TA-GRPO-*d*: Trajectory-Aware GRPO for Optimizing Denoising Trajectories in Diffusion LLMs

Gyunyeop Kim and Sangwoo Kang*
Department of Computing, Gachon University
{gyop817, swkang}@gachon.ac.kr

Abstract

Diffusion large language models (dLLMs) generate text by repeatedly unmasking a partially noised sequence in parallel, promising lower latency than autoregressive decoding. However, most discrete dLLMs still rely on fixed denoising schedules, which are non-adaptive to input difficulty and cannot learn efficient unmasking orders. This paper introduces a reinforcement learning (RL) framework that transforms dLLM decoding into a trajectory-aware, learnable policy. We propose a confidence-gated denoising strategy that dynamically decides which tokens to unmask and how many to unmask per step, enabling adaptive exploration of denoising trajectories. Building on Group Relative Policy Optimization, we reformulate it into a trajectory-aware variant, TA-GRPO-*d*, which combines a trajectory-level signal—captured as the z-score of the AUC over intermediate rewards—with a token-level unmasking-time weight. This design allows the model to learn not only the final output quality but also the efficiency of the decoding path itself. Experiments on MATH-500, Countdown, Sudoku, and code benchmarks (HumanEval, MBPP) show that TA-GRPO-*d* maintains or improves accuracy while reducing average denoising steps by up to half, achieving both faster inference and lower computational cost. Our approach provides an RL framework for optimizing dLLM decoding policies toward adaptive, efficient reasoning. Code is available at our GitHub[1].

1 Introduction

Diffusion Large Language Models (dLLMs) have emerged as a promising alternative to mitigate the high inference cost caused by token-by-token sequential generation in conventional autoregressive (AR) LLMs. Models that couple a Transformer decoder (Vaswani et al., 2017) with an AR policy—such as LLaMA (Touvron et al., 2023) and GPT (OpenAI et al., 2024)—generate exactly one token per step at inference, so the generation cost scales with sequence length and incurs latency and energy costs with social and environmental implications.

In contrast, dLLMs generate text by performing parallel denoising from a fully noised sequence. Recent studies report that dLLMs can achieve several-fold speedups while maintaining quality comparable to AR LLMs (Li et al., 2025). According to industry reports, Gemini Diffusion (DeepMind, 2024) has achieved throughput on the order of thousands of tokens/second while delivering performance comparable to AR LLMs.

In particular, mask token–based discrete dLLMs (Austin et al., 2023) start from a fully masked target sequence and, at each denoising step, selectively unmask only a subset of masked positions. Unlike AR models, which impose a "one next token per step" constraint, discrete dLLMs admit diverse denoising trajectories depending on both which masks to unmask at each step (position selection) and how many to unmask in parallel (the per-step unmasking budget). This unmasking order and budget directly affect not only the final sequence quality but also the total number of steps (i.e., inference time). Nonetheless, many existing discrete dLLMs (Gong et al., 2025; Liu et al., 2025a; Nie et al., 2025) fix the total number of denoising steps and determine the per-step unmasking budget by a predefined schedule, resulting in a policy that is non-adaptive to input difficulty. For example, if 256 masked tokens are to be resolved in 128 denoising steps, the schedule unmasks exactly 2 [MASK] tokens per step. Consequently, inference speed is effectively tied to the preset step count, and there is little guarantee that the chosen trajectory is optimal.

Accordingly, we convert dLLM denoising from

*Corresponding author.

[1] https://github.com/KimGyunYeop/TA-GRPO-d_re_implementation

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 37159–37174

a fixed schedule into a trajectory-aware, learnable policy trained with reinforcement learning. Concretely, we (i) introduce confidence-gated denoising that dynamically decides, at each step, both the unmasking positions and the per-step unmasking budget based on confidence, enabling exploration over diverse trajectories; (ii) construct a trajectory-level signal by group-normalizing the area under the intermediate-reward curve (AUC) of candidate trajectories so as to favor trajectories that attain high rewards in fewer denoising steps; and (iii) define a trajectory-aware advantage by combining this signal with a token-wise unmasking-time weight. We propose TA-GRPO-*d* to reformulate GRPO (Zhihong Shao, 2024) with this advantage, while keeping trajectory optimization aligned with the final task objective.

This design directly optimizes the quality and efficiency of the denoising trajectory, which prior GRPO-based methods—relying only on ex-post evaluation of final outputs without explicit credit to intermediate states—could not learn. Experiments on math, puzzle, and code benchmarks show that our method maintains or improves accuracy while significantly reducing the average number of denoising steps. The framework couples non-invasively with different models/architectures and explicitly targets inference latency/compute cost, thereby substantially extending the practical efficiency frontier of dLLM inference. We demonstrate on MATH-500, Countdown, Sudoku, and code tasks (HumanEval, MBPP) that the proposed method improves accuracy while simultaneously reducing the average number of denoising steps.

2 Preliminaries

2.1 Denoising Process in Discrete dLLM

We address denoising trajectory optimization in discrete dLLMs that generate the final output by progressively unmasking a masked sequence. Let a token sequence of length L be denoted by $s \in \mathcal{V}^L$, the vocabulary by $\mathcal{V}$, and the mask token by $[\mathrm{MASK}]$. We denote the state at denoising step $t = 0, 1, 2, \ldots$ by s_t, and define the set of masked positions as follows.

$$\mathcal{M}(s_t) = \{i \in \{1, \ldots, L\} : s_{t,i} = [\mathrm{MASK}]\} \tag{1}$$

The initial state s_0 is a noised sequence in which the target span is fully masked, and at each step t selectively unmask only a subset of $\mathcal{M}(s_t)$.

Per-denoising step distribution. At state s_t of step t, for each masked position $i \in \mathcal{M}(s_t)$, the dLLM predicts the token distribution to which the $[\mathrm{MASK}]$ token will be unmasked, defined as

$$p_{\theta,i}(v \mid s_t), \quad v \in \mathcal{V} \tag{2}$$

At positions i selected for unmasking, we sample $s_{t+1,i} \sim p_\theta(\cdot \mid s_t)$, while positions not selected keep their values $s_{t+1,i} = s_{t,i}$. Here, the index set of positions chosen for unmasking by the denoising strategy is denoted by $\mathcal{A}_t \subseteq \mathcal{M}(s_t)$.

$$s_{t+1,i} \sim \begin{cases} p_{\theta,i}(\cdot \mid s_t), & i \in \mathcal{A}_t \\ \delta_{s_{t,i}}(\cdot), & i \notin \mathcal{A}_t \end{cases} \tag{3}$$

Here, $\delta_{s_{t,i}}(\cdot)$ denotes the unit point mass at $s_{t,i}$ (value kept). The policy that determines $\mathcal{A}_t$ depends on the denoising strategy. In prior dLLMs, one typically unmasks top-N confidence tokens according to a fixed schedule. Our denoising strategy is described in Sec. 3.1; in this section we only define the model's basic probabilistic structure. This process is repeated until the fixed schedule completes or all $[\mathrm{MASK}]$ tokens have been denoised, yielding the final sequence.

Confidence. In the denoising strategy of dLLMs, the confidence at each masked position used to decide unmasking is defined as the top-1 probability predicted by the model.

$$c_i(p_{\theta,i}(\cdot \mid s_t)) = \max_{v \in \mathcal{V}} p_{\theta,i}(v \mid s_t) \in [0, 1] \tag{4}$$

Denoising Stopping time. The denoising process terminates once all masks have been removed. The stopping time $\tau(s_0)$ denotes the total number of denoising steps required for a single sample and is defined as

$$\tau(s_0) = \min\{t \geq 0 : \mathcal{M}(s_t) = \emptyset\} \tag{5}$$

3 Proposed Method

We propose a reinforcement learning approach that enables a dLLM to explore diverse trajectories and learn to select better denoising trajectories. To this end, we design a denoising strategy that, based on confidence, decides which mask tokens to unmask and how many to unmask given the current sequence state and the model's capability; during training, we introduce a stochastic unmasking

mechanism to encourage exploration over trajectories. We then augment the GRPO-based approach to develop a reinforcement learning algorithm that drives the dLLM to explore diverse trajectories and to learn the unmasking order directly.

