	82.8%
	Fast-dLLM	72	2.7s	4.3×	79.5%
	d3LLM	40	1.5s	7.8×	81.4%
	dParallel	39	1.5s	7.8×	82.3%
	Ours	**31**	**1.2s**	**9.8** ×	83.6%
MATH (0-shot)	Dream-7B-Instruct	256	10.6s	1.0×	39.4%
	Fast-dLLM	91	4.0s	2.7×	36.6%
	d3LLM	63	2.5s	4.2×	38.2%
	dParallel	65	2.6s	4.1×	38.3%
	Ours	**51**	**2.0s**	**5.3**×	39.4%
HumanEval (0-shot)	Dream-7B-Instruct	256	14.2s	1.0×	52.4%
	Fast-dLLM	74	2.8s	5.1×	53.3%
	d3LLM	39	2.2s	6.5×	57.1%
	dParallel	37	2.1s	6.8×	54.3%
	Ours	**28**	**1.6s**	**8.9**×	56.1%
MBPP (0-shot)	Dream-7B-Instruct	256	11.3s	1.0×	58.0%
	Fast-dLLM	45	2.1s	5.4×	54.3%
	d3LLM	24	1.1s	10.3×	55.6%
	dParallel	29	1.3s	8.7×	56.2%
	Ours	**23**	**1.0s**	**11.3**×	57.6%

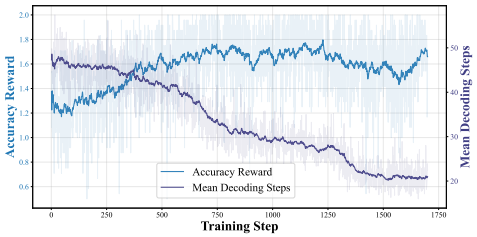


Figure 4: Training dynamics of accuracy reward and mean decoding steps during Stage II reinforcement learning on LLaDA with the GSM8K dataset.

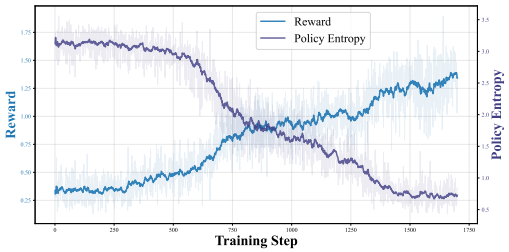


Figure 5: Training dynamics of reward and policy entropy during Stage II reinforcement learning on LLaDA with the GSM8K dataset.

Evaluation Details. We evaluate our model on four widely used benchmarks, including two mathematical reasoning tasks (GSM8K Cobbe et al. (2021), MATH Lewkowycz et al. (2022)) and two code generation tasks (HumanEval Chen (2021), MBPP Austin et al. (2021b)). We assess our model and baseline methods using several metrics, including accuracy, the average number of decoding steps, latency, and speedup ratio. During inference, we set the generation length to 256 and the block size to 32. To ensure a fair comparison, all evaluations are performed on a single GPU of the same type with a fixed batch size of 1.

Baselines. To validate our method's effectiveness, we compare against the vanilla dLLM baseline and state-of-the-art acceleration methods, including the training-free approach Fast-dLLMWu et al. (2025) and distillation-based approaches dParallelChen et al. (2025), d3LLMQian et al. (2026) and D2FWang et al. (2025b). Note that since D2F only provides weights for Dream-7B-Base, which differs from our Dream-8B-Instruct base model, we compare D2F solely on LLaDA-8B-Instruct. Additionally, we evaluate generation quality against RL-based methods on dLLMs, d1 Zhao et al. (2025) and wd1 Tang et al. (2025); detailed results are provided in Appendix D.

5.2 Main Results

Results on the LLaDA Model As shown in table 1, our method achieves a substantial improvement in sampling efficiency across all benchmarks while maintaining strong performance. Specifically, on

Table 3: Ablation studies on the LLaDA model. We report decoding steps, speedup ratio, and accuracy on GSM8K-CoT (0-shot).

