# *d*-TreeRPO: Towards More Reliable Policy Optimization for Diffusion Language Models

**Anonymous ACL submission**


## Abstract

Reinforcement learning (RL) is pivotal for enhancing the reasoning capabilities of diffusion large language models (dLLMs). However, existing dLLM policy optimization methods suffer from two critical reliability bottlenecks: (1) reward sparsity, arising from coarse or unverifiable signals that impede accurate advantage calculation; and (2) their probability estimates do not account for the gap to the unbiased expectation over all decoding orders, which are intractable to compute. To mitigate these issues, we propose *d*-TreeRPO, a reliable RL framework for dLLMs that leverages tree-structured rollouts and bottom-up advantage computation based on verifiable outcome rewards to provide fine-grained and verifiable step-wise reward signals. Furthermore, we provide a theoretical proof demonstrating that increasing prediction confidence effectively minimizes the gap between unbiased expected prediction probabilities and its single-step forward pass estimate. Guided by this analysis, we introduce a time-scheduled self-distillation loss during training that enhances prediction confidence in later training stages, thereby enabling more accurate probability estimation and better performance. Experiments demonstrate that *d*-TreeRPO outperforms existing baselines and achieves significant improvements across multiple reasoning benchmarks. Specifically, it achieves +86.2% on Sudoku, +51.6% on Countdown, +4.5% on GSM8K, and +5.3% on Math500 compared to the base model. Code and checkpoints are available in anonymous repositories.


## 1 Introduction

Diffusion large language models (dLLMs) (Nie et al., 2024; Ye et al., 2023; Zhang et al., 2025) provide a parallel decoding alternative to autoregressive (AR) models. Unlike AR's sequential generation, dLLMs start with masked inputs and iteratively reveal tokens through parallel denoising steps, enabling faster inference. Closed-source models (e.g., Gemini Diffusion, Seed Diffusion (Song et al., 2025)) achieve 1,400-2,150 tokens/s, while open-source models like LLaDA (Nie et al., 2025; Zhu et al., 2025a,b), and Dream (Ye et al., 2025; Xie et al., 2025b) match or exceed AR models with comparable size in performance.

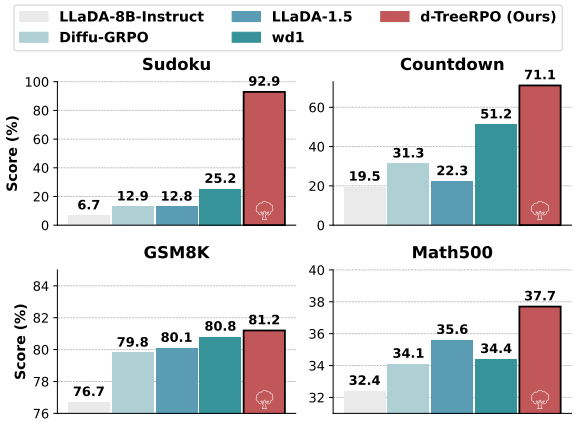



Figure 1: Performance comparison of *d*-TreeRPO with existing dLLM RL methods on four reasoning benchmarks, using LLaDA-8B-Instruct as base model.

Reinforcement learning (RL) plays a pivotal role in enhancing dLLM reasoning capabilities (Zhao et al., 2025a; Huang et al., 2025; Tang et al., 2025). Modern policy optimization frameworks, such as PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024), are fundamentally anchored by two core components: the advantage estimate $A(s, a)$ and the action log-probability $\log \pi_\theta(a|s)$. However, both of these components are prone to estimation errors, leading to suboptimal optimization outcomes. To ensure reliable RL for dLLMs, two requirements must be satisfied: *(1) granular and verifiable rewards that ensure accurate advantage calculation*, and *(2) precise estimation of action log-probabilities*.

However, existing RL approaches for dLLMs are hindered by two key reliability bottlenecks that prevent them from meeting the above requirements. **First, reward design lacks granularity**

**or verifiability**, often relying on sparse outcome rewards (Zhao et al., 2025a; Tang et al., 2025; Wang et al., 2025a; Zhu et al., 2025a) or unverified process signals (Huang et al., 2025; Yang et al., 2025a; Wang et al., 2025b), which can lead to issues like reward hacking. Second, probability estimation remains fundamentally challenging due to dLLMs' any-order decoding nature. Unlike AR models that decompose sequences via the chain rule for exact probability computation in a single forward pass (Shao et al., 2024; Zheng et al., 2025; Yu et al., 2025), dLLMs require approximations. **Existing log-probability estimation in dLLMs suffers from unanalyzed approximation errors.** ELBO-based methods (Ou et al., 2024; Nie et al., 2025; Yang et al., 2025c) provide only a biased lower bound and necessitate multi-time forward passes that are often too costly for practical use. While simplified single-step estimators (Zhao et al., 2025a; Tang et al., 2025) reduce computation, they leave the approximation bias unanalyzed.

To address these challenges, we propose $d$-TreeRPO, a reliable policy optimization framework specifically designed for dLLMs. As shown in Figure 2, we structure rollouts as a tree, with leaf nodes representing terminal states tied to verifiable rewards, and perform bottom-up reward and advantage calculations to obtain granular and verifiable process rewards. For estimating the prediction probability during parent-to-child generation, we theoretically prove that the gap between single-time forward pass probability estimates and the unbiased expectation decreases as the policy becomes more deterministic. However, overly deterministic policies can impair exploration in the early stages of training (Li et al., 2025; Cui et al., 2025; Tan et al., 2025), resulting in a trade-off between exploration and exploitation. To address this, we propose a time-scheduled self-distillation objective that is designed to be weak during the early stages to encourage exploration, and progressively strengthens to promote determinism, thereby improving estimation and overall model performance.

We evaluate $d$-TreeRPO on LLaDA-8B-Instruct (Nie et al., 2025) and LLaDA-MoE-7BA1B-Instruct (Zhu et al., 2025c) across Sudoku, Countdown, GSM8k (Cobbe et al., 2021), and Math500 (Lightman et al., 2023). Across two model backbones and four reasoning benchmarks, $d$-TreeRPO consistently outperforms baselines. Figure 1 summarizes the results on LLaDA-8B-Instruct, where $d$-TreeRPO achieves +86.2% (Sudoku), +51.6% (Countdown), +4.5% (GSM8k), and +5.3% (Math500) over the base model. Full experimental results are provided in Section 4. Moreover, ablation studies further confirm the benefit of the self-distillation loss. In summary, the key contributions are:

- We propose $d$-TreeRPO, a more reliable policy optimization algorithm for diffusion language models, which provides fine-grained and verifiable rewards and more reliable probability estimations.
- We theoretically show that the estimation error of prediction probabilities decreases as the model becomes more confident, and reveal the exploration-exploitation trade-off in dLLM RL. Based on this, we propose a time-scheduled self-distillation loss to relieve the trade-off and improve performance.
- Extensive experiments and ablations demonstrate significant gains on multiple reasoning benchmarks and verify the effectiveness and practicality of $d$-TreeRPO.

## 2 Preliminaries

**Diffusion Large Language Models.** Diffusion Large Language Models (dLLMs) generate text through an any-order denoising process, gradually reconstructing masked tokens into coherent text until generation is complete. During training, for a corpus sample $y$, a timestep $u \in [0, 1]$ is randomly sampled. Tokens in $y$ are then masked with probability $1 - \alpha_u$, where $\alpha_u$ is a noise schedule (e.g., linear schedule $\alpha_u = 1 - u$), resulting in full masking at $u = 1$ and no masking at $u = 0$. The corrupted version of the sample is given by:

$$\mathbf{y}_{\text{corrupt},u} = m_u \odot \mathbf{y} + (1 - m_u) \odot \texttt{[MASK]} \qquad (1)$$

where $m_u$ is a binary mask vector sampled such that each element $m_u^{(i)} \sim \text{Bernoulli}(\alpha_u)$, $\odot$ denotes element-wise multiplication. The model then predicts the original tokens at masked positions by leveraging bidirectional attention. During inference, we discretize denoising timestep into $N$ steps with $\Delta u = 1/N$ and $u_n = 1 - n\Delta u$ for $n = 0, \cdots, N$. We initialize the completion state at $u_0 = 1$ as $z_{u_0} = \{\texttt{[MASK]}\}^L$. For $n = 0, \cdots, N-1$, given the current state $z_{u_n}$, the model computes $P_{u_n} = f_\theta(q,\ z_{u_n})$, samples tokens to reveal (e.g., the top-$k$ most confident tokens in masked positions), and updates the state to

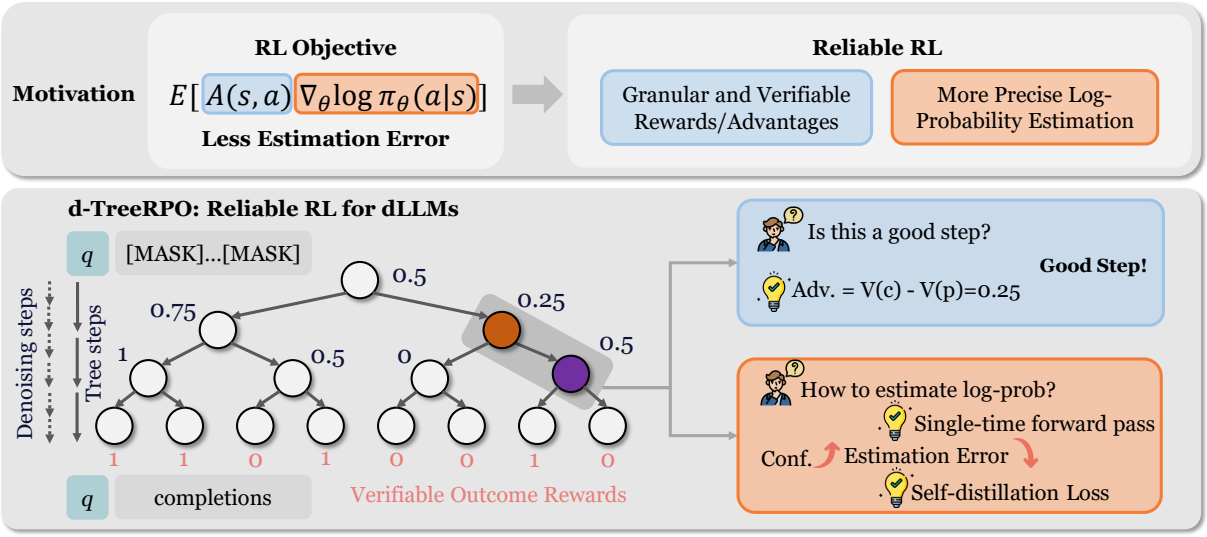



Figure 2: Overview of *d*-TreeRPO. Our framework employs a tree-structured rollout to propagate rewards and compute verifiable step-wise advantages. Guided by theoretical analysis, a time-scheduled self-distillation loss enhances model determinism in later training stages, improving estimation and delivering better performance.

$z_{u_{n+1}}$, where $u_{n+1} = u_n - \Delta u$. After $N$ steps, we obtain $z_{u_N}$ with $u_N = 0$, denotes as $z_0$.