3.1 Confidence-Gated Denoising Strategy

RL Training. During reinforcement learning, we employ a stochastic denoising strategy to enable exploration over diverse trajectories. To select a stable denoising trajectory, at each denoising step t with noised sequence s_t, we compute the confidence $c_i(p_{\theta,i}(\cdot \mid s_t))$ from the model's per-position token distribution $p_{\theta,i}(\cdot \mid s_t)$ and probabilistically unmask tokens among those whose confidence exceeds the training threshold t_{tra}. Accordingly, the probability that a mask token at position i is unmasked is given by

$$m_i(s_t) = \left(\frac{\max\big(0,\ c_i(p_{\theta,i}(\cdot \mid s_t)) + \xi - t_{\text{tra}}\big)}{1 - t_{\text{tra}}} \right)^{\gamma} \tag{6}$$

where $\xi \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise for generating diverse candidates, and the resulting gating probability is hard-clipped to $[0, 1]$; $\gamma > 0$ controls the sharpness of the probability distribution, governing the exploration–stability trade-off during reinforcement learning.

Inference. At inference time, for experimental consistency and reproducibility we use a deterministic threshold. We unmask all tokens whose confidence is at least t_{inf}. The probability that a mask token at position i is unmasked at inference is given by

$$m_i(s_t) = \begin{cases} 1, & c_i(p_{\theta,i}(\cdot \mid s_t)) - t_{\text{inf}} \geq 0 \\ 0, & c_i(p_{\theta,i}(\cdot \mid s_t)) - t_{\text{inf}} < 0 \end{cases} \tag{7}$$

This setup lets the model decide both the unmasking positions and the per-step unmasking budget according to its denoising confidence, so that inference time adapts to input difficulty and model capability. Accordingly, under the proposed denoising strategy, the active set of indices to be unmasked at each denoising step t, $\mathcal{A}_t$, is defined as follows.

$$u_{t,i} \sim \text{Bernoulli}\big(m_i(s_t)\big) \tag{8}$$

$$\mathcal{A}_t \ = \ \big\{\, i \in \mathcal{M}(s_t) \ : \ u_{t,i} = 1 \,\big\} \tag{9}$$

To prevent a stall step in which no tokens are unmasked ($\mathcal{A}_t = \emptyset$), we apply a fallback, if no token would be unmasked, we unmask the mask token with the top-1 confidence.

3.2 Trajectory-Aware GRPO in dLLM

In this section, under the denoising strategy of Sec. 3.1, we propose a reinforcement learning method that updates the policy to enable the model to explore diverse trajectories and select an optimal denoising trajectory. The proposed method augments the GRPO-based approach so that the model uses a trajectory-level, reward-based advantage for denoising trajectories, thereby allowing the model to learn not only the final outputs but also the denoising trajectory itself. The key idea is to (i) sample multiple candidate trajectories for each input, (ii) aggregate reward signals not only on the final output but also along intermediate (partially denoised) states, and (iii) update the policy with token-wise weighted advantages that incorporate unmasking-time information to measure each token's contribution to the final sequence.

3.2.1 Candidate Trajectory Sampling

For reinforcement learning, we sample n distinct denoising trajectories for a single input. Each candidate $j \in 1, \ldots, n$ starts from the same initial state $s_0^{(j)}$ (a sequence whose target span is masked), and is defined as a Markov process that unfolds at each step t according to the RL training denoising policy $u_{t,i}$ and the token distribution $p_{\theta,i}(\cdot \mid s_t^{(j)})$. The one-step transition kernel K_θ^u and the full trajectory of candidate (j) are given as follows.

$$K_\theta^u(s_{t+1} \mid s_t) = \prod_{i \notin \mathcal{M}(s_t)} \delta_{s_{t,i}}(s_{t+1,i}) \prod_{i \in \mathcal{M}(s_t)} \big[\, (1 - u_{t,i}) \delta_{s_{t,i}}(s_{t+1,i}) + u_{t,i} p_{\theta,i}(s_{t+1,i} \mid s_t) \big] \tag{10}$$

$$s_{t+1}^{(j)} \sim K_\theta^u(\cdot \mid s_t^{(j)}), \quad t = 0, 1, \ldots, k_j - 1 \tag{11}$$

$k_j = \tau(s_0^{(j)})$ is the stopping time of candidate (j)'s denoising trajectory. The final output at the stopping time when all [MASK] tokens have been denoised is $S^{(j)} := s_{k_j}^{(j)}$, and the sequence of intermediate states along the denoising process is denoted by $\{s_t^{(j)}\}_{t=0}^{k_j}$.

When a *fallback* is applied that forcibly unmasks the top-confidence position to prevent a stall step ($\mathcal{A}_t = \emptyset$), one may interpret the process as using the transition kernel conditioned on $\mathcal{A}_t \neq \emptyset$; for simplicity, however, we still denote it as K_θ^u in the main text. Consequently, we obtain $S = \{S^{(1)}, \ldots, S^{(n)}\}$ and, for each candidate, the sequence of intermediate states $s_t^{(j)}$, thereby forming the candidate set for reinforcement learning.

3.2.2 Trajectory-Aware Advantages

We augment the GRPO-based approach that updates the policy by comparing candidates within the same group. The proposed method uses not only signals from the final denoised outputs but also signals from the denoising trajectories to assess relative quality among trajectories. Moreover, we structure the objective so that the model can directly learn the trajectory itself.

First, the reward for each final output is defined as $R^{(j)} = f^R(S^{(j)})$. The function f^R is a reward function for evaluating outputs in downstream tasks, configured to match each task's objective. The specific f^R used for the datasets in our experiments is detailed in Appendix C.

To group-center the n candidates sampled for the same input, the advantage for the final outputs is computed as

$$A^{(j)} = R^{(j)} - \frac{1}{n}\sum_{m=1}^{n} R^{(m)} \tag{12}$$

This reduces variance within the same input and provides a baseline for policy updates. However, final rewards alone make it difficult to determine which denoising trajectory is preferable; therefore, we also apply the reward function to the intermediate states $s_t^{(j)}$ of candidate (j) to collect intermediate rewards $r_t^{(j)} = f^R\big(s_t^{(j)}\big)$ and derive a reward curve over denoising steps. Using the same reward function at both intermediate and final states keeps trajectory optimization aligned with the final task objective. Although intermediate states may not always fully satisfy the final evaluation criteria, aggregating these rewards over the denoising process provides a practically meaningful trajectory-level signal. We then use the area under this intermediate-reward curve (AUC) as a trajectory-level score for the denoising trajectory of candidate (j). The AUC is computed as follows.

$$\hat{r}_t^{(j)} = \begin{cases} r_t^{(j)}, & t \le k_j \\ r_{k_j}^{(j)}, & t > k_j \end{cases} \qquad t = \{1, \ldots, K_{\max}\} \tag{13}$$

$$d^{(j)} = \sum_{t=1}^{K_{\max}} \hat{r}_t^{(j)} \tag{14}$$

$K_{\max} := \max_m k_m$ denotes the maximum, over all candidates for a single input, of their stopping steps. Since the stopping step differs across candidates, we pad each trajectory with the final reward to match lengths and then compute the area under the intermediate-reward curve (AUC). This padding is intentional: if one uses a simple average over intermediate rewards, trajectories with comparable final quality but different stopping times can receive similar scores, whereas a simple sum without padding can systematically favor slower trajectories with more denoising steps. By contrast, AUC with padding assigns a larger score to trajectories that attain high rewards in fewer denoising steps. For comparison stability, we then take a z-score over the AUC values d across the candidate group to obtain a trajectory-level signal $\zeta^{(j)}$, which indicates how good candidate (j)'s trajectory is, on average, over the intermediate process while favoring reward-efficient trajectories.

$$\zeta^{(j)} = \frac{d^{(j)} - \mu(d)}{\sigma(d)} \tag{15}$$

However, because this is an ex-post evaluation of the trajectory, applying it directly in reinforcement learning does not allow the model to learn the trajectory itself. Therefore, we directly convey information about the action order by using when each token along the trajectory was unmasked. This plays a role analogous to the discount factor in classical reinforcement learning.