(a) Effect of different training strategies.

Stage one		Stage two	GSM8K-CoT (0-shot)		
$\mathcal{L}_{\text{CE}}$	$\mathcal{L}_{\text{Consistency}}$		Steps ↓	Speedup ↑	Acc ↑
✓	✗	✗	53	4.1×	71.9%
✓	✓	✗	42	5.8×	73.5%
✗	✗	✓	31	7.2×	78.2%
✓	✗	✓	30	7.5×	78.8%
✓	✓	✓	25	8.9×	81.6%

(b) Effect of hyperparameter α. **Bold** indicates our proposed setting.

α in reward func.	GSM8K-CoT (0-shot)			
	Steps ↓	Latency	Speedup ↑	Acc ↑
1.0	38	1.6s	6.1×	80.3%
2.0	**25**	**1.1s**	**8.9×**	**81.6%**
3.0	25	1.1s	8.9×	80.1%

mathematical datasets, we further reduce decoding steps compared to the leading distillation baseline dParallel, the training-free method Fast-dLLM, and vanilla LLaDA. Our method also achieves the highest answer accuracy among the compared methods, surpassing the second-best results by 6 and 2 percentage points on GSM8K (81.6% vs. 75.6%) and MATH (39.8% vs. 37.9%), respectively. On code generation benchmarks, our method delivers substantial acceleration while maintaining competitive accuracy, achieving a 10.6× speedup on HumanEval and an 11.3× speedup on MBPP.

Results on the Dream Model As shown in table 2, our method also achieves favorable performance on Dream, which is initialized from an autoregressive model. Within this setting, our method improves decoding efficiency over the compared dLLM acceleration baselines while maintaining generation quality. We observe that, on mathematical datasets, the performance gains on Dream are more modest compared to the substantial improvements observed on LLaDA. For the HumanEval code benchmark, our method delivers an $8.9\times$ speedup, alongside a 3.7 percentage point accuracy boost over vanilla Dream.

Training Dynamics in Stage II As shown in fig. 4, the accuracy reward improves during early training, while decoding steps decrease more substantially after approximately 500 updates. This observed training pattern illustrates improvements in both task performance and decoding efficiency under our reward design.

Additionally, as illustrated in fig. 5, policy entropy decreases as reward increases during training, indicating an empirical association between reduced trajectory uncertainty and improvements in performance and efficiency. Both curves stabilize toward the end of the observed training window. For more experimental results, please refer to Appendix E.

5.3 Ablation Study

Ablation on Training Strategies As shown in table 3a, adding consistency supervision in Stage I reduces decoding steps from 53 to 42 and improves accuracy by 1.6 percentage points on LLaDA's GSM8K-CoT evaluation. With Stage II, it further improves over CE-only initialization, reducing steps from 30 to 25 and increasing accuracy from 78.8% to 81.6%. The full framework achieves the best results, with an $8.9\times$ speedup. We further apply Stage II to a model initialized with dParallel weights. Under the same configuration, our Stage I initialization achieves 3.8 percentage points higher accuracy (81.6% vs. 77.8%) and fewer decoding steps (25 vs. 32); full results are provided in Appendix F.1. Ablation studies on the reward formulation in Stage II are provided in Appendix F.4.

Hyperparameter Analysis. Furthermore, we investigate the impact of hyperparameters on model performance. In Stage II, α controls the strength of the efficiency incentive. As shown in table 3b, $\alpha = 1$ yields modest speedup, while $\alpha = 3$ offers no additional speedup over $\alpha = 2$ and reduces accuracy by 1.5 percentage points. More ablation studies are provided in Appendix F.