**Group Relative Policy Optimization and its use in dLLMs.** GRPO (Shao et al., 2024) extends PPO (Schulman et al., 2017) with group-relative advantage estimation. The training objective is:

$$\mathcal{J}(\theta) = \mathbb{E}_{\substack{q\sim\mathcal{D},\\ o_i\sim\pi_{\theta_{old}}(\cdot|q)}} \left\{ \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{k=1}^{|o_i|}\left[\min\left(r_{i,k}(\theta)A_{i,k}, \right.\right.\right.$$
$$\left.\left.\left.\text{clip}\left(r_{i,k}(\theta), 1-\epsilon, 1+\epsilon\right)A_{i,k}\right) - \beta\mathcal{D}_{\text{KL}}\left(\pi_\theta \| \pi_{ref}\right)\right]\right\}, \quad (2)$$

where $\{o_i\}_{i=1}^{G}$ is a group of $G$ completions drawn from the old policy, group-relative advantage $A_{i,k} = R_{i,k} - \text{mean}(\{R_{i,k}\}_{i=1}^{G})$, and importance sampling ratio $r_{i,k}(\theta) = \frac{\pi_\theta(o_i^k)}{\pi_{\theta_{old}}(o_i^k)}$. Under this training objective and the specifics of dLLMs, two factors are crucial for reliable dLLM RL: *1) accurate advantage estimation, including verifiable and fine-grained discrimination*, and *2) precise probability estimation.* In AR models, the prediction probability is straightforward due to fixed decoding order. In dLLMs, however, the diversity of decoding orders necessitates taking an expectation over all denoising paths. Let $\sigma$ denote a permutation of the unmasking order, sampled from a distribution $\mathcal{Q}(\sigma)$, and let $u_{\sigma,k}$ be the timestep at which position $k$ is unmasked under $\sigma$. Accordingly, the per-token prediction probability is:

$$\pi_{\text{dLLM}}(o_i^k|q) = \mathbb{E}_{\sigma\sim\mathcal{Q}}\left[f_\theta^k\left(o_i^k \mid q, z_{u_{\sigma,k}}\right)\right], \quad (3)$$

where $f_\theta^k$ represents the distribution of model's forward function at position $k$, and $z_u$ denotes the completion state at timestep $u$.

**Existing dLLM RL Methods.** While existing works (Zhao et al., 2025a; Huang et al., 2025; Tang et al., 2025; Gong et al., 2025; Rojas et al., 2025; Zhao et al., 2025b) attempt to address the aforementioned two aspects, critical limitations remain. *1) Reward/advantage estimation:* Many methods directly distribute the final reward uniformly across all tokens. (Zhao et al., 2025a; Tang et al., 2025; Huang et al., 2025; Yang et al., 2025a; Xie et al., 2025a). Others introduce more fine-grained supervision via trained reward models (Wang et al., 2025b), but may sacrifice verifiability by introducing reward hacking risks. Consequently, existing designs often **cannot simultaneously provide *fine-grained* and *verifiable* signals**, leading to unreliable advantage estimates.

*2) Prediction probability estimation:* ELBO-based (Ou et al., 2024) methods use an evidence lower bound as the per-token log-likelihood approximation $\log \hat{\pi_\theta}(o_i^k|q)$ (we use $\hat{\cdot}$ to denote estimated quantities):

$$\mathbb{E}_{u,z_u}\left[\frac{1}{u}\mathbf{1}(z_u^k = \texttt{[MASK]})\log f_\theta^k(o_i^k|z_u, q)\right]. \quad (4)$$

Accordingly, the sequence-level ELBO (Rojas et al., 2025; Ou et al., 2025) is obtained by averaging the token-level estimates. However, ELBO only yields a biased lower-bound and requires computationally expensive multi-round Monte Carlo masking/forward passes for accuracy. For efficiency, methods such as VRPO (LLaDA-1.5) (Zhu et al., 2025a) and BGPO (Lin et al., 2025) reduce computation by decreasing the number of Monte Carlo samples or by giving another lower bound of ELBO

as an alternative objective. On the other hand, methods such as Diffu-GRPO (Zhao et al., 2025a) and wd1 (Tang et al., 2025) do per-token approximation using a single forward pass conditioned on the prompt and fully masked completions, which is

$$\log \hat{\pi_\theta}(o_i^k|q) = \log f_\theta^k(o_i^k|q) \tag{5}$$

It greatly reduces computational cost while still achieving noticeable training performance. However, **none of these methods rigorously quantify the bias between their approximations and the unbiased expected probability** in Equation (3), thereby undermining the reliability of policy-gradient updates.

# 3 *d*-TreeRPO

## 3.1 Tree-Structured RL: Constructing Granular and Verifiable Rewards

**Tree-Structured Rollout.** To reduce computation during rollout, we introduce a merging parameter $s$, grouping the $N$ denoising steps into $H = N/s$ tree steps. At each tree node (except leaves), $B$ independent samplings are performed, each involving $s$ consecutive diffusion steps. This follows the dLLM's inherent sampling mechanism, such as confidence-based sampling in LLaDA (Nie et al., 2025). The resulting tree structure produces $B^H$ leaf nodes for the same prompt, corresponding to $B^H$ complete generations.

**Rewards and Advantages.** The state value for each leaf node is determined by the verifiable outcome rewards. Based on this, the state value of each tree node is computed bottom-up as the average of its child nodes, i.e.,

$$V_p = \frac{1}{|C_p|} \sum_{c \in C_p} V_c, \tag{6}$$

where $C_p$ is the set of child nodes of node $p$ and $V_c$ is the value of child node $c$. The transition advantage from a parent node to a child node is defined as:

$$A_p^c = V_c - V_p = V_c - \frac{1}{|C_p|} \sum_{c' \in C_p} V_{c'}, \tag{7}$$

where the child nodes under the same parent node are treated as a group, and the relative advantage is calculated within this group.

**Optimization Overview.** During loss computation, we treat each depth-1 subtree (i.e., a parent node $p$ and its children) as a training unit. Specifically, we perform group-wise policy updates over each parent's child set $C_p$ using a GRPO-style objective. Although the update is group-relative at the branch level, the loss is applied token-wise: the parent-to-child transition advantage $A_p^c$ is assigned to every newly generated token in the child completion $c$. This also requires estimating the probabilities of the newly generated tokens conditioned on the parent state, which we detail in Section 3.2.

## 3.2 Estimation of Conditional Log-Probabilities in Tree Structures

**Single-time Forward Pass Estimation.** For computational efficiency, following Diffu-GRPO (Zhao et al., 2025a) and wd1 (Tang et al., 2025), we utilize single-time forward pass to estimate the log-probabilities of the newly generated tokens in child completions given the parent state. Suppose the positions newly decoded in the child node are $d_1, d_2, \ldots, d_k$, with the corresponding tokens $o^{d_1}, o^{d_2}, \ldots, o^{d_k}$. Then the estimation is:

$$\log \hat{\pi_\theta}(o^{d_i} \mid p) = \log f_\theta^{d_i}(o^{d_i} \mid p), \tag{8}$$

where $\log \hat{\pi_\theta}(o^{d_i}|p)$ denotes the estimation of $\log \pi_\theta(o^{d_i}|p)$, and $f_\theta^{d_i}$ represents the distribution of model's forward function at position $d_i$. $p$ corresponds to the parent node's state, which includes the prompt and the partially masked completion.

**Comparability Issue and Block-wise Adaptation.** The design above introduces a comparability issue: when child nodes of the same parent decode different positions within a tree step, the log-probabilities estimated by Equation (8) become incomparable. To address this, we adopt the block-wise decoding approach often used in dLLM inference, where tokens within each block are decoded in any order, and blocks are decoded sequentially. By aligning tree-search hyperparameters with the block-wise structure—ensuring the total tree steps $H$ divide the number of blocks $L/b$—each tree step decodes full blocks, restoring log-probability comparability among sibling nodes.

## 3.3 Relationship Between Prediction Confidence and Estimation Error

To understand the approximation error of prediction probability for dLLMs, we formally compare the single-time forward pass estimator with the unbiased expected probability in Equation (3) under all possible decoding orders.

**Theorem 1** (High-probability Estimation Error Bound). Let $p$ be the parent state. In one tree step, we decode $k$ masked positions with index set $\mathcal{D} = \{d_1, \ldots, d_k\}$, yielding realized tokens $o^{d_1}, \ldots, o^{d_k}$. A revelation order is a permutation $\sigma = (\sigma_1, \ldots, \sigma_k) \sim \mathcal{Q}$, where $\sigma_j \in \mathcal{D}$ is the posi-

tion revealed at sub-step $j$. Define:

$$q_j(\sigma) = f_\theta^{\sigma_j}\Big(o^{\sigma_j} \mid p, \{o^{\sigma_1}, \ldots, o^{\sigma_{j-1}}\}, \texttt{[MASK]}_{\text{else}}\Big). \quad (9)$$

The single-time forward pass estimator is:

$$\hat{p}_d := f_\theta^d(o^d \mid p), \quad (10)$$

And let $\tau(d, \sigma)$ be the unique index such that $\sigma_{\tau(d,\sigma)} = d$. For $\delta \in (0, 1)$, let $q_{d,1-\delta}$ be the $(1-\delta)$-quantile of $q_{\tau(d,\sigma)}(\sigma)$ under $\sigma \sim \mathcal{Q}$, i.e.:

$$\Pr_{\sigma\sim\mathcal{Q}}\big(q_{\tau(d,\sigma)}(\sigma) \geq q_{d,1-\delta}\big) \geq 1 - \delta. \quad (11)$$

And define

$$\epsilon_{d,\delta} := \max\{1 - \hat{p}_d,\ 1 - q_{d,1-\delta}\}. \quad (12)$$

Then for $d \in \mathcal{D}$ with $\epsilon_{d,\delta} \in [0, 1)$, we have:

$$\boxed{\Pr_{\sigma\sim\mathcal{Q}}\left(\left|\log \frac{q_{\tau(d,\sigma)}(\sigma)}{\hat{p}_d}\right| \leq -\log(1 - \epsilon_{d,\delta})\right) \geq 1 - \delta.} \quad (13)$$

**What does this imply?** With probability at least $1 - \delta$ over decoding orders $\sigma \sim \mathcal{Q}$, the log-ratio error between the path-wise probability $q_{\tau(d_i,\sigma)}(\sigma)$ and the single-pass estimate $\hat{p}_{d_i}$ is controlled within $-\log(1 - \epsilon_{d_i,\delta})$. When the model is more **confident** (i.e., smaller $\epsilon_{d_i,\delta}$), the right-hand side becomes smaller and **the bound is tighter**. Since $p_{d_i}^{\text{true}} = \mathbb{E}_{\sigma\sim\mathcal{Q}}[q_{\tau(d_i,\sigma)}(\sigma)]$ is **an expectation over decoding orders**, this result explains why **increasing confidence makes single-pass probability estimation more accurate** in practice. A detailed proof of Formula (13) is provided in Appendix A.

### 3.4 Design of Self-distillation Loss

**Exploration-Exploitation Trade-off.** Higher prediction confidence improves probability estimation accuracy (Section 3.3) but limits exploration (Cui et al., 2025; Zhang et al., 2024), creating a trade-off in dLLM RL. Early training benefits from greater exploration, while later stages require precise probability estimation for stable convergence and better performance. To balance this, we propose a **time-scheduled self-distillation loss** that starts with minimal weighting to encourage exploration and gradually increases to focus on high-advantage actions in later training.