We use, as a weight on the advantage, the number of remaining $[\mathrm{MASK}]$ tokens at the time each token on the trajectory is unmasked. In other words, tokens unmasked earlier (when more masks remain globally) are interpreted as having greater influence on subsequent evolution and thus receive larger weights. This enables the model to learn the denoising trajectory directly through reinforcement learning. Let $o_i^{(j)}$ denote the unmasking time of token i; then the unmasking-time weight is defined as follows.

Algorithm 1 Training: Trajectory-Aware GRPO and Stochastic Confidence-Gated Denoising

Require: Dataset $\mathcal{D}$, reward f^R, params θ; #candidates n; PPO hparams (ε, β), KL loss $\mathcal{L}^{\mathrm{KL}}$

for input $q \sim \mathcal{D}$ **do**
  for $j = 1..n$ **do**
    $s_0^{(j)} \leftarrow \textsc{MaskInput}(q)$; $t \leftarrow 0$
    while $\mathcal{M}(s_t^{(j)}) \neq \emptyset$ **do**
      For each $i \in \mathcal{M}(s_t^{(j)})$: Get Denoising token index set $\mathcal{A}_t$ via $p_{\theta,i}(\cdot \mid s_t^{(j)})$(see Denoising Strategy of Sec. 3.1)
      Unmask tokens using $p_{\theta,i}(\cdot \mid s_t^{(j)})$ at $i \in \mathcal{A}_t$; copy others unchanged; $o_i^{(j)} \leftarrow t$ at $i \in \mathcal{A}_t$; $t \leftarrow t{+}1$
      Intermediate reward $r_t^{(j)} = f^R(s_t^{(j)})$ and $M_t^{(j)} = |\mathcal{M}(s_t^{(j)})|$
    end while
    $k_j \leftarrow t$; $S^{(j)} \leftarrow s_{k_j}^{(j)}$; $R^{(j)} \leftarrow f^R(S^{(j)})$
  end for
  $K_{\max} = max(k_j) : j = 1...n$
  for $j = 1..n$ **do**
    Grouped rewards $A^{(j)} = R^{(j)} - \frac{1}{n}\sum_{m=1}^{n} R^{(m)}$
    AUC $d^{(j)} = \sum_{t=1}^{K_{\max}} r^{(j)}_{\min(t,\, k_j)}$
    z-score $\zeta^{(j)} = \frac{d^{(j)} - \mu(d)}{\sigma(d)}$
    For each $i \in \mathcal{M}(s_0^{(j)})$: $w_i^{(j)} = |\mathcal{M}(s^{(j)}_{o_i^{(j)}})|/|\mathcal{M}(s_0^{(j)})|$; $A_i'^{(j)} = \big(1 + \Delta\zeta^{(j)} w_i^{(j)}\big) A^{(j)}$
    Estimate per-token ratios $\rho_i^{(j)}(\theta)$ at $s_0^{(j)}$ using one-step per-token likelihood (see *Policy Optimization*, Eq. (18))
  end for
  Compute the objective $\mathcal{L}^{\mathrm{clip}}(\theta)$ from $\rho(\theta), A'$ via Eq. (21) (clipped PPO Loss)
  update $\theta \leftarrow \mathrm{OptimizerStep}(\nabla_\theta - L^{\mathrm{clip}} + \beta\mathcal{L}^{\mathrm{KL}})$
end for

$$w_i^{(j)} = \frac{|\mathcal{M}(s^{(j)}_{o_i^{(j)}})|}{|\mathcal{M}(s_0^{(j)})|} \in [0, 1] \tag{16}$$

$|\mathcal{M}(s_t^{(j)})|$ denotes the number of masks remaining immediately after denoising step t. A formal derivation of $o_i^{(j)}$ is provided in Appendix B.

Finally, we combine the candidate-level signal $\zeta^{(j)}$ with the token-level unmasking-time weight $w_i^{(j)}$ and a scaling coefficient Δ to control the strength of trajectory-aware shaping, thereby computing a weighted advantage for token i of candidate (j).

$$A_i'^{(j)} = \big(1 + \Delta\zeta^{(j)} w_i^{(j)}\big) A^{(j)} \tag{17}$$

Here, $A_i'^{(j)}$ reweights token-wise updates to reflect not only the final reward but also the trajectory quality ($\zeta^{(j)}$), the influence of the unmasking time ($w_i^{(j)}$), and the shaping strength controlled by Δ. As a result, (i) high-reward, high-quality trajectories are reinforced more strongly; (ii) high-reward but inefficient trajectories are moderated; and (iii) seemingly high-quality trajectories that actually converge to incorrect solutions (low reward) are suppressed earlier and more aggressively. This shaping drives the policy to converge toward trajectories that reduce the average number of denoising steps while maintaining or improving accuracy.

3.2.3 Policy Optimization

We use the trajectory-aware, token-wise shaped advantage A' as the training signal in PPO (Schulman et al., 2017) to update the policy so that it prefers high-quality trajectories. For policy optimization, following Zhao et al. (2025), we compute the likelihood of a candidate sequence under a mean-field approximation that assumes independence across masked positions—i.e., a one-step per-token likelihood—and perform clipped PPO updates based on per-token importance ratios. Specifically, for the noised input $s_0^{(j)}$ and a token index $i \in \mathcal{M}(s_0)$ that is actually unmasked, the per-token importance ratio is defined as

$$\rho_i^{(j)}(\theta) = \frac{p_{\theta,i}\big(s^{(j)}_{k_j,i} \big| s_0^{(j)}\big)}{p_{\theta_{\mathrm{old}},i}\big(s^{(j)}_{k_j,i} \big| s_0^{(j)}\big)} \tag{18}$$

Here, $p_{\theta,i}$ denotes the current policy, and $p_{\theta_{\mathrm{old}},i}$ denotes the previous policy used for update stability. We then define the objective via the PPO *clipped* term and, for training stability, add a KL term against a reference policy to obtain the final objective $\mathcal{L}_{\mathrm{PPO}}$.

$$\mathcal{L}^{\mathrm{clip}}_{(j),i}(\theta) = \min\Big(\rho_i^{(j)}(\theta) A_i'^{(j)}, \; \mathrm{clip}\big(\rho_i^{(j)}(\theta), 1-\varepsilon, 1+\varepsilon\big) A_i'^{(j)}\Big) \tag{19}$$

$$\mathcal{L}^{\mathrm{KL}}(\theta) = D_{\mathrm{KL}}\big(p_\theta(\cdot \mid s_0) || p_{\mathrm{ref}}(\cdot \mid s_0)\big) \quad (20)$$

$$\mathcal{L}_{\mathrm{PPO}}(\theta) = -\frac{1}{n}\sum_{j=1}^{n}\frac{1}{|\mathcal{M}(s_0)|}\sum_{i\in\mathcal{M}(s_0)}\mathcal{L}^{\mathrm{clip}}_{(j),i}(\theta) + \beta\mathcal{L}^{\mathrm{KL}}(\theta) \quad (21)$$

The hyperparameters ε and β denote the clip width and the KL weight, respectively.

Accordingly, Trajectory-Aware GRPO (i) generates candidate trajectories via stochastic confidence-gated denoising, (ii) constructs a trajectory-aware token-wise advantage A' by combining the final reward, the z-scored area under the intermediate-reward curve (AUC), and the unmasking-time weight, and (iii) updates the policy with PPO using a one-step per-token likelihood approximation. This enables the model to adapt its unmasking budget and order to input difficulty and to learn a preference for better trajectories with fewer steps. The proposed reinforcement learning procedure is summarized in Algorithm 1.

4 Experiment

4.1 Experiment Setups

Datasets. We evaluate on five standard datasets across two domains: **Math & Puzzle**—MATH-500 (Lightman et al., 2023; Hendrycks et al., 2021), Countdown (Pan et al., 2025), and 4×4 Sudoku; and **Code**—HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). We report answer accuracy (MATH-500, Countdown), blank-level accuracy (Sudoku), and Pass@1 (HumanEval, MBPP).