6 Conclusion

In this paper, we introduce ***SwiftDLM***, a unified two-stage framework that transitions dLLMs from imitating teacher trajectories to self-exploring optimal decoding paths. Consistency distillation on intermediate states enables skip-step generation, while reinforcement learning further rewards trajectories balancing accuracy and efficiency. Extensive evaluations demonstrate state-of-the-art performance across multiple benchmarks, and we believe SwiftDLM provides an effective foundation for future dLLM acceleration research.

REFERENCES

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

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

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

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

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

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

Zigeng Chen, Gongfan Fang, Xinyin Ma, Ruonan Yu, and Xinchao Wang. dparallel: Learnable parallel decoding for dllms. *arXiv preprint arXiv:2509.26488*, 2025.

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

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

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

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

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. *arXiv preprint arXiv:2505.21467*, 2025.

Metod Jazbec, Theo X Olausson, Louis Béthune, Pierre Ablin, Michael Kirchhof, Joao Monterio, Victor Turrisi, Jason Ramapuram, and Marco Cuturi. Learning unmasking policies for diffusion language models. *arXiv preprint arXiv:2512.09106*, 2025.

Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 1, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Pengxiang Li, Yefan Zhou, Dilxat Muhtar, Lu Yin, Shilin Yan, Li Shen, Yi Liang, Soroush Vosoughi, and Shiwei Liu. Diffusion language models know the answer before decoding. *arXiv preprint arXiv:2508.19982*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025a.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *URL https://arxiv.org/abs/2503.20783*, 2025b.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.

Chen Qian, Dongrui Liu, Haochen Wen, Zhen Bai, Yong Liu, and Jing Shao. Demystifying reasoning dynamics with mutual information: Thinking tokens are information peaks in llm reasoning. *arXiv preprint arXiv:2506.02867*, 2025.

Yu-Yang Qian, Junda Su, Lanxiang Hu, Peiyuan Zhang, Zhijie Deng, Peng Zhao, and Hao Zhang. d3llm: Ultra-fast diffusion llm using pseudo-trajectory distillation. *arXiv preprint arXiv:2601.07568*, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 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.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

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

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. *arXiv preprint arXiv:2507.08838*, 2025.

Chenyu Wang, Paria Rashidinejad, DiJia Su, Song Jiang, Sid Wang, Siyan Zhao, Cai Zhou, Shannon Zejiang Shen, Feiyu Chen, Tommi Jaakkola, et al. Spg: Sandwiched policy gradient for masked diffusion language models. *arXiv preprint arXiv:2510.09541*, 2025a.

Xu Wang, Chenkai Xu, Yijie Jin, Jiachun Jin, Hao Zhang, and Zhijie Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025b.

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

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

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. Acecoder: Acing coder rl via automated test-case synthesis. *arXiv preprint arXiv:2502.01718*, 2025.

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.

Kaiwen Zheng, Yongxin Chen, Hanzi Mao, Ming-Yu Liu, Jun Zhu, and Qinsheng Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.

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

A Derivation of the NELBO for Masked Diffusion Language Models

In this section, we provide the derivation of the Negative Evidence Lower Bound (NELBO) for Masked Diffusion Language Models Shi et al. (2024); Wang et al. (2025a).

We begin by considering discrete-time diffusion models. Given a fixed number of discretization steps T, Masked Diffusion Language Models (MDLMs) gradually corrupt a clean text sequence $\boldsymbol{x}$ into noisy sequences z_t by progressively replacing tokens with a special [mask] token. In this formulation, $z_1 = x$ represents the clean data, while $z_T = \mathbf{M}$ denotes the fully masked state. The forward process is defined as:

$$q(\boldsymbol{z}_t|\boldsymbol{x}) = \text{Cat}(\boldsymbol{z}_t; \alpha_t \boldsymbol{x} + (1-\alpha_t)\boldsymbol{M}) \tag{11}$$

where α_t is a strictly decreasing function of t with $\alpha_1 = 1$ and $\alpha_T = 0$.