**Advantage-Weighted Target Distribution.** For a depth-1 subtree, we first select child nodes with positive advantages $C_p^+ = \{c \in C_p \mid A_p^c > 0\}$. The time-dependent temperature $\tau(t)$ controls distribution sharpness:

$$\tau(t) = \tau_{\text{max}} \cdot \left(1 - \frac{t}{T}\right)^\beta, \quad (\beta \in (0, 1]) \quad (14)$$

where $T$ denotes total training steps, and $t$ is the current training step. The advantage-weighted coefficients are computed through:

$$w_c = \frac{\exp\left(A_p^c/\tau(t)\right)}{\sum_{c' \in C_p^+} \exp\left(A_p^{c'}/\tau(t)\right)}, \quad \forall c \in C_p^+ \quad (15)$$

For each position-token pair $(\sigma_i, v)$ ($\sigma_1, \ldots, \sigma_k$ are positions decoded in current tree step, and $v \in \mathcal{V}$, $\mathcal{V}$ is the vocabulary set), we construct the target distribution $P_{\text{target}}^{\sigma_i}(v)$ through advantage-weighted aggregation:

$$P_{\text{target}}^{\sigma_i}(v) = \sum_{c \in C_p^+} w_c \cdot \mathbf{1}[v_c^{\sigma_i} = v], \quad \forall v \in \mathcal{V}. \quad (16)$$

**Time-scheduled Self-distillation Loss.** The final self-distillation loss $\mathcal{L}_{\text{distill}}$ is then defined as:

$$\lambda(t) \cdot \frac{1}{k} \sum_{i=1}^{k} D_{\text{KL}}\left(P_{\text{target}}^{\sigma_i}(\cdot) \parallel \pi_\theta^{\sigma_i}(\cdot|p)\right), \quad (17)$$

where $k$ is the number of newly generated tokens in this tree step, and $\lambda(t)$ is a time-scheduled weight:

$$\lambda(t) = \lambda_{\text{max}} \cdot \frac{e^{\gamma t/T} - 1}{e^\gamma - 1}. \quad (18)$$

As training progresses, $\tau(t)$ becomes increasingly sharp, favoring selections from child nodes with the highest advantages, while $\lambda(t)$ grows progressively larger, causing the self-distillation loss being increasingly important.

**Complete Loss Function.** Therefore, the overall loss of $d$-TreeRPO is given in Equation (19). Here, $\mathcal{T}(q)$ denotes the rollout tree for prompt $q$, and $k$ is the number of newly decoded tokens in a tree step.

## 4 Experiments

### 4.1 Experimental Setup

**Models and Datasets.** We employ LLaDA-8B-Instruct (Nie et al., 2025) and LLaDA-MoE (Zhu et al., 2025c) as the base model[1]. For training and evaluation, we utilize two puzzle-based reasoning tasks: Sudoku and Countdown, and two mathematical reasoning tasks: GSM8K (Cobbe et al., 2021) and Math500 (Lightman et al., 2023).

**Baselines.** Evaluated dLLM RL baselines include Diffu-GRPO (Zhao et al., 2025a), VRPO (LLaDA-1.5) (Zhu et al., 2025a), wd1 (Tang et al., 2025), SAPO (Xie et al., 2025a), GDPO (Rojas et al., 2025) and TraceRL (Wang et al., 2025b). Detailed descriptions of each algorithms are provided in Appendix B.

**Training Details.** Following previous works (Zhao et al., 2025a; Tang et al., 2025), we use LoRA training with rank $r = 128$ and scaling factor $\alpha = 64$. We adopt a learning rate of $3 \times 10^{-5}$, set $\tau_{\text{max}} = 2$

[1]We did not use Dream (Ye et al., 2025) because our experiments found its output format inconsistent, which caused instability across various dLLM RL algorithms.

$$\mathcal{L}_{d\text{-TreeRPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\, p\in\mathcal{T}(q)}\Bigg[ -\frac{1}{|C_p|}\sum_{c\in C_p}\frac{1}{k}\sum_{i=1}^{k}\Bigg(\underbrace{\min\Bigg(\frac{f_\theta^{d_i}(o^{d_i}|p)}{f_{\theta_{old}}^{d_i}(o^{d_i}|p)}A_p^c, \text{clip}\Big(\frac{f_\theta^{d_i}(o^{d_i}|p)}{f_{\theta_{old}}^{d_i}(o^{d_i}|p)}, 1-\epsilon, 1+\epsilon\Big)A_p^c\Bigg)}_{\text{policy-gradient loss}} \tag{19}$$
$$\underbrace{-\beta D_{\text{KL}}\left[\pi_\theta(\cdot \mid p)||\pi_{\text{ref}}(\cdot \mid p)\right]}_{\text{KL loss}}\Bigg) + \underbrace{\frac{\lambda(t)}{k}\sum_{i=1}^{k} D_{\text{KL}}\Big(P_{\text{target}}^{\sigma_i}(\cdot) \parallel \pi_\theta^{\sigma_i}(\cdot|p)\Big)}_{\text{self-distillation loss}}\Bigg].$$

| Methods / Datasets | Sudoku | | Countdown | | GSM8k | | Math500 | |
|---|---|---|---|---|---|---|---|---|
| | 256 | 512 | 256 | 512 | 256 | 512 | 256 | 512 |
| LLaDA-8B-Instruct | 6.7 (+0.0) | 5.5 (+0.0) | 19.5 (+0.0) | 16.0 (+0.0) | 76.7 (+0.0) | 78.2 (+0.0) | 32.4 (+0.0) | 36.2 (+0.0) |
| + *Diffu-GRPO* (Zhao et al., 2025a) | 12.9 | 11.2 | 31.3 | 37.1 | 79.8 | 81.9 | 34.1 | <u>39.0</u> |
| + *VRPO (LLaDA-1.5)* (Zhu et al., 2025a) | 12.8 | 9.6 | 22.3 | 18.0 | 80.1 | 81.5 | <u>35.6</u> | 34.8 |
| + *wd1* (Tang et al., 2025) | 25.2 | 24.2 | 51.2 | 46.1 | 80.8 | 82.3 | 34.4 | <u>39.0</u> |
| + *SAPO* (Xie et al., 2025a) | 20.3 | 16.1 | 52.0 | 56.3 | 80.6 | 82.1 | 33.8 | 38.4 |
| + *GDPO* (Rojas et al., 2025) | <u>25.7</u> | 24.8 | <u>64.1</u> | <u>60.2</u> | <u>81.1</u> | 82.0 | <u>37.0</u> | 38.5 |
| + *TraceRL* (Wang et al., 2025b) | 25.6 | <u>25.4</u> | 50.4 | 52.6 | 80.3 | <u>82.4</u> | 35.6 | **39.1** |
| + *d-TreeRPO (ours)* | **92.9** (+86.2) | **80.3** (+74.8) | **71.1** (+51.6) | **62.1** (+46.1) | **81.2** (+4.5) | **82.6** (+3.6) | **37.7** (+5.3) | 38.9 (+2.7) |
| LLaDA-MoE-7BA1B-Instruct | 12.7 (+0.0) | 8.1 (+0.0) | 42.6 (+0.0) | 41.4 (+0.0) | 71.8 (+0.0) | 71.0 (+0.0) | 30.1 (+0.0) | 42.2 (+0.0) |
| + *Diffu-GRPO* (Zhao et al., 2025a) | 17.3 | 15.4 | 50.1 | 48.2 | 74.2 | 73.7 | 38.1 | 43.4 |
| + *wd1* (Tang et al., 2025) | <u>40.6</u> | <u>35.4</u> | 56.6 | <u>58.7</u> | **76.2** | 76.4 | 39.8 | 43.4 |
| + *SAPO* (Xie et al., 2025a) | 21.2 | 18.7 | 54.2 | 54.8 | 75.2 | <u>76.8</u> | 38.6 | 40.4 |
| + *GDPO* (Rojas et al., 2025) | 31.2 | 30.6 | <u>58.1</u> | 53.4 | 74.8 | 75.2 | 38.4 | 41.2 |
| + *TraceRL* (Wang et al., 2025b) | 27.4 | 25.2 | 54.2 | 49.1 | 75.4 | 76.1 | <u>40.0</u> | <u>44.1</u> |
| + *d-TreeRPO (ours)* | **78.3** (+65.6) | **72.0** (+63.9) | **67.2** (+24.6) | **60.6** (+19.2) | <u>75.5</u> (+3.7) | **77.4** (+6.4) | **41.2** (+11.1) | **46.3** (+4.1) |

Table 1: Performance comparison of *d*-TreeRPO with existing dLLM RL methods on reasoning tasks under 256/512-token generation settings. **Bold** values indicate the best results, with the second-best <u>underlined</u>. Performance gains of *d*-TreeRPO relative to the base model are shown in parentheses.

and $\beta = 0.7$ in Eq. (14), and set $\lambda_{\max} = 0.003$ and $\gamma = 2$ in Eq. (18) (see Appendix E for hyper-parameter analysis). The maximum generation length is set to 256 tokens with block-wise decoding (block length= 32) over 128 denoising steps. We set temperature= 0.9 and use confidence-based decoding during rollout. To balance computational efficiency and performance, the tree-structured rollout employs depth $H = 2$ and branch factor $B = 4$ (see Section 4.3 for sensitivity analysis). More training details are provided in Appendix C.

**Evaluation Details.** We evaluate model performance under 256-token and 512-token outputs, with block-wise decoding (block length= 32) and temperature=0.0. Denoising steps are configured as half the generation length. All tasks adopt zero-shot evaluation with pass@1 scoring.

## 4.2 Main Results

**Overall Performance**. Table 1 presents a comparison between *d*-TreeRPO and existing dLLM RL methods, using LLaDA-8B-Instruct and LLaDA-MoE-7BA1B-Instruct as the base models. Our approach achieves state-of-the-art performance in most settings, showcasing substantial improvements over the base models: **+86.2%** on Sudoku, **+51.6%** on Countdown, **+4.5%** on GSM8K, and **+5.3%** on Math500 with 256-token generations on LLaDA-8B-Instruct, as well as **+65.6%** on Sudoku, **+24.6%** on Countdown, **+3.7%** on GSM8K, and **+11.1%** on Math500 with 256-token generations on LLaDA-MoE-7BA1B-Instruct. Appendix D.1 demonstrates the training reward curves. Furthermore, case studies are provided in Appendix F.

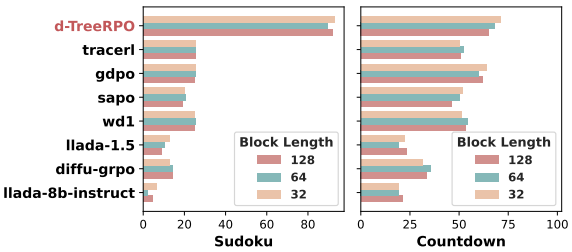
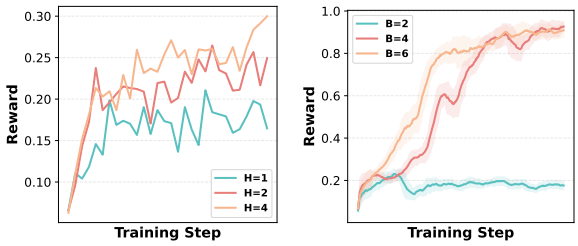



Figure 3: Performance comparison of *d*-TreeRPO with dLLM RL baselines under different decoding strategies.

**Robustness Against Decoding Strategies.** Figure 3 shows the performance of *d*-TreeRPO and dLLM RL baselines under different block-wise decoding strategies. Using LLaDA-8B-Instruct as the base model with 256-token generations, experiments on Sudoku and Countdown tasks were conducted with block lengths of 32, 64, and 128 for both training and inference. *d*-TreeRPO consistently outperforms baselines across all strategies, demonstrating strong robustness.