Baselines. We evaluate two large open-source dLLMs, LLaDA-8B (Nie et al., 2025) and Dream-7B (Ye et al., 2025). As baselines, we report (i) each pre-trained model with its original denoising strategy and (ii) the same models with our strategy (Sec. 3.1). Prior work uses fixed Top-k schedules evenly distributed over a preset number of steps; for example, to denoise 256 tokens in 128 steps, the schedule unmasks the Top-2 confidence [MASK] tokens at each step. We also compare against *diffu*-GRPO (Zhao et al., 2025), which adapts GRPO to the dLLM setting.

Settings. We evaluate each benchmark after train for 5,000 updates on **Math & Puzzle** and 3,500 on **Code**, using semi-autoregressive decoding. Following Zhao et al. (2025), we employ prompt masking (`num iteration=12`). Our denoising strategy uses $\gamma = 0.5$ with thresholds $t_{\mathrm{tra}} = 0.7$ and $t_{\mathrm{inf}} = 0.8$. For the trajectory-aware shaping coefficient, we use $\Delta = 0.5$ on MATH-500 and **Code**, and $\Delta = 1.0$ on the **Puzzle** tasks (Countdown and Sudoku). For **Code**, we perform RL on Kodcode and evaluate on HumanEval and MBPP.

Further details on datasets and experimental settings are provided in Appendices D and E.

4.2 Main Result

Across both domains, TA-GRPO-*d* consistently reduces the average number of denoising steps while maintaining or improving accuracy, whereas the threshold-only Gated setting can be unstable depending on task difficulty. Fixed Top-2 schedules remain non-adaptive.

Math & Puzzle (Table 1). On LLaDA-8B, TA-GRPO-*d* reduces steps on MATH-500 from 128 to about 71 with accuracy essentially unchanged, and delivers large gains on Countdown and Sudoku while also shortening denoising. Dream-7B shows the same pattern (lower steps with comparable or higher accuracy). Combining a trajectory-level signal with an unmasking-time weight is key to improving both accuracy and efficiency. TA-GRPO-*d* also outperforms *diffu*-GRPO across datasets (with a minor trade-off on LLaDA-8B MATH-500), indicating that learning better trajectories can improve final quality as well as speed.

Code (Table 2). On LLaDA-8B, TA-GRPO-*d* achieves the best scores on HumanEval and MBPP with the fewest steps, requiring about half as many steps as the fixed top-2 schedule. This confirms benefits for both mathematical reasoning and code generation.

By internalizing the selection of efficient denoising trajectories, TA-GRPO-*d* lowers denoising steps with comparable or higher accuracy, yielding practical latency and compute savings for dLLM inference.

5 Analysis

5.1 t_{inf} Sweep

With t_{tra} fixed, increasing t_{inf} improves accuracy while also increasing the average number of denoising steps up to a knee point; beyond that point, accuracy saturates (or slightly declines) while the

Model	Method	Denoising Strategy	MATH-500		Countdown		Sudoku	
			Score (↑)	Avg.DS (↓)	Score (↑)	Avg.DS (↓)	Score (↑)	Avg.DS (↓)
LLaDA-8B	pre-trained	Top-2	**34.00**	128.00	14.84	128.00	6.64	128.00
	pre-trained	Gated	33.20	84.00	21.48	128.84	7.81	135.10
	diffu-GRPO	Top-2	33.80	128.00	22.66	128.00	14.50	128.00
	TA-GRPO-*d*	Gated	33.60	**70.95**	**29.69**	**59.41**	**14.94**	**67.67**
Dream-7B	Pre-Trained	Top-2	39.60	128.00	22.27	128.00	6.15	128.00
	Pre-Trained	Gated	**40.00**	90.59	22.27	156.38	8.35	97.11
	diffu-GRPO	Top-2	34.00	128.00	18.75	128.00	9.52	128.00
	TA-GRPO-*d*	Gated	39.60	**90.14**	**36.72**	**104.59**	**21.00**	**61.49**

Table 1: Results on **Math & Puzzle** tasks. Score (higher is better) and Avg.DS (average denoising steps; lower is better). Within each column, the **bold** numbers denote the best and the underlined numbers denote the second-best.

Method	Denoising Strategy	HumanEval		MBPP	
		Score	Avg.DS	Score	Avg.DS
Pre-Trained	Top-2	30.49	128.0	39.69	128.0
Pre-Trained	Gated	36.59	75.25	43.19	69.18
diffu-GRPO	Top-2	28.05	128.0	38.13	128.0
TA-GRPO-*d*	Gated	**37.80**	**68.87**	**45.14**	**62.72**

Table 2: Results on **Code** tasks in **LLaDA-8B**. Score (higher is better) and Avg.DS (lower is better).

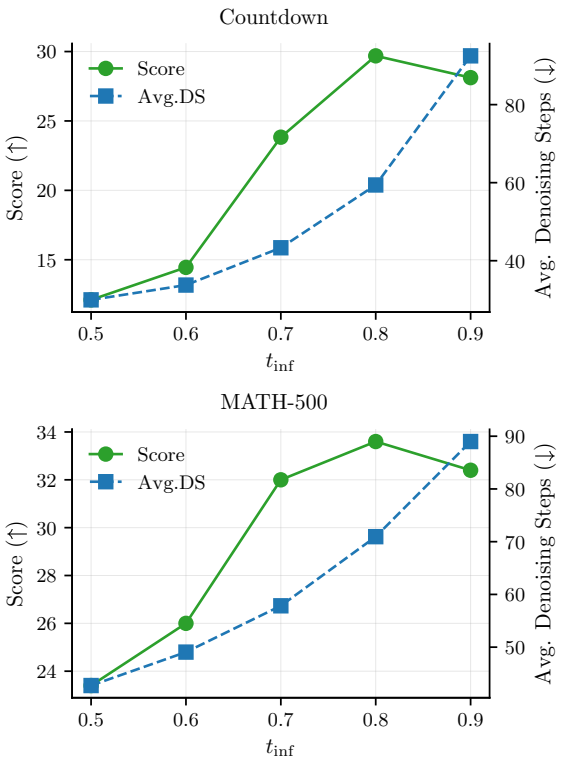

Figure 1: Performance curves of t_{inf}.

step count grows rapidly. The same qualitative pattern holds across both Countdown and MATH-500. We also observe that overly low training thresholds degrade performance, indicating that the choice of t_{tra} should avoid overly permissive gating.

This behavior arises because higher t_{inf} makes gating more conservative, keeping lower-confidence positions masked longer; consequently, the per-step unmasking budget shrinks and the overall number of steps increases.

One possible explanation is that, up to a certain point, a higher t_{inf} improves accuracy by delaying low-confidence decisions and allowing the model to commit earlier only to sufficiently confident tokens. However, when t_{inf} becomes overly conservative, even positions with relatively high confidence may remain masked for additional steps and be repeatedly resampled. This may introduce unnecessary denoising steps and, in some cases, lead to revisions of tokens that were already reasonable choices. As a result, the accuracy gain can diminish beyond a knee point, while the number of denoising steps continues to increase.

5.2 Ablation Study

Table 3 examines ablating components of $A' = (1+\Delta\zeta w)A$. Removing the unmasking-time weight $(-w)$ lowers accuracy and increases average steps, indicating that w is needed to learn an effective unmasking order. Removing both signals $(-w-\zeta)$ attains the highest scores but greatly lengthens trajectories, showing that the trajectory-level signal ζ prevents accuracy gains from coming at the cost of excessive steps. Using both signals yields the best accuracy–efficiency trade-off, suggesting that learning trajectories requires ex-post preference over completed paths (ζ) together with procedural guidance to reproduce efficient paths (w).

Variant	MATH-500		Sudoku	
	Score	Avg.DS	Score	Avg.DS
TA-GRPO-*d*	33.6	**70.95**	14.94	**67.67**
$-w$	30.2	78.43	7.76	76.79
$-w-\zeta$	**34.0**	79.27	**15.33**	140.54

Table 3: Ablation on the components of the trajectory-aware advantage A'.