To establish the training objective in eq. (3), we begin by constructing the Negative Evidence Lower Bound (NELBO). Let $\boldsymbol{z} = \boldsymbol{z}_{1:T}$ denote a sequence of T latent variables following a Markov chain, where $\boldsymbol{z}_1 = \boldsymbol{x}$ represents the clean data and $\boldsymbol{z}_T = \mathbf{M}$ represents the fully masked state.

$$\begin{aligned}\mathcal{L}_{\text{NELBO}}(\boldsymbol{x};\theta) &= \mathbb{E}_{\boldsymbol{z}\sim q(\cdot|\boldsymbol{x})}\left[\log \frac{q(\boldsymbol{z} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{x},\boldsymbol{z})}\right] \\ &= \mathbb{E}_{\boldsymbol{z}\sim q(\cdot|\boldsymbol{x})}\Bigg[-\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_1) + \log \frac{q(\boldsymbol{z}_T \mid \boldsymbol{x})}{p_\theta(\boldsymbol{z}_T)} + \mathcal{L}_T\Bigg],\end{aligned} \tag{12}$$

where $\mathcal{L}_T = \sum_{t=1}^{T-1} \log \frac{q(\boldsymbol{z}_t|\boldsymbol{z}_{t+1},\boldsymbol{x})}{p_\theta(\boldsymbol{z}_t|\boldsymbol{z}_{t+1})}$. The reconstruction term ($\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_1)$) vanishes since $\boldsymbol{z}_1 = \boldsymbol{x}$ by definition, yielding $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_1) = 1$. The prior matching term ($\log \frac{q(\boldsymbol{z}_T|\boldsymbol{x})}{p_\theta(\boldsymbol{z}_T)}$) equals zero because we assume $\boldsymbol{z}_T = \mathbf{M}$ (fully masked state), implying that both distributions collapse to an indicator function centered at the mask token: $q(\boldsymbol{z}_T \mid \boldsymbol{x}) = p_\theta(\boldsymbol{z}_T) = \mathbf{1}[\boldsymbol{z}_T = \mathbf{M}]$. Consequently, the $\mathcal{L}_{\text{NELBO}}$ simplifies to the following form:

$$\begin{aligned}\mathcal{L}_{\text{NELBO}}(x,\theta) &= \sum_{t=1}^{T-1} \mathbb{E}_{\boldsymbol{z}_t,\boldsymbol{z}_{t+1}\sim q}\left[\log \frac{q(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1})}\right] \\ &= \sum_{t=1}^{T-1} \mathbb{E}_{\boldsymbol{z}_{t+1}\sim q(\cdot|\boldsymbol{x})}\mathbb{E}_{\boldsymbol{z}_t\sim q(\cdot|\boldsymbol{z}_{t+1},\boldsymbol{x})}\left[\log \frac{q(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1})}\right].\end{aligned} \tag{13}$$

Here, $p_{\boldsymbol{\theta}}$ can be parameterized using q and the model prediction $\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{z}_{t+1})$: $p_\theta(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1}) = q(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1}, \boldsymbol{x} = \pi_\theta(\cdot \mid \boldsymbol{z}_{t+1}))$. Under this parameterization of the reverse model, we obtain:

$$\log \frac{q(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1},\boldsymbol{x})}{p_\theta(\boldsymbol{z}_t \mid \boldsymbol{z}_{t+1})} = \begin{cases} 0 & \boldsymbol{z}_t = \boldsymbol{z}_{t+1} = \boldsymbol{x}, \\ -\log \pi_\theta(\boldsymbol{x} \mid \boldsymbol{z}_{t+1}) & \boldsymbol{z}_t = \boldsymbol{x}, \boldsymbol{z}_{t+1} = \mathbf{M}, \\ 0 & \boldsymbol{z}_t = \boldsymbol{z}_{t+1} = \mathbf{M}. \end{cases} \tag{14}$$