(a) $H$ (b) $B$

Figure 4: Training rewards on Sudoku task under different parameters $H$ and $B$ (LLaDA-8B-Instruct).

| Method | Sudoku | Countdown | GSM8k | Math500 |
|---|---|---|---|---|
| *d*-TreeRPO | **92.9** | **71.1** | **81.2** | **37.7** |
| *d*-TreeRPO w/o distill. | 89.8 | 66.4 | 80.9 | 36.1 |
| *d*-TreeRPO w. div. | 84.2 | 63.4 | 78.5 | 35.2 |

Table 2: Comparison of evaluation performance of $d$-TreeRPO, $d$-TreeRPO without self-distillation loss and $d$-TreeRPO with diversity-promoting loss, using LLaDA-8B-Instruct with 256-token generations.

### 4.3 Tree Height and Branch Factor Analysis

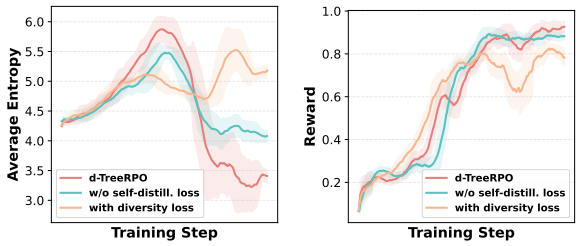
**Tree Height $H$ Analysis.** Tree height $H$ represents the steps required to traverse from the root to a leaf node. Increasing $H$ enables finer granularity in generation, improving reward precision. Training curves for $H = 1, 2, 4$ over the first 30 batches (Figure 4a) show that $H = 4$ improves the fastest, followed by $H = 2$, while $H = 1$ progresses the slowest, validating our analysis. However, the exponential computational cost of $d$-TreeRPO at higher $H$ prevented $H = 4$ from converging within reasonable time given our resource constraints (see Section 4.6 for details). Therefore, we adopted $H = 2$ as the primary setting for our experiments, balancing efficiency and performance.

**Branch Factor $B$ Analysis.** The branch factor $B$ represents the number of child nodes expanded from a parent during rollout. A larger $B$ improves exploration diversity and reduces parent node reward variance. Training curves for $B = 2$, 4, and 6 (Figure 4b) show that $B = 2$ achieves a final reward of only 0.2, while $B = 4$ and $B = 6$ converge above 0.9, with $B = 6$ showing faster initial improvement. However, due to polynomial growth in computational cost with $B$ (Section 4.6), we use $B = 4$ in our main experiments.

### 4.4 Effectiveness of Self-distillation Loss

**Setup**. To validate the effectiveness of the proposed self-distillation loss, we perform ablation studies. First, we compare the full $d$-TreeRPO model to a variant without the self-distillation loss. Additionally, to highlight the importance of progressively sharpening policy distribution, we introduce the **diversity-promoting loss** ($\mathcal{L}_{\text{div}}$). Designed as a symmetric counterpart to $\mathcal{L}_{\text{distill}}$, $\mathcal{L}_{\text{div}}$ intentionally increases policy diversity by reducing model confidence in later training stages. Inspired by prior work (Yang et al., 2025b), the loss increases entropy for negative samples. It operates on child nodes with negative advantages ($C_p^- = \{c \in C_p \mid A_p^c < 0\}$), assigning weights proportional to $|A_p^c|$. A negative target distribution, $P_{\text{div}}$, is constructed through weighted voting. Rather than minimizing KL divergence, the policy is trained to move away from this target via a negative KL objective, using the same time-scheduling coefficient $\lambda(t)$ as $\mathcal{L}_{\text{distill}}$. Therefore, $\mathcal{L}_{\text{diversity}}$ is:

$$-\lambda(t) \cdot \mathbb{E}_{q\sim\mathcal{D},\, p\in\mathcal{T}(q)} \left[ \frac{1}{k} \sum_{i=1}^{k} D_{\text{KL}} \left( P_{\text{div}}^{\sigma_i}(\cdot) \parallel \pi_\theta^{\sigma_i}(\cdot|p) \right) \right]. \quad (20)$$




(a) Entropy curves (b) Reward curves

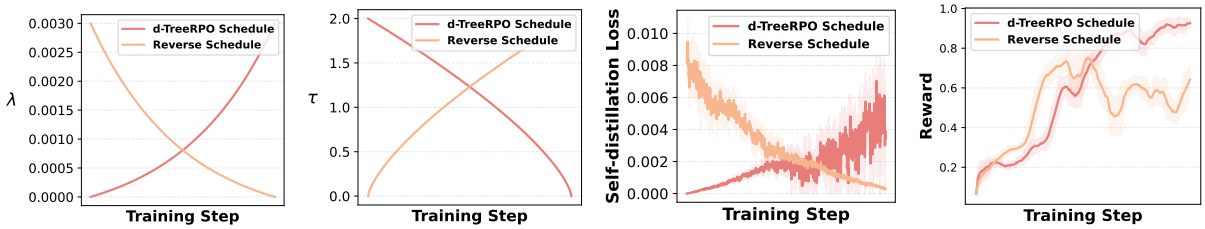
Figure 5: Training curves for the Sudoku task under different ablations of self-distillation loss (LLaDA-8B-Instruct).

| Method | Sudoku | Countdown | GSM8k | Math500 |
|---|---|---|---|---|
| *d*-TreeRPO | $1.25 \pm 1.15$ | $1.45 \pm 1.21$ | $1.32 \pm 0.97$ | $1.42 \pm 1.14$ |
| *d*-TreeRPO w/o distill. | $2.64 \pm 1.97$ | $1.61 \pm 1.30$ | $1.40 \pm 1.01$ | $1.76 \pm 1.32$ |
| *d*-TreeRPO w. div | $2.83 \pm 2.02$ | $1.66 \pm 1.24$ | $1.43 \pm 1.01$ | $1.82 \pm 1.26$ |

Table 3: Comparison of probability estimation error $\log(p_{\text{true}}/\hat{p})$ of $d$-TreeRPO, $d$-TreeRPO without self-distillation loss and $d$-TreeRPO with diversity-promoting loss (LLaDA-8B-Instruct).

**Self-distillation Loss Enhance Determinism**. Figure 5 shows policy entropy and reward trends for the above three settings on the Sudoku task (additional results on more task is in Appendix D.2). Initially, all models exhibit increasing entropy, typical of the exploration phase. Later, model with self-distillation loss sharply reduces entropy, improving policy confidence. In contrast, model with diversity-promoting loss maintains high entropy, reflecting increased variability. Correspondingly, model trained with the complete $d$-TreeRPO achieves the highest reward, followed by the the model without self-distillation loss, while the diversity-focused variant has the lowest reward and unstable training behavior.

**Self-distillation Loss Reduces Estimation Error.** We estimate Eq. (3) via Monte Carlo by sampling




(a) $\lambda$ (b) $\tau$ (c) Self-distillation loss (d) Reward

Figure 6: Training dynamics comparison of $d$-TreeRPO and its reverse-scheduled variant on the Sudoku task with LLaDA-8B-Instruct as the base model.

32 random decoding orders per sample and computing, for each token, the probability of the realized token at the step when it is revealed; this yields $p_{\text{true}}$ and the per-token error $\log(p_{\text{true}}/\hat{p})$. We evaluate converged checkpoints trained with and without self-distillation, as well as with diversity-promoting loss on four tasks, using 100 samples per task, and report the mean and standard deviation of $\log(p_{\text{true}}/\hat{p})$. As shown in Table 3, $d$-TreeRPO with self-distillation loss achieve the smallest estimation error, proving its effectiveness.

**Performance Results.** Furthermore, we report the final evaluation performance of these three settings across all four tasks in Table 2. The results align with our training-time observations: the full $d$-TreeRPO algorithm consistently outperforms the other two variants. Removing the self-distillation loss leads to a noticeable drop in performance, while adding the diversity-promoting loss has the worst performance. These findings provide compelling evidence for the practical effectiveness of our proposed self-distillation loss.

### 4.5 Analysis of Time-scheduled Design

**Comparison with Reverse-Scheduled Self-Distillation Loss.** To assess the design choice of increasing $\lambda(t)$ and decreasing $\tau(t)$ in our self-distillation loss, we tested a reverse schedule for $\lambda(t)$ and $\tau(t)$. Figure 6 compares the forward schedule used in $d$-TreeRPO with the reverse schedule, showing the evolution of $\lambda$, $\tau$, self-distillation loss, and reward during training. While the reverse schedule initially yields faster reward growth due to a strong distillation effect, its performance drops after reaching a reward of 0.75, highlighting insufficient generalization. This analysis supports the forward scheduling used in $d$-TreeRPO.

### 4.6 Computational Overhead

**Computational Analysis of *d*-TreeRPO.** The computational cost of $d$-TreeRPO stems mainly from two components: (1) the rollout phase and (2) loss computation with gradient updates. For the rollout phase, a tree with height $H$ and branch factor $B$ involves $\sum_{i=1}^{H} B^i = \frac{B(B^H-1)}{B-1}$ tree steps. With $N$ denoising steps for a complete generation, each tree step includes $N/H$ denoising steps. Thus, constructing a single tree requires $\left(\frac{B(B^H-1)}{B-1}\right) \cdot \frac{N}{H}$ denoising steps. During loss computation and gradient updates, $\frac{B(B^H-1)}{B-1}$ forward passes are needed. To optimize memory usage, we compute loss for one subtree at a time during each update.

| Method | Batch (s) | Update (s) | Conv. (h) | Acc (%) |
|---|---|---|---|---|
| Diffu-GRPO | 109 | 9.08 | ≈24 | 12.9 |
| wd1 | 87 | 7.25 | ≈24 | 25.2 |
| SAPO | 423 | 35.25 | ≈72 | 20.3 |
| GDPO | 153 | 12.72 | ≈48 | 25.7 |
| TraceRL | 604 | 43.14 | ≈48 | 25.6 |
| *d*-TreeRPO | 598 | 9.96 | ≈48 | 92.9 |

Table 4: Training time comparison between $d$-TreeRPO and other dLLM RL baselines on the Sudoku task using LLaDA-8B-Instruct as base model, including the training time for processing a batch, the time per parameter update, and total time to convergence. All methods are tested under 8 H20 GPUs, batch size = 4/device.

**Training Time Comparison.** To demonstrate the efficiency of $d$-TreeRPO, we compared its training time with other dLLM RL baselines under the same GPU setup (8 H20 GPUs) and train batch size (4 per device). As shown in Table 4, $d$-TreeRPO achieves moderate training speed: slower than Diffu-GRPO and wd1, comparable to TraceRL and GDPO, but faster than SAPO. Despite this, its final performance surpasses all baselines significantly, making it a practical and effective choice.

## 5 Conclusion

We introduced $d$-TreeRPO, a reliable RL algorithm for dLLMs. $d$-TreeRPO organizes rollouts as trees and propagates rewards bottom-up to acquire granular advantage signals. The incorporation of a self-distillation loss enhances determinism in later training stages and reduces estimation error. Experimental results demonstrate that $d$-TreeRPO consistently outperforms baselines across multiple reasoning tasks, proving its effectiveness and practicality.