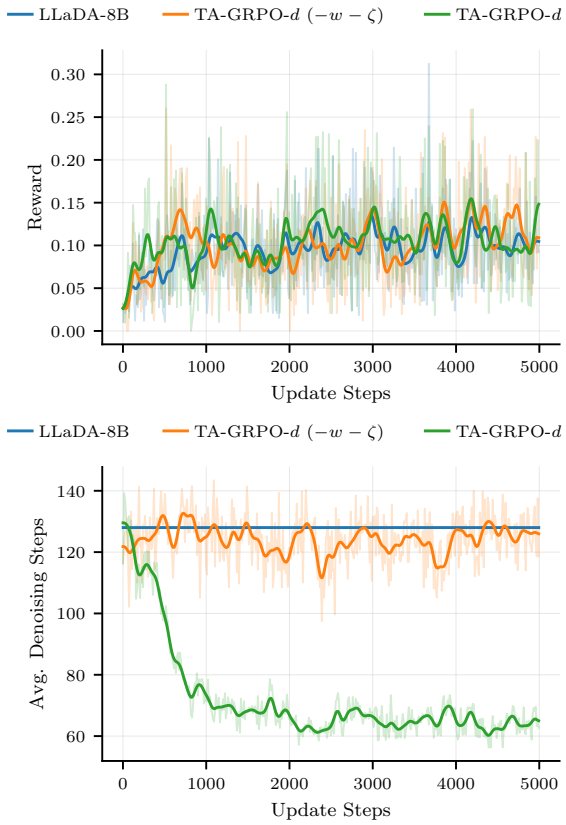

Figure 2: Training curves for the Sudoku experiment. Visualized with Gaussian smoothing (`sigma=3`).

5.3 Training Dynamics

Fig. 2 shows the training curves for Sudoku. The upper reward curve converges with a trend similar to that of *diffu*-GRPO(LLaDA-8B). In contrast, the lower curve for denoising steps reveals a clear difference: the proposed TA-GRPO-*d* rapidly reduces the average number of steps during the first few hundred updates and stabilizes around 60. By comparison, the baseline with the fixed schedule plus gating ($-w-\zeta$) stays around 120–140 throughout training, showing no reduction in steps.

These results suggest that although the overall reward improvement pattern is comparable to prior methods, the proposed approach quickly induces shorter denoising trajectories. In other words, even without an explicit penalty on steps, the advantage shaped by the intermediate-reward curve and unmasking-time information steers the policy toward shorter, more efficient trajectories. When only gating is applied, this pressure is absent and the step count does not decrease. Similar trends appear in other datasets; additional training-dynamics plots are provided in Appendix G. We also evaluate performance at 1000 steps to examine the impact of the early, sharp drop in average steps; results are reported in Appendix H.

Method	Train		Inference	
	Time	Avg.DS	Time	Avg.DS
diffu-GRPO	1.00×	128.0	1.00×	128.0
TA-GRPO-*d*	0.73×	72.24	0.68×	59.41

Table 4: Time & compute efficiency on LLaDA for Countdown. Train Avg.DS is computed as the average over all candidates generated throughout training.

5.4 Time & Compute Efficiency

Reducing the average number of denoising steps lowers runtime in both training and inference. We measured wall-clock time over all generated sequences in each phase. Note that RL training also includes backpropagation and gradient updates. If we hypothetically exclude those components, the number of forward passes required by GRPO during training is approximately proportional to $\big($average denoising steps $+$ `num iteration`$\big)\times$ num candidate n. In our experiments, the proposed method reduced the average denoising steps during training to 72.24, yielding a 27% reduction in training time. During inference, the average denoising steps decreased to 59.41, resulting in a 32% reduction in inference time. These results indicate that optimizing to reduce the number of denoising steps leads directly to real savings in computational cost and wall-clock time.

5.5 Case Studies: Visualizing Denoising Trajectories

We visualize the unmasking order to examine how the model trained with our method generates sequences quickly. Fig. 3 shows the generation order on a subset of the dataset under TA-GRPO-*d* and LLaDA-8B (Gated). We observe that the proposed method frequently decodes contiguous token spans in one shot, rather than unmasking scattered positions across the sequence. In contrast, for the baseline LLaDA-8B (Gated), cases that denoising a long clause in a single step are rare despite the

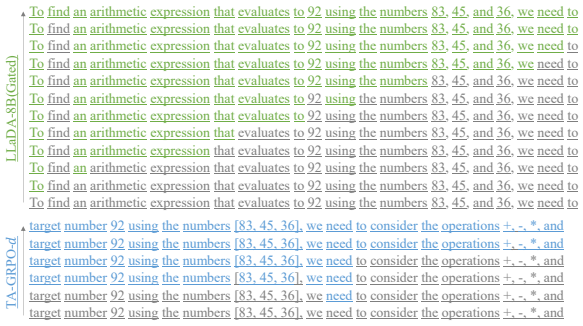

Figure 3: Visualization of denoising trajectories for Numbers: [83, 45, 36], Target: 92 (Countdown data), over steps 0–11 and 0–5, respectively. A colored token indicates the step at which the position was unmasked.

Gated. Additional cases beyond this example are provided in Appendix I.

6 Conclusion

We propose replacing fixed-schedule denoising in dLLMs with a stochastic confidence-gated denoising for trajectory exploration, and to train it using trajectory-aware GRPO that combines the intermediate-reward curve with unmasking-time information. The proposed method substantially reduces the average number of denoising steps while maintaining or improving accuracy on Math, Puzzle, and Code tasks. In the training-curve analysis, we observed that while rewards increase similarly to prior methods, the step count rapidly decreases from early updates, yielding efficient convergence. The ablation study shows that both the intermediate-reward signal and the timing weight contribute to balancing speed and performance. This approach is useful for simultaneously reducing the latency and computational cost of dLLM inference. Future directions include task-specific automatic threshold adaptation, multi-objective reward design, and scaling to large code and natural language generation.

Limitations

Because the proposed method aggregates rewards over intermediate states, training time can increase substantially as the cost of computing the reward function grows. Recent work in natural-language RL and GRPO-based studies reports that lightweight rewards—such as rule-based scoring, format verification, and simple correctness checks—are sufficient to improve reasoning ability (Wen et al., 2025); accordingly, most GRPO-style studies adopt lightweight reward functions (DeepSeek-AI, 2025; Mu et al., 2024). However, if a large verifier or composite grader is used (e.g., a large reward model or human-in-the-loop RLHF (Ouyang et al., 2022)), one must evaluate the reward at every intermediate denoising state for each input, so the actual cost accumulates linearly with the number of candidates and steps. Under such settings, computing trajectory-wide rewards may be impractical. In this paper, because we operate within the GRPO-based setting, we use only lightweight rewards; consequently, as reported in Table 4, the overall training time decreases when use lightweight rewards—the savings from reducing the average number of denoising steps (Avg.DS) outweigh the time spent aggregating rewards over intermediate states.

Acknowledgments

This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Government of Korea (MSIT) (No. 2022R1A2C1005316).

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2023. Structured denoising diffusion models in discrete state-spaces. *Preprint*, arXiv:2107.03006.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code.

DeepMind. 2024. Gemini diffusion. https://deepmind.google/technologies/gemini. Accessed: 2025-10-06.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Justin Deschenaux and Caglar Gulcehre. 2025. Beyond autoregression: Fast LLMs via self-distillation through time. In *The Thirteenth International Conference on Learning Representations*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025. Scaling diffusion language models via adaptation from autoregressive models. In *The Thirteenth International Conference on Learning Representations*.

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

Tianyi Li, Mingda Chen, Bowei Guo, and Zhiqiang Shen. 2025. A survey on diffusion language models. *arXiv preprint arXiv:2508.10875*.

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

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Xiaoran Liu, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. 2025a. Longllada: Unlocking long context capabilities in diffusion llms. *Preprint*, arXiv:2506.14429.

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

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian D Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. 2024. Rule based rewards for language model safety. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. Tinyzero. https://github.com/Jiayi-Pan/TinyZero. Accessed: 2025-01-24.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Preprint*, arXiv:1707.06347.