Given that $q(\boldsymbol{z}_t = \boldsymbol{x} \mid \boldsymbol{z}_{t+1} = \mathbf{M}, \boldsymbol{x}) = \frac{\alpha_t - \alpha_{t+1}}{1-\alpha_{t+1}}$, the $\mathcal{L}_{\text{NELBO}}$ objective can be further reformulated as:

$$\mathcal{L}_{\text{NELBO}}(\boldsymbol{x};\boldsymbol{\theta}) = -\sum_{t=1}^{T-1} \mathbb{E}_{\boldsymbol{z}_{t+1}\sim q(\cdot|\boldsymbol{x})}\left[\frac{\alpha_t - \alpha_{t+1}}{1-\alpha_{t+1}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_{t+1})\mathbf{1}[\boldsymbol{z}_{t+1} = \boldsymbol{m}]\right]. \tag{15}$$

Taking the limit $T \to \infty$, we have:

$$\mathcal{L}_{\text{NELBO}}(\boldsymbol{x};\boldsymbol{\theta}) = \int_{t=0}^{1} \mathbb{E}_{\boldsymbol{z}_t\sim q(\cdot|\boldsymbol{x})}\left[\frac{\alpha_t'}{1-\alpha_t} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_t)\mathbf{1}[\boldsymbol{z}_t = \mathbf{M}]\right]. \tag{16}$$

When $\alpha_t = 1 - t$, we recover eq. (3).

B TRAINING ALGORITHM IN STAGE II

The GRPO-based reinforcement learning algorithm for the second stage is provided in algorithm 1.

Algorithm 1 Reinforcement Learning in Stage II.

Require: Diffusion LLM model π_θ, dataset $\mathcal{D}$, reward function $\mathrm{R}(x, a, S)$, confidence threshold τ for dynamic sampling, intermediate states of the denoising trajectory $\{x_0^g, \ldots, x_{S^g}^g\}_{g=1}^G$, and the corresponding unmasked positions $\{u_1^g, \ldots, u_{S^g}^g\}_{g=1}^G$.

while π_θ not converged or max epochs unreached **do**
  Sample question-answer pair $(q, a) \in \mathcal{D}$
  for g **in** $1, ..., G$ **do** $\{G$ denotes the number of groups per question$\}$ **do**
    Initialize x_0^g with q and mask tokens
    $s \leftarrow 1$
    while x_s^g has mask tokens $\{$Confidence-threshold sampling$\}$ **do**
      Compute the confidence for each unmasked token $c_{s-1}^g[i] = \max(\text{softmax}(\pi_\theta(\cdot|x_{s-1}^g).\text{logits}))$
      Unmask tokens where $c_{s-1}^g[i] > \tau$ and remask the rest to obtain x_s^g
      Record the unmasked positions as u_s^g
      $s \leftarrow s + 1$
    end while
    Output: Generated response $x_{S^g}^g$ with total denoising steps S^g
    $r^g = \mathrm{R}(x_{S^g}^g, a, S^g)$ $\{$Compute the rewards based on answer and denoising steps$\}$
  end for
  $A^g = r^g - \text{mean}(r^{1:G})$ $\{$Compute the advantages$\}$
  for i **in** $1, \ldots, \mu$ **do** ▷ The number of GRPO iterations **do**
    for g **in** $1, \ldots, G$ **do do**
      for s **in** $1, \ldots, S^g$ **do do**
        Retrieve the intermediate states x_{s-1}^g and unmasked positions u_s^g
        For each i such that $u_s^g[i] = 1$, recompute $p_\vartheta(x_s^g[i] \mid x_{s-1}^g) := \pi_\vartheta(x_s^g[i] \mid x_{s-1}^g)$ for $\vartheta \in \{\theta, \theta_{\text{old}}\}$
        $\mathcal{L}_{\theta,g} = -\frac{1}{|\mathcal{S}^g|} \sum_{s=1}^{S^g} \frac{1}{\|u_s^g\|_1} \sum_{i:\, u_s^g[i]=1} \min\left\{r_s^{g,i} A^g, \text{clip}(r_s^{g,i}, 1-\epsilon, 1+\epsilon) A^g\right\}$, $r_s^{g,i} = \frac{\mathrm{p}_\theta(x_s^g[i]|x_{s-1}^g)}{\mathrm{p}_{\theta_{\text{old}}}(x_s^g[i]|x_{s-1}^g)}$ $\{$Using eq. (9)$\}$
      end for
    end for
    $\mathcal{L}_\theta = \frac{1}{G} \sum_{g=1}^G \mathcal{L}_{\theta,g}$
    Update θ of π_θ with $\nabla_\theta \mathcal{L}_\theta$
  end for
end while
Return π_θ