## Limitations

While *d*-TreeRPO provides a reliable RL framework for dLLMs, several limitations remain. Our experiments primarily focus on reasoning tasks with automatically verifiable outcome rewards (e.g., exact-solution puzzles and correctness-graded math problems), where *d*-TreeRPO can propagate reliable signals through tree-structured rollouts. Extending the framework to settings with less verifiable or more subjective objectives and studying how to construct reliable intermediate credit assignment in such cases remain important future directions. In addition, our evaluation is conducted in a zero-shot manner and mainly reports pass@1 score. Incorporating broader evaluation protocols would provide a more comprehensive assessment.

## References


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

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, and 1 others. 2025. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*.

Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. 2025. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*.

Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. 2025. Reinforcing the diffusion chain of lateral thought with diffusion language models. *arXiv preprint arXiv:2505.10446*.

Xianzhi Li, Ethan Callanan, Xiaodan Zhu, Mathieu Sibue, Antony Papadimitriou, Mahmoud Mahfouz, Zhiqiang Ma, and Xiaomo Liu. 2025. Entropy-aware branching for improved mathematical reasoning. *arXiv preprint arXiv:2503.21961*.

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*.

Nianyi Lin, Jiajie Zhang, Lei Hou, and Juanzi Li. 2025. Boundary-guided policy optimization for memory-efficient rl of diffusion large language models. *arXiv preprint arXiv:2510.11683*.

Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. 2024. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*.

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*.

Jingyang Ou, Jiaqi Han, Minkai Xu, Shaoxuan Xu, Jianwen Xie, Stefano Ermon, Yi Wu, and Chongxuan Li. 2025. Principled rl for diffusion llms emerges from a sequence-level perspective. *arXiv preprint arXiv:2512.03759*.

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

Kevin Rojas, Jiahe Lin, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, Molei Tao, and Wei Deng. 2025. Improving reasoning for diffusion language models via group diffusion policy optimization. *arXiv preprint arXiv:2510.08554*.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, and 1 others. 2025. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*.

Hongze Tan, Jianfei Pan, Jinghao Lin, Tao Chen, Zhihang Zheng, Zhihao Tang, and Haihua Yang. 2025. Gtpo and grpo-s: Token and sequence-level reward shaping with policy entropy. *arXiv preprint arXiv:2508.04349*.

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

Guanghan Wang, Yair Schiff, Gilad Turok, and Volodymyr Kuleshov. 2025a. d2: Improved techniques for training reasoning diffusion language models. *arXiv preprint arXiv:2509.21474*.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. 2025b. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv:2509.06949*.

Shaoan Xie, Lingjing Kong, Xiangchen Song, Xinshuai Dong, Guangyi Chen, Eric P Xing, and Kun Zhang. 2025a. Step-aware policy optimization for reasoning in diffusion large language models. *arXiv preprint arXiv:2510.01544*.

Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, and 1 others. 2025b. Dreamcoder 7b: An open diffusion language model for code. *arXiv preprint arXiv:2509.01142*.

Jingyi Yang, Guanxu Chen, Xuhao Hu, and Jing Shao. 2025a. Taming masked diffusion language models via consistency trajectory reinforcement learning with fewer decoding step. *arXiv preprint arXiv:2509.23924*.

Kai Yang, Xin Xu, Yangkun Chen, Weijie Liu, Jiafei Lyu, Zichuan Lin, Deheng Ye, and Saiyong Yang. 2025b. Entropic: Towards stable long-term training of llms via entropy stabilization with proportional-integral control. *arXiv preprint arXiv:2511.15248*.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. 2025c. Mmada: Multimodal large diffusion language models. *arXiv preprint arXiv:2505.15809*.

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

Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Quanquan Gu. 2023. Diffusion language models can perform many tasks with scaling and instruction-finetuning. *arXiv preprint arXiv:2308.12219*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Hanning Zhang, Pengcheng Wang, Shizhe Diao, Yong Lin, Rui Pan, Hanze Dong, Dylan Zhang, Pavlo Molchanov, and Tong Zhang. 2024. Entropy-regularized process reward model. *arXiv preprint arXiv:2412.11006*.

Lingzhe Zhang, Liancheng Fang, Chiming Duan, Minghua He, Leyi Pan, Pei Xiao, Shiyu Huang, Yunpeng Zhai, Xuming Hu, Philip S Yu, and 1 others. 2025. A survey on parallel text generation: From parallel decoding to diffusion language models. *arXiv preprint arXiv:2508.08712*.

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

Siyan Zhao, Mengchen Liu, Jing Huang, Miao Liu, Chenyu Wang, Bo Liu, Yuandong Tian, Guan Pang, Sean Bell, Aditya Grover, and 1 others. 2025b. Inpainting-guided policy optimization for diffusion large language models. *arXiv preprint arXiv:2509.10396*.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, and 1 others. 2025. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and 1 others. 2025a. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, and 1 others. 2025b. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*.

Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, and 1 others. 2025c. Llada-moe: A sparse moe diffusion language model. *arXiv preprint arXiv:2509.24389*.

## A Proof of Theorem 1: High-probability Estimation Error Bound

**Setup and Notation.** We consider one rollout *tree step* in the tree-structured generation. Let $p$ denote the *parent state*, i.e., the prompt together with a partially masked completion. In this tree step, the diffusion decoder reveals $k$ masked positions with index set $\mathcal{D} = \{d_1, \ldots, d_k\}$, producing the realized tokens $o^{d_1}, \ldots, o^{d_k}$.

A *revelation order* is a permutation $\sigma = (\sigma_1, \ldots, \sigma_k) \in \Omega$, where $\Omega$ is the set of all permutations of $\mathcal{D}$. The decoding strategy induces a distribution $\mathcal{Q}$ over $\Omega$, i.e., $\sigma \sim \mathcal{Q}(\sigma)$. At sub-step $j$, the position revealed is $\sigma_j \in \mathcal{D}$.

For a given order $\sigma$, define the *path-wise probability* at sub-step $j$ ($j \in [1, k]$) as

$$q_j(\sigma) = f_\theta^{\sigma_j}\Big(o^{\sigma_j} \mid p, \{o^{\sigma_1}, \ldots, o^{\sigma_{j-1}}\}, \texttt{[MASK]}_{\text{else}}\Big), \tag{21}$$

where $\{o^{\sigma_1}, \ldots, o^{\sigma_{j-1}}\}$ denotes the already revealed realized tokens inserted back into the current completion state, and all other unrevealed positions remain [MASK].

**Path-wise Probability.** For any position $d \in \mathcal{D}$ and order $\sigma \in \Omega$, let $\tau(d, \sigma) \in \{1, \ldots, k\}$ be the unique index such that $\sigma_{\tau(d,\sigma)} = d$. Thus, the realized token at position $d$ is revealed at sub-step $\tau(d, \sigma)$, and the corresponding path-wise probability is $q_{\tau(d,\sigma)}(\sigma)$.

**Single-time Forward Pass Estimate.** The single-time forward pass estimate is

$$\hat{p}_d := f_\theta^d(o^d \mid p). \tag{22}$$

**Unbiased Expected Probability.** We also recall the unbiased probability (expectation over all decoding orders) for completeness:

$$p_d^{\text{true}} := \mathbb{E}_{\sigma\sim\mathcal{Q}}\big[q_{\tau(d,\sigma)}(\sigma)\big]. \tag{23}$$

Note that Theorem 1 bounds the deviation between $\hat{p}_d$ and the random path-wise probability $q_{\tau(d,\sigma)}(\sigma)$ with high probability over $\sigma$, which in turn provides an explanation for why $\hat{p}_d$ can serve as a reliable proxy when estimating the expectation $p_d^{\text{true}}$.

**Quantile-based confidence gap.** Fix a confidence level $\delta \in (0, 1)$. For each decoded position $d \in \mathcal{D}$, define the $(1-\delta)$-quantile of the random variable $q_{\tau(d,\sigma)}(\sigma)$ under $\sigma \sim \mathcal{Q}$ as $q_{d,1-\delta}$, namely

$$\Pr_{\sigma\sim\mathcal{Q}}\big(q_{\tau(d,\sigma)}(\sigma) \geq q_{d,1-\delta}\big) \geq 1 - \delta. \tag{24}$$

We then define the (typical-path) confidence gap at level $\delta$:

$$\epsilon_{d,\delta} := \max\{1 - \hat{p}_d,\ 1 - q_{d,1-\delta}\}. \tag{25}$$

By definition, $\hat{p}_d \geq 1 - \epsilon_{d,\delta}$, and also $q_{d,1-\delta} \geq 1 - \epsilon_{d,\delta}$. Combining the latter with (24) yields the key high-probability event:

$$\Pr_{\sigma\sim\mathcal{Q}}\Big(q_{\tau(d,\sigma)}(\sigma) \geq 1 - \epsilon_{d,\delta}\Big) \geq 1 - \delta. \tag{26}$$

**Proof of Theorem 1.** Fix any decoded position $d \in \mathcal{D}$ and assume $\epsilon_{d,\delta} \in [0, 1)$. Consider the event

$$E := \Big\{\sigma \in \Omega :\ q_{\tau(d,\sigma)}(\sigma) \geq 1 - \epsilon_{d,\delta}\Big\}. \tag{27}$$

From (26), we have $\Pr_{\sigma\sim\mathcal{Q}}(E) \geq 1 - \delta$.

On the event $E$, we have the following two inequalities:

$$q_{\tau(d,\sigma)}(\sigma) \geq 1 - \epsilon_{d,\delta}, \tag{28}$$

$$\hat{p}_d \geq 1 - \epsilon_{d,\delta}. \tag{29}$$

Therefore, for any $\sigma \in E$,

$$1 - \epsilon_{d,\delta} \leq \frac{q_{\tau(d,\sigma)}(\sigma)}{\hat{p}_d} \leq \frac{1}{1 - \epsilon_{d,\delta}}. \tag{30}$$

Indeed, the left inequality follows from $q_{\tau(d,\sigma)}(\sigma) \geq 1 - \epsilon_{d,\delta}$ and $\hat{p}_d \leq 1$; the right inequality follows from $q_{\tau(d,\sigma)}(\sigma) \leq 1$ and $\hat{p}_d \geq 1 - \epsilon_{d,\delta}$.

Taking logarithms in (30) gives, for all $\sigma \in E$,

$$\log(1 - \epsilon_{d,\delta}) \leq \log \frac{q_{\tau(d,\sigma)}(\sigma)}{\hat{p}_d} \leq -\log(1 - \epsilon_{d,\delta}), \tag{31}$$

which implies

$$\left|\log \frac{q_{\tau(d,\sigma)}(\sigma)}{\hat{p}_d}\right| \leq -\log(1 - \epsilon_{d,\delta}). \tag{32}$$

Combining (32) with $\Pr_{\sigma\sim\mathcal{Q}}(E) \geq 1 - \delta$ proves

$$\Pr_{\sigma\sim\mathcal{Q}}\left(\left|\log \frac{q_{\tau(d,\sigma)}(\sigma)}{\hat{p}_d}\right| \leq -\log(1 - \epsilon_{d,\delta})\right) \geq 1 - \delta, \tag{33}$$

which is exactly Eq. (13) in the main text.

**Additional Remarks.**

1. *Why "high probability"?* The randomness comes from $\sigma \sim \mathcal{Q}$. Theorem 1 states that the estimation error is bounded on an event of probability at least $1 - \delta$ w.r.t. $\mathrm{d}\mathcal{Q}$.

2. *Why does confidence tighten the bound?* The bound is monotone in $\epsilon_{d,\delta}$: as $\epsilon_{d,\delta}$ decreases, $-\log(1 - \epsilon_{d,\delta})$ decreases and approaches 0. Thus, higher token confidence (smaller typical-path confidence gap) yields a tighter error bound.