Saksham Sahai Srivastava and Vaneet Aggarwal. 2025. A technical survey of reinforcement learning techniques for large language models. *Preprint*, arXiv:2507.04136.

Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, and 1 others. 2022. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

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

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. 2025. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *Preprint*, arXiv:2506.14245.

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

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

Qihao Zhu Runxin Xu Junxiao Song Mingchuan Zhang Y.K. Li Y. Wu Daya Guo Zhihong Shao, Peiyi Wang. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models.

Symbol	Meaning
i	token position (word index)
t	denoising step index
j	candidate index in a group
n	number of candidates per input
L	sequence length
$\mathcal{V}$	vocabulary set
[MASK]	special mask token
$\mathcal{M}(s_t)$	masked index set at step t
$\mathcal{A}_t$	indices unmasked at step t
$p_{\theta,i}(v \mid s_t)$	per-position unmasking distribution
$c_i(s_t)$	confidence at position i (top-1 prob.)
t_{tra}, t_{inf}	train / inference thresholds
γ	sharpness (explore–exploit control)
ξ	Gaussian noise on confidence
$m_i(s_t)$	gating probability at (t, i)
$u_{t,i}$	Bernoulli gate at (t, i)
$\tau(s_0)$	stopping time (all masks removed)
k_j	stopping time of candidate j
$S^{(j)}$	final sequence of candidate j
$r_t^{(j)}$	intermediate reward at step t
$K_{\max}$	maximum k_j in the candidate group
Δ	trajectory-aware shaping coefficient
$d^{(j)}$	area of intermediate reward curve (AUC)
$\zeta^{(j)}$	trajectory-level z-score
$w_i^{(j)}$	unmasking-time weight of token i
$R^{(j)}$	final reward of candidate j
$A^{(j)}$	group-centered advantage
$A_i^{\prime(j)}$	token-weighted advantage
$\rho_i^{(j)}(\theta)$	PPO importance ratio
ε	PPO clip width
β	KL regularization weight
p_{ref}	reference policy for KL
K_θ^u	one-step conditional transition kernel
$\lvert\cdot\rvert$	size of set

Table 5: Notation used throughout the paper.

A Notations

For the reader's convenience, Table 5 summarizes the definitions and descriptions of the symbols used in the equations throughout this paper.

B Formal Definition of $o_i^{(j)}$

In this paper, $o^{(j)} = \{o_1^{(j)}, \ldots, o_{|S^{(j)}|}^{(j)}\}$ is defined as the set of unmasking times for candidate j. Specifically, define the initial set of masked indices as

$$I^{(j)} \;=\; \{\, i \in \{1, \ldots, L\} \;:\; s_{0,i}^{(j)} = [\text{MASK}] \,\} \tag{22}$$

and fix a bijection $\iota^{(j)} : \{1, \ldots, |I^{(j)}|\} \to I^{(j)}$ that enumerates the elements of $I^{(j)}$ *from left to right (ascending order)*. For each $i \in I^{(j)}$ and each

step $t \geq 1$, define the unmasking indicator

$$r_{i,t}^{(j)} = \mathbf{1}\{ s_{t-1,i}^{(j)} = [\text{MASK}] \wedge s_{t,i}^{(j)} \neq [\text{MASK}] \} \tag{23}$$

then the unmasking time of token i is

$$o_i^{(j)} = \min\{ t \geq 1 : r_{i,t}^{(j)} = 1 \} \tag{24}$$

Because a fallback is used to prevent stall steps, $o_i^{(j)}$ is finite.

C Reward Functions

We design reward functions by considering the formats of the datasets and the generative capabilities of the pre-trained models.

MATH-500 (LLaDA-8B)

- **Format and Correct Reward**:

  +0.1 if an expression of the form `\boxed{...}` can be detected;

  +1.0 if the answer extracted from `\boxed{...}` is correct.

MATH-500 (Dream-7B)

- **Format Reward**:

  +0.5 if the output follows `<reasoning> ... <\reasoning> <answer> ... <\answer>`;

  +0.5 if it prints **\boxed{...}**.

- **Correct Reward**:

  +2.0 if the answer extracted via `\boxed{...}` is correct.

Countdown

- **Format and Success Reward**:

  +0.1 if the answer can be detected in `<reasoning> ... <\reasoning> <answer> \boxed{...} <\answer>`;

  +1.0 if evaluating the detected expression yields the exact target value.

Sudoku

- **Blank-level Correct Reward**: Using `<reasoning> ... <\reasoning> <answer> ... <\answer>`, compare the detected answer with the Sudoku solution and use, as the reward, the proportion of correctly filled blanks in the 4×4 puzzle.

Kodcode

- **Format and Success Reward**:

  +0.1 if the answer can be detected in `<reasoning> ... <\reasoning> <answer>` “‘`python ... “‘ <\answer>`;

  +1.0 if the detected program, when executed by a Python interpreter, passes all dataset-provided test cases.

D Detail of Datasets

We use MATH-500, Countdown, Sudoku, Kodcode, HumanEval, and MBPP for training and evaluation. All datasets are publicly available for research.

MATH-500. A curated evaluation subset drawn from MATH (Hendrycks et al., 2021), which consists of high-school competition mathematics problems. It includes descriptive problems across algebra, number theory, geometry, and combinatorics, and requires final numeric or short symbolic answers. Grading is based on exact-match after simple normalization. Official train/test splits are provided with 7,500 and 500 problems, respectively.

Countdown. An arithmetic puzzle in which, given a set of integers and a target value, one must form an expression using only the four basic operations and parentheses that exactly equals the target. For example, with Numbers: [49, 55, 53], Target: 51, using each number exactly once, an expression like 55 - 53 + 49 is counted as correct. Invalid expressions are marked incorrect; both training and evaluation use exact equality to the target for scoring. Because multiple valid solutions can exist even with small sets, we first perform format validation and safe evaluation before checking the value. The dataset provides 490k (numbers, target) pairs; following Zhao et al. (2025), we use 256 pairs for evaluation.

Sudoku. A 4×4 Sudoku dataset with 2×2 subgrids. The input is a length-16 string where 0 denotes a blank (e.g., 4320004330100004). The output is a completed length-16 string in which every row, column, and each 2×2 subgrid contains the digits 1–4 exactly once (e.g., 4321124334122134). Grading checks both constraint satisfaction and consistency with the input. Via Github[2], the dataset provides 1M (puzzle, solution) pairs; following Zhao et al. (2025), we use 256 for evaluation.

[2]https://github.com/Black-Phoenix/4x4-Sudoku-Dataset

Hyperparameter	Value
Training threshold t_{tra}	0.7
Inference threshold t_{inf}	0.8
Sharpness γ	0.5
TA shaping Δ (MATH-500, Code)	0.5
TA shaping Δ (Sudoku, Countdown)	1.0
Diffusion steps(Top-2)(256/128 = 2)	128
Block length (Semi-Autoregressive)	32
KL β	0.04
PPO clip ε	0.5
Num cand. n (LLaDA / Dream)	6 / 4
Optimizer	AdamW
Learning rate	3×10^{-6}
Adam betas	$(0.9,\ 0.99)$
Weight decay	0.1
Max grad norm	0.2
Gradient accumulation	2
Max prompt length	200
Max completion length(Mcl)	256
num iterations μ	12
num iterations masking prob.	0.15
Quantization	4-bit load
LoRA $(r,\ \alpha,$ dropout)	(128, 64, 0.05)
Seed	42

Table 6: Detailed Experiment Hyperparameters.

Kodcode. A code-generation dataset where the model writes Python function bodies from natural-language descriptions and examples. Each item includes a problem statement, a function signature, and hidden unit tests. We conduct RL on this dataset and evaluate generalization on other coding benchmarks. We use the 9,000 preprocessed training problems provided by the official repository[3] and use them only for training.

HumanEval. A collection of Python programming tasks with function signatures and short descriptions; evaluation runs public unit tests to determine correctness. Evaluation reports Pass@1 using the public tests; We evaluate on all 164 tasks.