C MORE IMPLEMENTATION DETAILS

All experiments are conducted on 8 NVIDIA A800 GPUs (80GB). SwiftDLM is a computationally efficient framework, and we report the wall-clock training costs for each stage as follows. Teacher trajectory collection, using the same dataset configuration as dParallel Chen et al. (2025) (92k samples for LLaDA and 122k samples for Dream), requires approximately 12 hours for LLaDA and 14 hours for Dream. Stage I fine-tuning for 5 epochs on the collected trajectories requires approximately 4 hours for LLaDA and 6 hours for Dream. For Stage II, we train separate models for mathematical reasoning and code generation: LLaDA requires approximately 12 hours on math and 4 hours on code, while Dream requires approximately 10 hours on math and 6 hours on code.

For Stage II optimization, we use LoRA with rank 128 and a learning rate of 1×10^{-6}. For each prompt, we sample $G = 6$ trajectories using Gumbel-max sampling with temperature $T = 1.0$, with independent sampling noise providing trajectory diversity. At this temperature, rollout tokens are sampled directly from $\pi_{\theta_{\text{old}}}$, matching the distribution used to compute the token-level likelihood ratios. At each decoding step, positions whose confidence exceeds τ are unmasked; if none exceeds the threshold, the position with the highest confidence is unmasked to ensure decoding progress. Each batch of rollouts is reused for $\mu = 2$ policy updates, during which the recorded intermediate states and unmasking positions are reused as a local approximation to trajectory-level policy optimization. After these updates, new trajectories are sampled from the updated policy, which serves as $\pi_{\theta_{\text{old}}}$ in the next iteration.

D MORE EXPERIMENT RESULTS

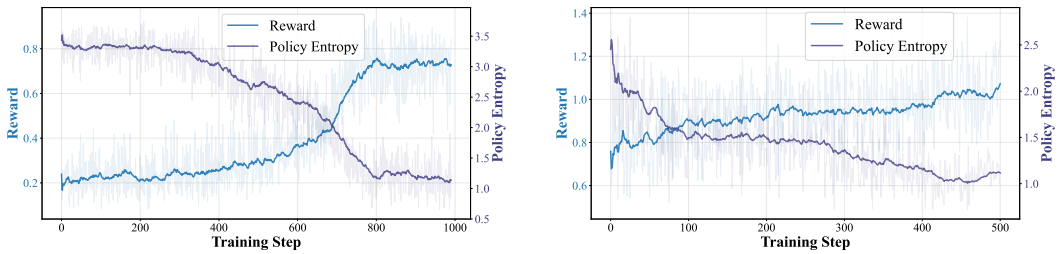
We present a comparison of SwiftDLM against RL-based methods on dLLMs, specifically d1 Zhao et al. (2025) and wd1 Tang et al. (2025), in terms of generation quality. Since several methods do not report results on code benchmarks, we restrict our comparison to mathematical benchmarks, evaluating accuracy and decoding steps. As shown in table 4, SwiftDLM achieves accuracy comparable to d1 and wd1 while substantially reducing decoding steps, demonstrating a significant improvement in inference efficiency.