3. *Relation to $p_d^{\text{true}}$.* Theorem 1 bounds the deviation between $\hat{p}_d$ and the random variable $q_{\tau(d,\sigma)}(\sigma)$ for typical decoding orders. Since $p_d^{\text{true}}$ is the expectation of $q_{\tau(d,\sigma)}(\sigma)$ over $\sigma \sim \mathcal{Q}$, reducing the typical-path deviation (smaller $\epsilon_{d,\delta}$) makes $\hat{p}_d$ a more reliable proxy when approximating $p_d^{\text{true}}$.

## B Detailed Descriptions of dLLM RL Baselines

**Diffu-GRPO (Zhao et al., 2025a).** Diffu-GRPO adapts GRPO-style critic-free policy optimization to masked dLLMs by using sparse, outcome-based rewards $r_i$ for each sampled completion $o_i$, and then assigning a group-relative advantage uniformly to all tokens in that completion: $A_{i,k} = r_i - \text{mean}(\{r_j\}_{j=1}^G)$. The key technical choice is its efficient log-probability approximation for dLLMs: instead of marginalizing over denoising orders, it estimates per-token log-probabilities with a single forward pass conditioned on a randomly masked prompt $q'$ and a fully masked completion, i.e., $\hat{\log \pi_\theta}(o_i^k \mid q') = \log f_\theta^k(o_i^k \mid q')$. During optimization, Diffu-GRPO plugs these estimates into the standard GRPO clipped importance ratio and KL-regularized objective, using $\hat{\pi}$ (denoted $\phi$ in their paper) for $\pi_\theta, \pi_{\theta_{\text{old}}}, \pi_{\text{ref}}$, and re-samples $q'$ each inner update as a form of regularization while keeping $q, o_i, r_i$ fixed.

**VRPO (Zhu et al., 2025a).** VRPO applies DPO-style preference optimization to masked dLLMs using offline preference pairs $(q, o^w, o^l)$, where the training signal encourages the preferred completion $o^w$ to have a higher relative score than $o^l$ without learning an explicit reward model. Since exact $\log \pi_\theta(o \mid q)$ is intractable for dLLMs, VRPO estimates sequence log-probabilities via an *ELBO-based* surrogate $B_\pi(o \mid q) \le \log \pi(o \mid q)$ and substitutes it into the DPO score:

$$\begin{aligned} s_\theta(o^w, o^l) = \beta\Big(B_{\pi_\theta}(o^w \mid q) - B_{\pi_{\text{ref}}}(o^w \mid q)\Big) \\ - \beta\Big(B_{\pi_\theta}(o^l \mid q) - B_{\pi_{\text{ref}}}(o^l \mid q)\Big). \end{aligned} \tag{34}$$

In practice, each $B_\pi(o \mid q)$ is approximated by a *doubly* Monte Carlo estimator with $n_t$ samples of the diffusion noise level (timestep) and $n_{z_t}$ masked-state samples per timestep, which can induce substantial variance in the resulting preference score. The main contribution of VRPO is to reduce this variance—most notably via an *optimal allocation* of a fixed sampling budget across timesteps (favoring larger $n_t$ and smaller $n_{z_t}$), and via antithetic sampling that shares Monte Carlo randomness between the current policy and the reference when forming the score differences. A remaining limitation is that, even with variance reduction, the method does not explicitly account for the gap between the ELBO and the unbiased expected log-likelihood, since the ELBO is only a lower bound. In their experiments, VRPO is applied to LLaDA-8B-Instruct, producing the released model LLaDA-1.5.

**wd1 (Tang et al., 2025).** wd1 is a diffusion-LM RL method that avoids GRPO/PPO-style *policy ratios* and instead optimizes a *weighted log-likelihood* objective. For each prompt $q$, it samples a group of completions $\{o_i\}_{i=1}^G$, computes a outcome reward $R(q, o_i)$ from a task-specific verifier, and forms *group-relative advantages* $A_i = R(q, o_i) - \text{mean}(R(q, o_{1:G}))$. Since the reward is only available at the completion level, the same scalar signal is effectively broadcast to all tokens in $o_i$ via the sample weight, by mapping $\{A_i\}$ to normalized weights

$$w_i^+ = \frac{\exp(\psi A_i)}{\sum_{j=1}^G \exp(\psi A_j)}, w_i^- = \frac{\exp(-\psi A_i)}{\sum_{j=1}^G \exp(-\psi A_j)}, \tag{35}$$

and optimizing

$$\mathcal{L}_{\text{wd1}}(\theta) = \mathbb{E}\Big[\sum_{i=1}^G \big(-w_i^+ + w_i^-\big) \log \pi_\theta(o_i \mid q)\Big]. \tag{36}$$

For probability estimation, wd1 still requires approximating the diffusion policy likelihood and adopts the single-pass mean-field estimator, $\log \pi_\theta(o_i \mid q) \approx \sum_k \log \pi_\theta(o_i^k \mid q')$, where $q'$ is obtained by randomly masking the prompt at each gradient step.

**SAPO (Xie et al., 2025a).** SAPO extends diffu-GRPO by augmenting the outcome reward with a process (step-aware) reward that estimates whether an intermediate denoising interval makes progress toward a correct final answer. For each prompt $q$, it samples a group of completions $\{o_i\}_{i=1}^G$ and computes scalar outcome rewards $r_i$, then forms the group-relative advantage $A_i = r_i - \text{mean}(\{r_j\}_{j=1}^G)$. It further estimates a step-aware reward $R_{\text{process}}$ by comparing the expected outcome reward of rollouts continued from two intermediate diffusion states; in practice it often uses the efficient case $t_2 = T$ (all `[MASK]`), reusing the original $G$ rollouts for the baseline term and sampling $N$ additional rollouts from a randomly chosen intermediate state for the other term. SAPO combines these signals via

$$A_i^{\text{total}} = A_i + \mathbf{1}[A_i > 0] \cdot R_{\text{process}}. \tag{37}$$

During optimization, SAPO directly replaces the GRPO advantage with $A_i^{\text{total}}$ in the same PPO/GRPO-style clipped objective, i.e., the scalar $A_i^{\text{total}}$ is *broadcast* to all tokens of $o_i$ (no token/step-specific advantage), and is used with token-wise likelihood ratios and a KL regularizer. For probability estimation, SAPO inherits diffu-

GRPO's masked-dLLM likelihood approximation, using mean-field per-token log-probabilities under randomly masked conditioning to form ratios between the current and old policies (and the KL term to a reference).

**GDPO (Rojas et al., 2025).** GDPO is a GRPO-style verifier-RL algorithm for diffusion LMs that keeps the same *sequence-level* outcome reward $R(q, o_i)$ and group-relative advantage $A_i = R_i - \mathrm{mean}(R_{1:G})$, but changes the *probability estimation* and the *importance ratio* from token-level (mean-field) to *sequence-level* via an ELBO surrogate. Concretely, GDPO defines a sequence-level ratio using ELBO estimates

$$r_i(q, o_i) \;=\; \frac{L_{\mathrm{ELBO}}(o_i \mid q; \theta)}{L_{\mathrm{ELBO}}(o_i \mid q; \theta_{\mathrm{old}})}, \tag{38}$$

and plugs $r_g$ and $A_g$ into the same PPO/GRPO-style clipped objective (with a KL penalty to $\pi_{\mathrm{ref}}$); thus the scalar advantage is still broadcast at the sample level rather than giving token-/step-specific credit. The key technical contribution is a lower-variance, budget-efficient ELBO estimator: it replaces the outer Monte Carlo over diffusion time $t$ with a small set of deterministic quadrature points $\{(t_n, w_n)\}_{n=1}^{N}$ (typically $N = 2$ or 3) and uses a lightweight inner Monte Carlo over masked states, yielding a semi-deterministic approximation $L_{\mathrm{ELBO}}(o \mid q) \approx \sum_{n=1}^{N} w_n \, \ell(\pi_\theta; o, q, t_n)$. Notably, like other ELBO-based approaches, GDPO optimizes and forms ratios using this lower bound and does not explicitly account for the gap relative to the unbiased expected probability.

**TraceRL (Wang et al., 2025b).** TraceRL is a trajectory-aware PPO-style framework for dLLMs that optimizes on the model's actual decoding trajectory rather than random masking. For each prompt $q$, a rollout yields a trajectory of intermediate states $(z_1, \ldots, z_0)$ with $z_0 = o$ (the final completion), and the sets of tokens revealed at each step (optionally aggregated by a shrinkage factor $s$ for efficiency). It starts from a verifiable outcome reward $r(z_0)$ and can provide *fine-grained* credit assignment by training a diffusion-based value model $V_{\theta_v}$ and constructing token-/step-wise advantages $A_j$ (e.g., GAE-style) along $(z_1, \ldots, z_0)$ instead of broadcasting a single scalar advantage to all tokens; however, because these advantages depend on a learned value model, they bring the usual risk of value mis-specification and reward/value hacking relative to purely verifier-based outcome rewards. For probability estimation, TraceRL computes PPO importance ratios *along the realized trajectory*: for a token $o_j$ decoded when transitioning from $z_t$ to $z_{t-1}$, it uses

$$r_j \;=\; \frac{\pi_\theta(o_j \mid z_t)}{\pi_{\mathrm{old}}(o_j \mid z_t)}. \tag{39}$$

A remaining gap is that these likelihood terms are defined conditional on the single sampled trajectory $(z_1, \ldots, z_0)$, whereas the true $\pi(o \mid q)$ would require marginalizing over all possible decoding trajectories leading to $z_0 = o$; TraceRL does not explicitly quantify or correct this trajectory-marginalization gap.

# C Training Details

## C.1 Prompts

Prompts used in Sudoku, Countdown, GSM8k and Math500 are as follows:

```
### Sudoku

Please solve the following 4x4 Sudoku puzzle.
The puzzle is provided as a 16-character
string reading left-to-right, top-to-bottom,
where '0' represents empty cells.

Rules:
- Fill empty cells with digits 1-4
- Each row must contain digits 1-4 exactly
once
- Each column must contain digits 1-4 exactly
once
- Each 2x2 box must contain digits 1-4 exactly
once

Important: Your solution must be a COMPLETE
16-character string with only the digits 1-4,
representing your final solved grid.

Respond in this exact format:
<reasoning>
Your step-by-step solving process
</reasoning>
<answer>
[16-character solution string with no spaces
or separators]
</answer>

Solve the following Sudoku puzzle: {PUZZLE}
```

```
### Countdown

Respond in the following format:
<reasoning>
...
</reasoning>
<answer>
...
</answer>
```

```
Using only the numbers {NUMBERS}, create an
arithmetic expression that evaluates to
exactly {TARGET}. You must use all numbers
from the list, and each number must be used
exactly once. You may use the operations +, -,
*, and / as needed. After reasoning, provide
only your final expression inside
<answer></answer> tags without including an
equals sign or the target number. For example,
if the numbers are [2, 3, 4] and the target is
5, a valid answer is: <answer>2*4-3</answer>.
```

```
### GSM8k

Respond in the following format:
<reasoning>
...
</reasoning>
<answer>
...
</answer>

You are a math expert. You will be given a
question to solve. Solve it step by step. Put
only the final answer (the number) in the
<answer> </answer> tag without any other text.

{QUESTION}
```

```
### Math500

Respond in the following format:
<reasoning>
...
</reasoning>
<answer>
...
</answer>

You are a math expert. You will be given a
question to solve. Solve it step by step. Wrap
the final answer in a \\boxed{{}}.

{QUESTION}
```

### C.2 Reward Functions

**Sudoku.** The reward is defined as the percentage of correctly filled previously empty cells, with a maximum value of 1 and a minimum value of 0.
**Countdown.** The reward is 1.0 if the provided expression achieves the target using the given numbers. If the numbers used are correct but the target is missed, the reward is 0.1. In all other cases, the reward is 0.
**GSM8k.** During training, we utilize a correctness-based reward, assigning a value of 1.0 if the provided answer matches the ground truth.
**Math500.** Similarly, during training, we employ a correctness-based reward system, assigning a reward of 1.0 if the given answer matches the ground truth.