MBPP. A collection of short Python function-implementation problems with problem statements, I/O examples, and reference tests. Evaluation reports Pass@1 using the public tests; We use simple format checks and test pass rate as rewards. We evaluate on all 257 tasks.

E Detail of Experiment Setup

We describe the hyperparameters used for training and evaluation in the main experiments. We use open-source discrete diffusion LLMs: LLaDA-8B-Instruct as LLaDA-8B and Dream-v0-Instruct-7B as Dream-7B. The key hyperparameters of the proposed method are $t_{\text{tra}} = 0.7$ and $t_{\text{inf}} = 0.8$, with $\gamma = 0.5$. During reinforcement learning, we trained with AdamW at a learning rate of 3×10^{-6}. The maximum prompt length was 200 tokens, and the maximum completion length was fixed at 256 tokens. Accordingly, for Kodcode (whose prompts can be relatively long), we used only problems whose prompt length was ≤ 200 tokens. The batch size during training was 1, and the number of candidates n was 6 for LLaDA and 4 for Dream. We trained using QLoRA (Dettmers et al., 2023) on pre-trained dLLMs

We ran experiments on either RTX3090 $\times$ 2 or RTX4090 $\times$ 2. For fairness, all experiments for a given dataset in the main results were conducted on the same server. Specifically, MATH-500, Countdown, and HumanEval were run on RTX3090 $\times$ 2, while Sudoku, Kodcode, and MBPP were run on RTX4090 $\times$ 2. For LLaDA-8B, the approximate RL training wall-clock times per dataset were: Kodcode 35 h, Sudoku 15 h, Countdown 27 h, and MATH-500 31 h. Additional hyperparameters used in training and evaluation are listed in Table 6 and can also be found in our GitHub repository.

F Related Works

F.1 Diffusion Large Language Models

According to Li et al. (2025), Diffusion Large Language Models are broadly classified into continuous dLLMs and discrete dLLMs. The continuous family (Li et al., 2022; Strudel et al., 2022) performs diffusion–reverse diffusion in a continuous embedding space, while the discrete family (Gong et al., 2025; Liu et al., 2025a; Nie et al., 2025; Ye et al., 2025) operates in token space by iterating noise injection and selective unmasking. A typical inference pipeline for discrete dLLMs (i) starts from an initial state where the target span is fully masked, (ii) at each step unmasks only a subset of masked positions. Recently, inference acceleration has been standardized by combining semi-autoregressive decoding that preserves causal structure across blocks while performing parallel unmasking within blocks (Nie et al., 2025), caching mechanisms (dKV/dLLM-cache) (Liu et al., 2025b), and step distillation/guidance (Deschenaux and Gulcehre, 2025). Large-scale discrete dLLMs report AR-parity quality either via adaptation with AR initialization (Ye et al., 2025) or training from scratch (Nie et al., 2025), and there

[3] https://github.com/KodCode-AI/kodcode

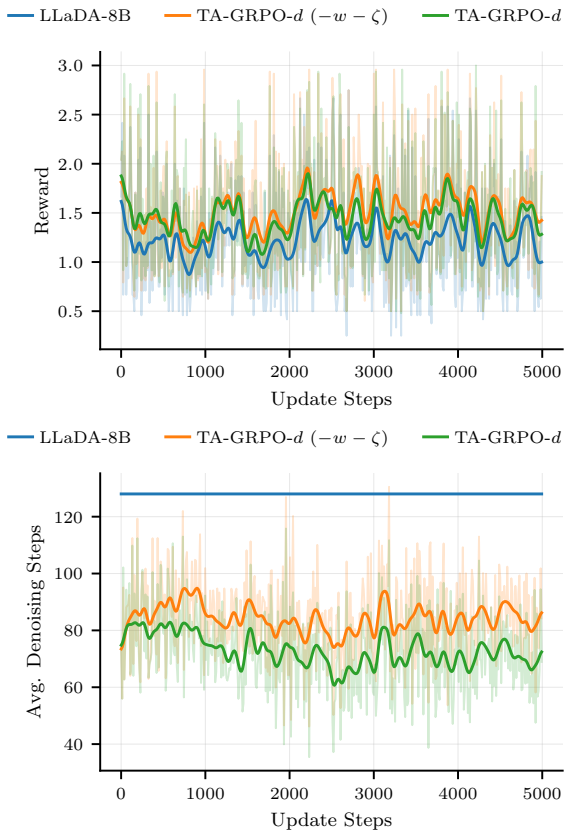

Figure 4: Training curves for the MATH-500 experiment. Visualized with Gaussian smoothing (`sigma=3`).

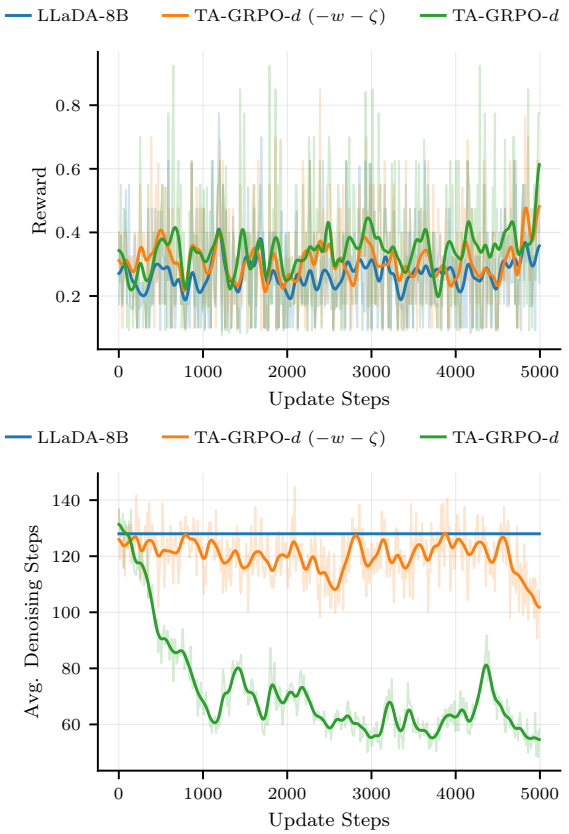

Figure 5: Training curves for the Countdown experiment. Visualized with Gaussian smoothing (`sigma=3`).

are also reports of throughput at the level of thousands of tokens per second (DeepMind, 2024).

F.2 Reinforcement Learning in LLMs

In LLMs, reinforcement learning (RL) (Srivastava and Aggarwal, 2025) initially established the RLHF pipeline (Ouyang et al., 2022) as the standard for improving alignment and instruction following. Subsequently, preference-optimization methods such as DPO (Rafailov et al., 2023) emerged to reduce human effort, though they face limitations in multi-objective optimization and exploration. Recent trends converge toward (i) GRPO-style updates (Zhihong Shao, 2024), which bundle multiple samples from the same prompt and update via relative advantage to gain stability and efficiency, and (ii) combining RL with verifiable rewards (RLVR) (Wen et al., 2025) and outcome-based rewards to boost reasoning in math and logic. In tandem, an early report of transplanting RL to discrete dLLMs, *diffu*-GRPO (Zhao et al., 2025), circumvents the non-sequential likelihood issue of dLLMs via mean-field and one-step per-token estimators, showing consistent improvements across math, puzzle, and code tasks.

Step	MATH-500		Countdown		Sudoku	
	Score	Avg.DS	Score	Avg.DS	Score	Avg.DS
0	33.2	84.00	21.48	128.84	7.81	135.10
1000	32.2	74.88	21.48	82.05	13.48	87.15
5000	33.6	70.95	29.69	59.41	14.94	67.67

Table 7: Experiment result of LLaDA-8B at step 0(pre-trained), 1000, 5000.

G Training Dynamics on MATH-500, Countdown

Fig. 4,5 depict the training-dynamics plots for the Math and Countdown datasets. The results exhibit trends similar to those reported in Sec. 5.3. For MATH-500, as shown in Table 1, applying the proposed denoising strategy yields a lower number of denoising steps from the outset; moreover, when the proposed method is used, the gap widens in later steps. This suggests that the method actively optimizes the denoising steps and achieves further reductions over training. In both experiments, we observe pronounced early decreases in the number of denoising steps followed by stabilization.