Table 4: Comparison of SwiftDLM with RL-based methods.

Benchmark	**Method**	**Accuracy ↑**	**Steps ↓**
GSM8K-CoT	d1	81.9%	58
	wd1	82.4%	55
	SwiftDLM	**81.6%**	**25**
MATH	d1	40.1%	94
	wd1	39.2%	89
	SwiftDLM	**39.8%**	**43**

E POLICY ENTROPY AND REWARD

As illustrated in fig. 6, we similarly recorded the dynamic changes in policy entropy and reward for the LLaDA model on the code dataset (the AceCode subset) and for the Dream model on the math dataset (the GSM8K training set). LLaDA was trained on the code dataset for 1000 steps, while the Dream model demonstrated more rapid convergence on the math task, achieving optimal performance at 500 steps. Consistent with our findings in the main text, we also observe an inverse correlation between reward and policy entropy across different models and datasets during the Stage II reinforcement learning.



(a) LLaDA on AceCode subset

(b) Dream on GSM8K training set

Figure 6: Dynamic changes in policy entropy and reward during training. (a) LLaDA trained on the AceCode subset. (b) Dream trained on the GSM8K training set.

F More Ablation Studies

F.1 Stage II with Different Initializations

To examine the effect of initialization on Stage II, we apply the same Stage II training procedure to a dParallel checkpoint. As shown in table 5, Stage II improves the accuracy of dParallel from 74.7% to 77.8%, while the average number of decoding steps slightly increases from 30 to 32. In comparison, initializing Stage II from our consistency-distilled Stage I achieves both higher accuracy and fewer decoding steps.

Table 5: Effect of Stage II with different initializations on LLaDA using GSM8K-CoT (0-shot). All Stage II variants use the same training data, reward formulation, and optimization configuration.

Initialization	**Stage II**	**Steps** $\downarrow$	**Acc** $\uparrow$
dParallel checkpoint	✗	30	74.7%
dParallel checkpoint	✓	32	77.8%
SwiftDLM Stage I	✗	42	73.5%
SwiftDLM Stage I	✓	**25**	**81.6%**

dParallel employs a certainty-forcing objective that minimizes the predictive entropy of correctly predicted masked tokens, encouraging multiple positions to reach high confidence in parallel Chen et al. (2025). Its checkpoint therefore provides Stage II with a more concentrated initial policy, which may limit the diversity of decoding trajectories available for further exploration. This offers one possible explanation for why Stage II improves answer accuracy on dParallel but provides no additional reduction in decoding steps. Under this setting, our consistency-distilled initialization yields a better final accuracy–efficiency trade-off after Stage II.

F.2 Additional Hyperparameter Analysis

Here we investigate the effect of the hyperparameter β on consistency in Stage I. The hyperparameter β balances the consistency loss and entropy loss. As shown in table 6, we find that dLLM achieves optimal inference efficiency and generation performance when $\beta = 1$. This is because consistency distillation for aligning intermediate states $\mathcal{T}_i$ and $\mathcal{T}_j$ requires both a hard constraint from the decoded tokens on newly-unmasked positions and a soft constraint from the model-predicted logits on remaining-masked positions. The best results are obtained when these two components are weighted equally.

F.3 Sensitivity Analysis of Confidence Threshold

The confidence threshold τ is a key inference hyperparameter that controls the trade-off between generation quality and decoding efficiency. Following prior works Chen et al. (2025); Wang et al. (2025b); Wu et al. (2025), we adopt $\tau = 0.9$ as the default setting. To evaluate the robustness of

Table 6: Ablation study of the hyperparameter β in the LLaDA model. We report the number of decoding steps, latency, speedup ratio, and accuracy on the GSM8K-CoT (0-shot) benchmark. **Bold** indicates our proposed setting.