### C.3 Dataset Links

Following previous works (Zhao et al., 2025a; Tang et al., 2025), we adopt the 4×4 Sudoku training and test sets provided by Diffu-GRPO (Zhao et al., 2025a) for Sudoku. For Countdown, models are trained on the 3to4 Countdown task training set[2] and evaluated using the test set provided by Diffu-GRPO. For GSM8K[3] and Math500[4], we strictly follow their official training-test splits for both training and evaluation.

## D Training Dynamics

### D.1 Reward Curves Compared with Open-source Baselines

Figure 7 compares the training reward curves of *d*-TreeRPO against two open-source baselines: Diffu-GRPO (Zhao et al., 2025a) and wd1 (Tang et al., 2025). We include these baselines because both our training code and wd1's implementation are built upon the Diffu-GRPO framework, ensuring better alignment of training parameters and making the curve comparison more meaningful. The figure specifically compares the training rewards of these three methods after processing *equivalent amounts of data* (i.e., at the same point on the x-axis) across different tasks. It demonstrates that *d*-TreeRPO consistently outperforms the baselines across all tasks, with particularly significant advantages in planning tasks like Sudoku and Countdown. wd1 achieves the second-highest performance, followed by Diffu-GRPO.

### D.2 More Training Dynamics Regarding Self-distillation Loss

In this section, we present the evolution of the self-distillation loss over the course of training on all evaluated tasks, both with and without the weighting parameter $\lambda(t)$ (Figures 8 and 9, respectively). We also report the entropy curves (Figure 10) and training reward curves (Figure 11) under three settings: the full *d*-TreeRPO algorithm, a variant without the self-distillation loss, and a variant with the diversity-promoting loss. These results serve as supplementary experiments to Section 4.4 in the main text.

[2] https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4
[3] https://huggingface.co/datasets/openai/gsm8k
[4] train: https://huggingface.co/datasets/ankner/math-500, test: https://huggingface.co/datasets/HuggingFaceH4/MATH-500

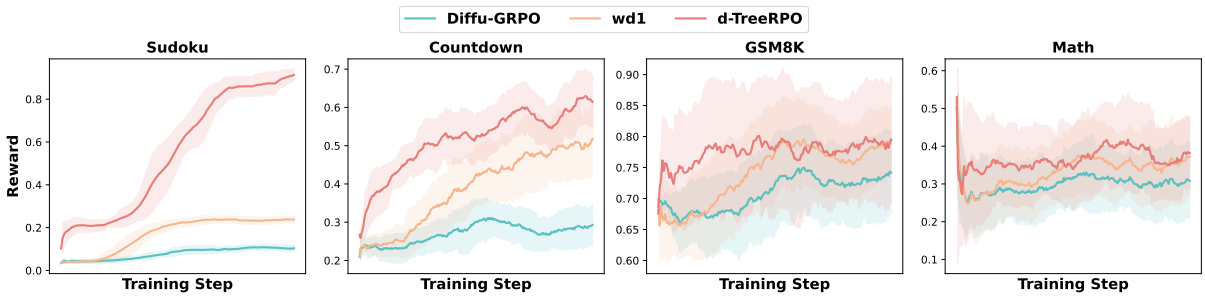
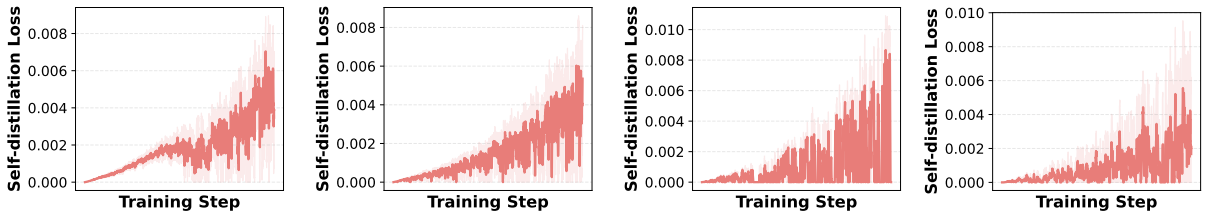
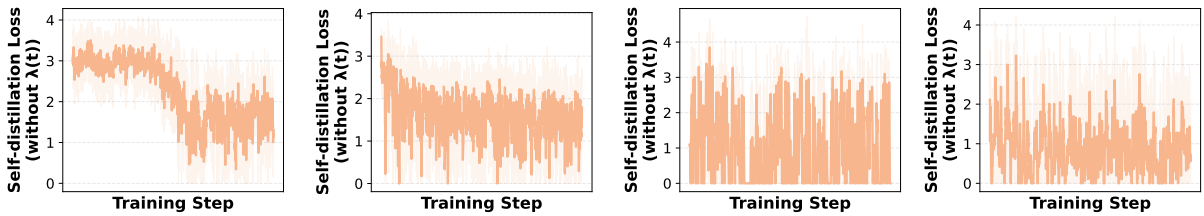



Figure 7: Comparison of training reward curves between $d$-TreeRPO and open-source baselines: Diffu-GRPO (Zhao et al., 2025a) and wd1 (Tang et al., 2025).




(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 8: Self-distillation loss with $\lambda(t)$ over the course of training on the four evaluated tasks.




(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

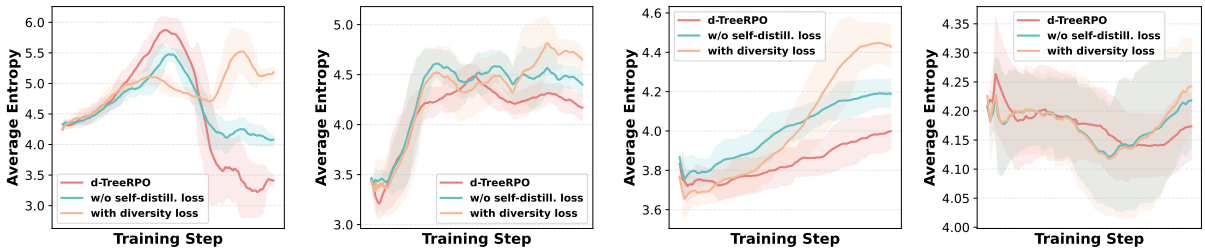
Figure 9: Self-distillation loss without $\lambda(t)$ over the course of training on the four evaluated tasks.

As shown in Figure 8, the self-distillation loss on all four tasks gradually increases as training progresses, which aligns with our design motivation: we impose a relatively loose constraint early on to encourage exploration, and then gradually tighten it later to promote determinism. This reduces probability estimation error and leads to better convergence performance. If we instead examine the self-distillation loss without multiplying by $\lambda(t)$, Figure 9 shows an overall decreasing trend over training, which is particularly pronounced on planning tasks such as Sudoku and Countdown. This is because, as the self-distillation objective is strengthened, the policy progressively move closer to the predictions with positive advantages, causing the loss to decrease.

To further confirm the role of the self-distillation loss, we extend the experiments in Section 4.4 to all four datasets. The entropy and reward curves are shown in Figures 10 and 11, respectively. We observe that, across all four datasets, the full $d$-TreeRPO algorithm with the self-distillation loss achieves the lowest entropy in the later stage of training, followed by the variant without the self-distillation loss, while the variant with the diversity-promoting loss exhibits the highest entropy. The reward curves show a consistent pattern at convergence: the full $d$-TreeRPO achieves the highest reward, the variant without self-distillation is second, and the diversity-promoting variant performs the worst. Since the overall performance gains on GSM8K and Math500 are relatively small for all methods, the benefit of adding the self-distillation loss is less pronounced, though still observable.

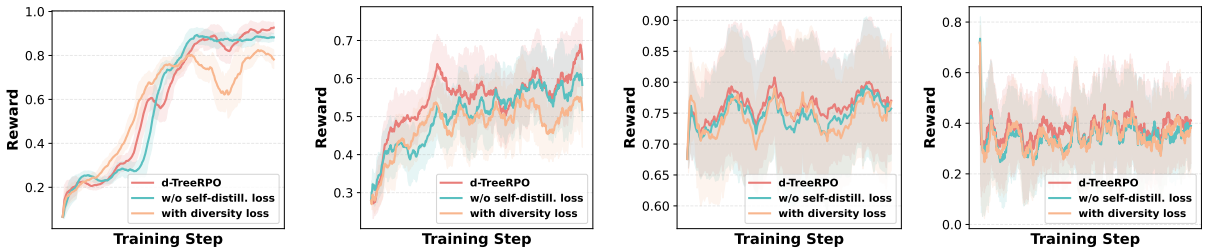
## E Hyper-Parameter Analysis

In this section, we provide additional hyper-parameter analyses for $d$-TreeRPO by reporting evaluation performance on the four datasets under different hyper-parameter settings.




(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 10: Entropy curves over the course of training on the four evaluated tasks under three settings: the full $d$-TreeRPO algorithm, a variant without the self-distillation loss, and a variant with the diversity-promoting loss.




(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 11: Reward curves over the course of training on the four evaluated tasks under three settings: the full $d$-TreeRPO algorithm, a variant without the self-distillation loss, and a variant with the diversity-promoting loss.

### E.1 Hyper-parameters regarding target distribution in self-distillation loss: $\tau_{max}$ and $\beta$

Recall that the self-distillation loss uses a KL constraint to pull the current policy toward a target distribution obtained by voting among positive-advantage samples. This target distribution is advantage-weighted and defined as:

$$P^{\sigma_i}_{\text{target}}(v) = \frac{\sum_{c \in C_p^+} w_c \cdot \mathbf{1}[v_c^{\sigma_i} = v]}{\sum_{c \in C_p^+} w_c}, \quad \forall v \in \mathcal{V}. \tag{40}$$

The weight $w_c$ is computed by applying a temperature-scaled softmax to the advantages:

$$w_c = \frac{\exp\left(A_p^c/\tau(t)\right)}{\sum_{c' \in C_p^+} \exp\left(A_p^{c'}/\tau(t)\right)}, \quad \forall c \in C_p^+. \tag{41}$$

A smaller $\tau(t)$ makes the target distribution more sharply concentrated on the branch with the largest positive advantage, whereas a larger $\tau(t)$ yields a more uniform weighting across all positive-advantage branches. In our implementation, we control $\tau(t)$ using a time-dependent schedule:

$$\tau(t) = \tau_{\max} \cdot \left(1 - \frac{t}{T}\right)^{\beta}, \quad (\beta \in (0, 1]), \tag{42}$$

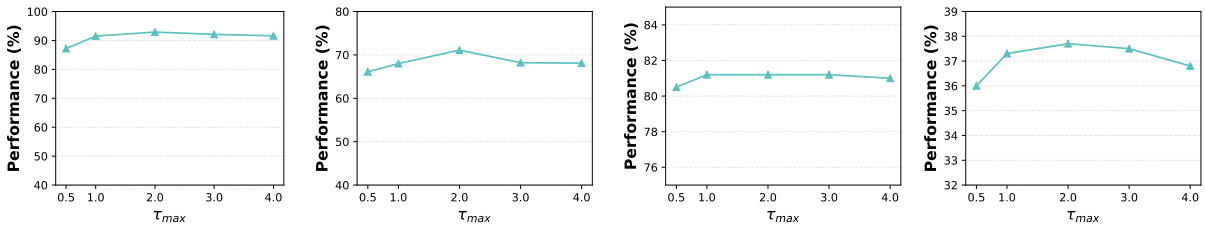
where $T$ denotes the total number of training steps and $t$ is the current training step.