H Evaluation at 1,000 Step

Table 7 presents the evaluation at 1,000 steps. The results show that even with only 1,000 training steps, the average number of denoising steps can be substantially reduced: on MATH-500 from 84 to 74.88, on Countdown from 128.84 to 82.05, and on Sudoku from 135.1 to 87.15. Accuracy also changes slightly. When training is extended to 5000 steps, we observe additional reductions in the average number of denoising steps along with performance improvements. We interpret this as indicating that a large portion of the average-step reduction happens early, while further reductions continue steadily as training progresses.

I Visualizing Denoising Trajectories

In this paper, Fig. 7, 8 visualizes the denoising trajectories over the sequence. In our experiments, we generate a total of 256 tokens, so we visualize the full trajectories for all 256 tokens under LLaDA-8B (Gated) and TA-GRPO-*d*. For both methods, denoising generally proceeds from left to right in generated block. Under TA-GRPO-*d*, cases that decode long contiguous spans—such as clauses or phrases—in one shot occur frequently, whereas such cases are rare for LLaDA-8B (Gated). This confirms that, over the full dataset, we observe trends consistent with those described in the main text.

J Effect of Larger Fixed Top-*k* Schedules

To examine whether the efficiency gains of TA-GRPO-*d* can be explained by simply increasing the number of tokens restored per denoising step, we additionally evaluate larger fixed Top-*k* schedules on the pre-trained Dream-7B model. As shown in Table 8, larger fixed Top-*k* schedules shorten denoising trivially, but at the cost of severe performance degradation across all three Math & Puzzle tasks. This indicates that the gains of TA-GRPO-*d* cannot be explained by naive aggressive unmasking alone, and instead arise from learning better denoising trajectories.

Use of AI Assistant

We used ChatGPT(OpenAI) solely to translate the draft into English, and all translated content was thoroughly reviewed by the authors.

Model	Method	Denoising Strategy	MATH-500		Countdown		Sudoku	
			Score (↑)	Avg.DS (↓)	Score (↑)	Avg.DS (↓)	Score (↑)	Avg.DS (↓)
Dream-7B	Pre-Trained	Top-2	**39.60**	128.00	**22.27**	128.00	**6.15**	128.00
	Pre-Trained	Top-4	23.80	64.00	13.30	64.00	2.49	64.00
	Pre-Trained	Top-8	8.40	**32.00**	3.12	**32.00**	0.78	**32.00**

Table 8: Effect of larger fixed Top-k schedules on **Math & Puzzle** tasks in **Dream-7B**. Increasing Top-k reduces Avg.DS mechanically, but causes substantial degradation in task performance.

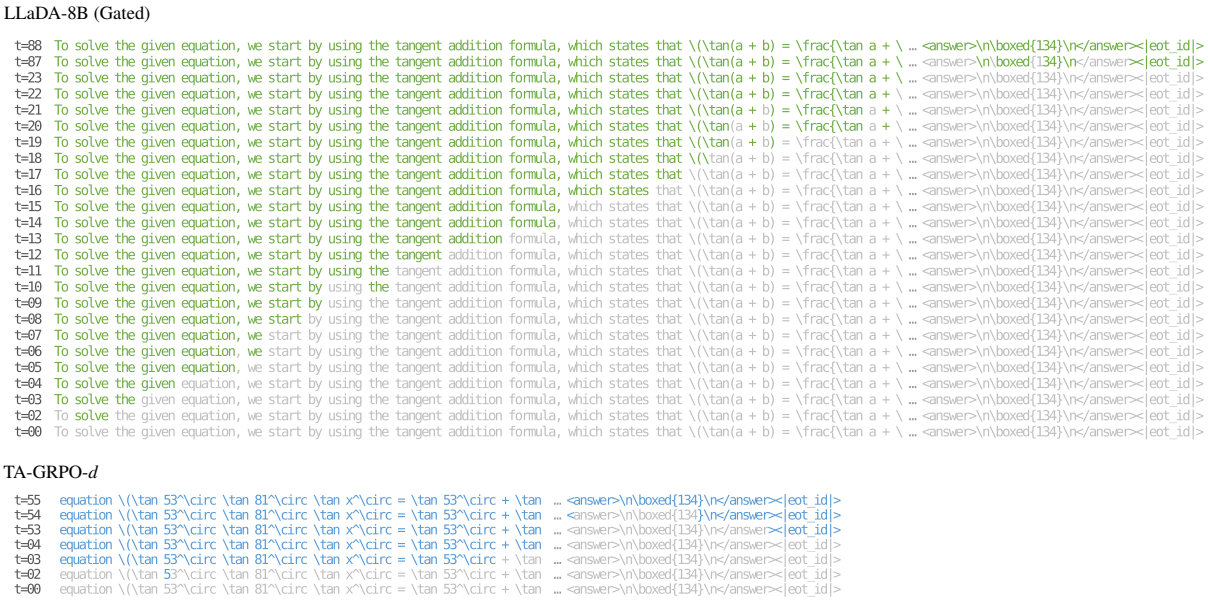

Figure 7: Visualization of denoising trajectories for 8th MATH-500 test data on LLaDA-8B(green) and TA-GRPO-d(blue). We visualize the first 32 tokens and the last 16 tokens (without padding). A colored token indicates the step at which the position was unmasked.

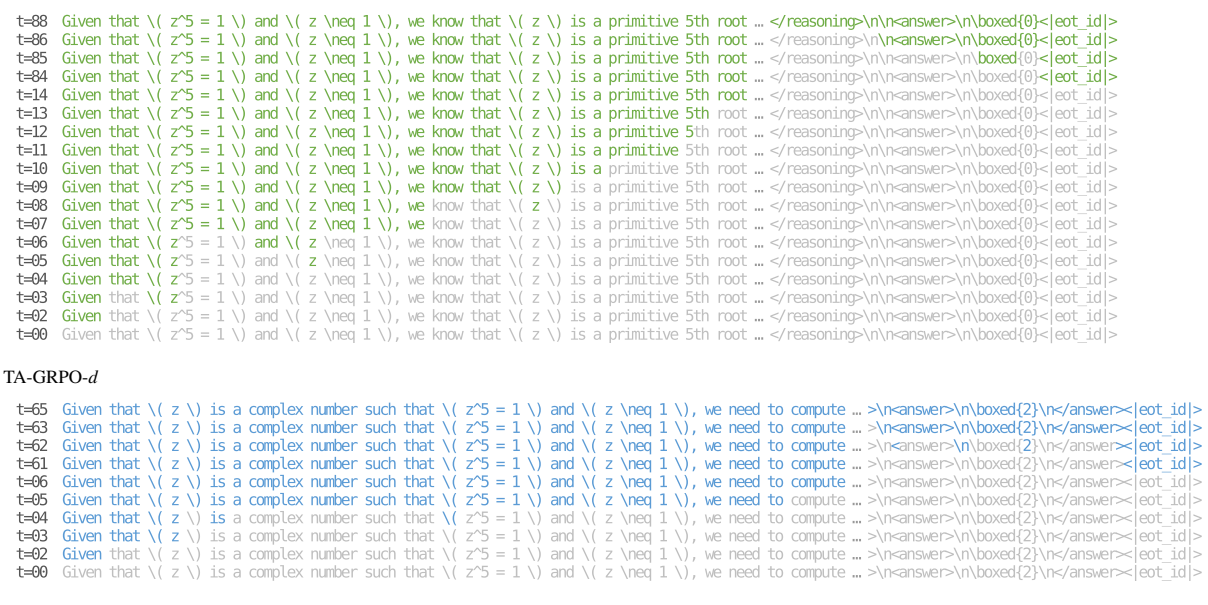

Figure 8: Visualization of denoising trajectories for 77th MATH-500 test data on LLaDA-8B(green) and TA-GRPO-d(blue). We visualize the first 32 tokens and the last 16 tokens (without padding). A colored token indicates the step at which the position was unmasked.