β in consistency distillation	GSM8K-CoT (0-shot)			
	Steps ↓	Latency	Speedup ↑	Acc ↑
0.5	50	2.3s	4.3×	72.1%
1.0	**42**	**1.7s**	**5.8×**	**73.5%**
2.0	47	2.0s	4.9×	70.3%

SwiftDLM to this choice, we compare SwiftDLM and dParallel across a range of confidence thresholds on the GSM8K-CoT benchmark using LLaDA. As shown in table 7, SwiftDLM consistently outperforms dParallel in both accuracy and decoding efficiency across all threshold values, demonstrating that our method achieves a superior efficiency–accuracy trade-off and is robust to the choice of confidence threshold.

Table 7: Sensitivity analysis of the confidence threshold τ for SwiftDLM and dParallel on GSM8K-CoT (0-shot) using LLaDA. We report accuracy and the average number of decoding steps across varying thresholds.

Method	Metric	0.95	0.90	0.85	0.80	0.75	0.70	0.65	0.60	0.55	0.50
SwiftDLM	Acc	81.8%	**81.6%**	81.4%	79.8%	78.2%	78.1%	76.4%	75.4%	70.8%	68.3%
	Steps	31	**25**	24	22	20	19	18	17	16	16
dParallel	Acc	76.4%	74.7%	72.5%	71.5%	70.8%	68.4%	66.6%	62.6%	59.7%	55.3%
	Steps	38	30	27	25	24	23	22	21	20	19

F.4 Effect of Reward Formulation in Stage II

We compare our multiplicative reward with the weighted-sum formulation, both with and without Stage I pretraining. As shown in table 8, the tested weighted-sum formulation achieves fewer decoding steps but substantially lower accuracy. The multiplicative formulation provides a better accuracy–efficiency trade-off under these settings. Combining it with Stage I yields the highest accuracy among the evaluated configurations, while also reducing decoding steps relative to using the same reward without Stage I.

Table 8: Ablation study of reward formulation and training stages on LLaDA. We report decoding steps, speedup ratio, and accuracy on GSM8K-CoT (0-shot). **Bold** indicates our proposed setting.

Stage I	Stage II	Reward Type	Steps ↓	Speedup ↑	Acc ↑
✗	✓	Weighted sum	28	8.1×	68.6%
✗	✓	Multiplicative	31	7.2×	78.2%
✓	✓	Weighted sum	22	9.6×	69.2%
✓	✓	**Multiplicative (ours)**	**25**	**8.9×**	**81.6%**

G Limitations and Future Work.

While SwiftDLM achieves promising results, several limitations remain. In Stage II, the accuracy reward is inherently task-specific, which may constrain the generalizability of our approach. Future work will explore more domain-agnostic reward formulations, such as learned reward models, to broaden applicability. Furthermore, due to computational resource constraints, our current experiments are limited to relatively small-scale models. We anticipate that scaling SwiftDLM to substantially larger models, such as LLaDA 2.0 (100B), will further validate the scalability of our framework.

H LLM USAGE

This section describes the role of large language models (LLMs) in preparing this manuscript. LLMs were used exclusively for writing assistance and language refinement, helping improve clarity, grammatical accuracy, and academic style while preserving the authors' original research contributions, methodological approaches, and scientific insights. Specifically, these tools assisted in refining sentence structure, improving the readability of technical content, adjusting academic terminology, and maintaining a consistent writing style.

LLMs were not used for research ideation, conceptual development, literature review, citation discovery, data analysis, experimental design, or the formulation of research hypotheses and conclusions. All research ideas, experiments, analyses, and scientific conclusions presented in this paper arose from the authors' independent intellectual work. LLM assistance was restricted to linguistic improvements and does not constitute a contribution warranting authorship. The authors take full responsibility for all content, including any text that was refined with LLM assistance, ensuring that the core intellectual contributions and scientific merit of this work remain wholly attributable to the listed authors.