This schedule introduces two hyper-parameters, $\tau_{\max}$ and $\beta$. A larger $\tau_{\max}$ corresponds to a higher overall temperature and, consequently, higher entropy. As shown in Figure 12, $d$-TreeRPO is generally robust to the choice of $\tau_{\max}$. Upon closer inspection, overly large or overly small values of $\tau_{\max}$ can slightly degrade performance, either by reducing determinism or by making the policy overly deterministic (thereby hindering exploration). Therefore, we set $\tau_{\max} = 2.0$ in our main experiments.

The exponent $\beta \in (0, 1]$ controls the decay rate: smaller $\beta$ leads to a slower decrease in $\tau(t)$ early in training and a much faster drop later. When $\beta$ is close to 0, the entropy-reducing effect of the self-distillation loss becomes weaker, while $\beta$ close to 1 yields an approximately linear schedule. As shown in Figure 13, $d$-TreeRPO is also fairly robust to the choice of $\beta$. When $\beta$ approaches 0, the behavior becomes close to constructing the self-distillation target distribution using uniform weights over positive samples, which slightly degrades performance. When $\beta$ approaches 1, the performance changes only marginally. Based on the performance trends, we set $\beta = 0.7$ in our main experiments.

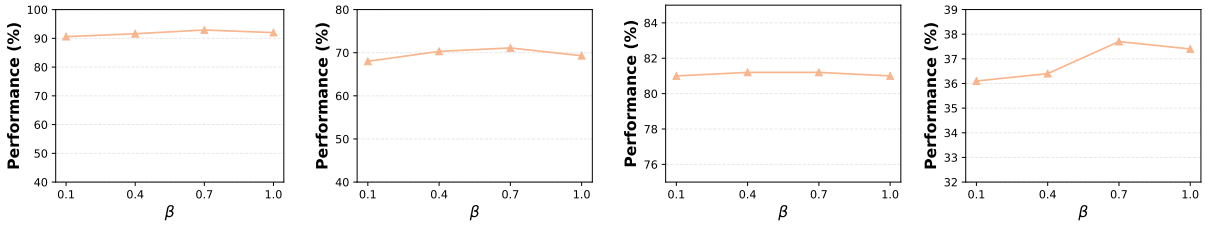
### E.2 Hyper-parameters regarding $\lambda(t)$: $\gamma$ and $\lambda_{max}$

In this subsection, we focus on how the coefficient $\lambda(t)$ in the self-distillation loss affects performance.



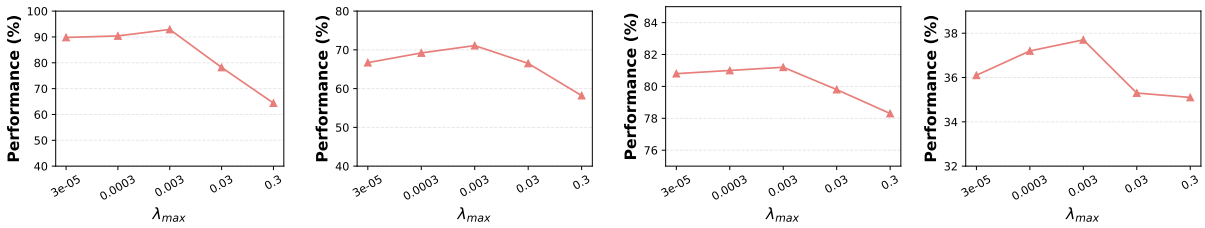

(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 12: Evaluation performance under different $\tau_{max}$ settings on the four evaluated tasks.



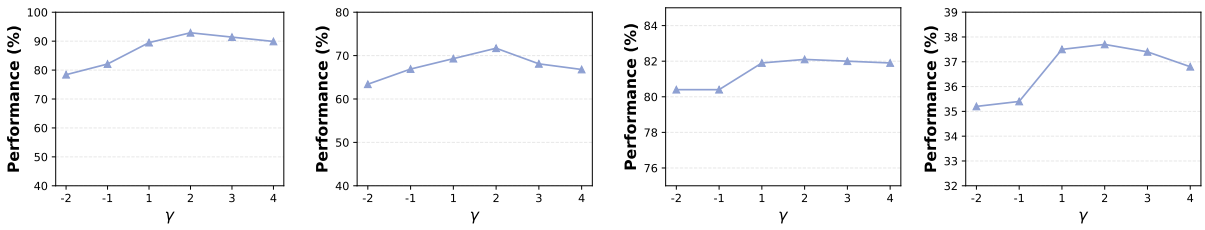

(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 13: Evaluation performance under different $\beta$ settings on the four evaluated tasks.




(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 14: Evaluation performance under different $\lambda_{max}$ settings on the four evaluated tasks.




(a) Sudoku (b) Countdown (c) GSM8k (d) Math500

Figure 15: Evaluation performance under different $\gamma$ settings on the four evaluated tasks.

Recall that $\lambda(t)$ is defined as

$$\lambda(t) = \lambda_{\max} \cdot \frac{e^{\gamma t/T} - 1}{e^{\gamma} - 1}, \tag{43}$$

which introduces two hyper-parameters: $\lambda_{\max}$ and $\gamma$. Larger $\lambda_{\max}$ increases the overall scale of the self-distillation loss and thus strengthens its effect in promoting determinism. As shown in Figure 14, when $\lambda_{\max}$ is too small, the behavior becomes similar to removing the self-distillation loss altogether. In contrast, when $\lambda_{\max}$ is too large—especially on the order of 0.3—it can substantially hinder exploration and degrade performance. Therefore, to balance the exploration–exploitation trade-off, and based on the results in Figure 14, we set $\lambda_{\max} = 3 \times 10^{-3}$ in our main experiments.

The parameter $\gamma$ controls how fast $\lambda(t)$ increases. When $\gamma > 0$, $\lambda(t)$ grows slowly at the beginning and accelerates later; when $\gamma < 0$, it exhibits the opposite behavior. As shown in Figure 15, settings with $\gamma > 0$ generally outperform those with $\gamma < 0$, because increasing the weight too quickly

in the early stage can suppress exploration. A schedule with an increasing first derivative is therefore preferable. Based on these results, we choose $\gamma = 2$ as the default setting in our main experiments.

## F Case Studies

This section presents case studies of *d*-TreeRPO, highlighting its advantages over alternative training methods. We compare the original LLaDA-8B-Instruct with models trained using Diffu-GRPO, wd1, and *d*-TreeRPO on the same GSM8K question, as shown in Figures 16, 17, 18, and 19, respectively. Among these methods, only *d*-TreeRPO produces both a correct reasoning process and the correct final answer.

LLaDA-8B-Instruct

**Question:** Ten treeks weigh as much as three squigs and one goolee. Two treeks and one goolee are equal in weight to one squig. The combined weight of how many treeks equals the weight of one squig?

**Model Output:**

<reasoning>

To solve this problem, we need to establish the relationships between the treeks, squigs, and goolees based on the given information. Let's denote the weight of one treek as $T$, the weight of one squig as $S$, and the weight of one goolee as $G$.

From the problem, we have the following relationships:

1. Ten treeks weigh as much as three squigs and one goolee: $10T = 3S + G$
2. Two treeks and one goolee are equal in weight to one squig: $2T + G = S$

We need to find the number of treeks whose combined weight equals the weight of one squig. From the second relationship, we know that $2T + G = S$. This implies that the combined weight of two treeks and one goolee is equal to the weight of one squig. Therefore, the weight of two treeks equals the weight of one squig. ✗

</reasoning>

<answer>

$\boxed{2}$ ✗

</answer>

Figure 16: A case study of LLaDA-8B-Instruct's response to a GSM8K question.

LLaDA-8B-Instruct + Diffu-GRPO

**Question:** Ten treeks weigh as much as three squigs and one goolee. Two treeks and one goolee are equal in weight to one squig. The combined weight of how many treeks equals the weight of one squig?

**Model Output:**

<reasoning>

To solve this problem, we need to establish the relationships between the treeks, squigs, and goolees based on the given information. Let's denote the weight of one treek as $T$, the weight of one squig as $S$, and the weight of one goolee as $G$.

From the problem, we have the following relationships:

1. Ten treeks weigh as much as three squigs and one goolee: $10T = 3S + G$
2. Two treeks and one goolee are equal in weight to one squig: $2T + G = S$

We need to find the number of treeks whose combined weight equals the weight of one squig. From the second relationship, we know that $2T + G = S$. This means that the combined weight of two treeks and one goolee is equal to the weight of one squig. Therefore, the number of treeks that equal the weight of one squig is two. ✗

</reasoning>

<answer>

$\boxed{2}$ ✗

</answer>

Figure 17: A case study of LLaDA-8B-Instruct trained with Diffu-GRPO responding to a GSM8K question.

LLaDA-8B-Instruct + wd1

**Question:** Ten treeks weigh as much as three squigs and one goolee. Two treeks and one goolee are equal in weight to one squig. The combined weight of how many treeks equals the weight of one squig?

**Model Output:**

<reasoning>

1. Let the weight of one treek be $T$, the weight of one squig be $S$, and the weight of one goolee be $G$
2. From the first equation: $10T = 3S + G$
3. From the second equation: $2T + G = S$
4. Substitute $S = 2T + G$ from the second equation into the first: $10T = 3(2T + G) + G$, $10T = 6 + T + G$, $4T = G$ ✗
5. Now, $S = 2T + 4T = 6T$
6. To find how many treeks equal the weight of one squig: $S = 6T$ which means 6 treeks

</reasoning>

<answer>

$\boxed{6}$ ✗

</answer>

Figure 18: A case study of LLaDA-8B-Instruct trained with wd1 responding to a GSM8K question.

LLaDA-8B-Instruct + $d$-TreeRPO

**Question:** Ten treeks weigh as much as three squigs and one goolee. Two treeks and one goolee are equal in weight to one squig. The combined weight of how many treeks equals the weight of one squig?

**Model Output:**

<reasoning>

Let the weights in terms of a treek be $T$, a squig be $S$, and a goolee be $G$. From the problem statement, we have the following equations:

1. $10T = 3S + G$
2. $2T + G = S$

We need to find the number of treeks that equal the weight of one squig, i.e., $nT = S$.

First, let's solve the second equation for $G$: $G = S - 2T$. Substitute $G$ into the first equation: $10T = 3S + (S - 2T)$, $10T = 4S - 2T$, $10T + 2T = 4S$, $12T = 4S$, $3T = S$.

Thus, the weight of three treeks equals the weight of one squig.

</reasoning>

<answer>

$\boxed{3}$ ✓

</answer>

Figure 19: A case study of LLaDA-8B-Instruct trained with $d$-TreeRPO responding to a GSM8K question.