Diffusion-State Policy Optimization for Masked Diffusion Language Models

Daisuke Oba [1] **Hiroki Furuta** **Naoaki Okazaki** [1 2 3]

Abstract

Masked diffusion language models generate text through iterative masked-token filling, but terminal-only rewards on final completions provide coarse credit assignment for the intermediate filling decisions that shape the generation process. We propose Diffusion-State Policy Optimization (DiSPO), a plug-in credit-assignment layer that directly optimizes intermediate filling decisions. At selected intermediate masked states, DiSPO branches by resampling the currently masked positions from rollout-cached logits, scores the resulting completions, and updates only the newly filled tokens, requiring no additional multi-step diffusion rollouts or optimizer steps. We formalize a fixed-state objective for branched completions and derive a policy-gradient estimator that reuses the same rollouts as terminal-feedback policy optimization. Experiments on LLaDA-8B-Instruct show that DiSPO consistently improves terminal-feedback baselines, including diffu-GRPO and SPG, on math and planning benchmarks under matched rollout compute and optimizer steps, supporting its use as a general plug-in for masked diffusion policy optimization.

1. Introduction

Policy optimization (PO) is widely used to improve the reasoning and alignment behavior of language models, from preference-based RLHF (Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022) to policy-gradient methods (Williams, 1992; Schulman et al., 2015; 2017) and recent reasoning-oriented variants (Shao et al., 2024). Most PO methods, however, learn from scalar rewards on final completions, effectively assigning one sequence-level outcome to many local generation decisions. This coarse credit assignment motivates our central question: *can PO exploit intermediate information already produced during generation, rather than only the final outcome?*

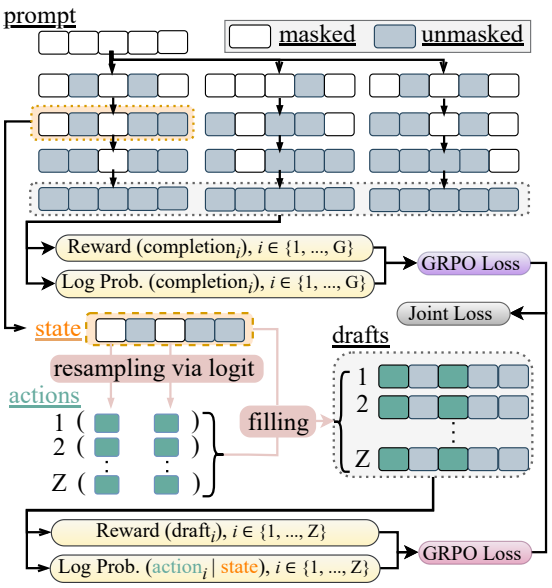


Figure 1. **Conceptual overview.** ***Top***: Terminal-feedback PO assigns reward to the final denoising trajectory. ***Bottom***: DiSPO branches from **intermediate masked states** using cached logits, scores the branched completions with the same reward, and updates only **newly filled tokens.**

This question is particularly natural for masked diffusion language models (MDLMs), which generate text by repeatedly filling masked positions over multiple denoising steps (Austin et al., 2021; Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025; Ye et al., 2025; Gong et al., 2025). During rollout, each denoising step already produces token-level distributions over many masked positions, but terminal-feedback PO trains only from a scalar final-completion reward, leaving this intermediate state-action information largely unused for fine-grained credit assignment (Zhao et al., 2025b; Wang et al., 2026). Prior work goes beyond purely terminal feedback, e.g., via reward shaping based on intermediate states or trajectory-level comparisons at fixed steps (Xie et al., 2025; He et al., 2025), densifying supervision over realized denoising trajectories. In contrast, we make the state-conditioned filling action the unit of optimization: under the fixed intermediate masked sequence, we compare alternative assignments to the current masks.

[1]Institute of Science Tokyo [2]AIST [3]NII LLMC. Correspondence to: Daisuke Oba <daisuke.oba@nlp.comp.isct.ac.jp>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

To exploit intermediate rollout information for state-conditioned credit assignment, we propose *Diffusion-State Policy Optimization* (DISPO), a plug-in objective for MDLM policy optimization (Fig. 1). Motivated by policy-gradient learning (Williams, 1992; Sutton et al., 1999), DISPO treats an intermediate masked sequence as the **state** and its mask fillings as the **action**. At selected states, it fixes the masked sequence, samples alternative fillings from rollout-cached logits, scores the resulting branched completions with the base terminal reward function, and updates only the newly filled tokens. Since DISPO only requires intermediate masked states and masked-position logits already produced during rollout, it adds same-state training comparisons without additional multi-step diffusion rollouts or optimizer updates. Thus, DISPO can be plugged into terminal-feedback MDLM PO methods as a credit-assignment objective (Alg. 1).

We formalize DISPO as a fixed-state expected-return objective over branched completions and derive a policy-gradient estimator for state-conditioned mask filling (Theorem A.1). This objective can be mixed with terminal-feedback PO using the same rollouts, giving a joint objective in expectation and supporting DISPO as a plug-in objective (Theorem A.2). Our analysis further motivates updating only newly filled tokens and averaging over same-state branches, which reduce variance and improve the efficiency of the step-wise update (Propositions A.3 and A.4).

We evaluate DISPO on LLaDA-8B-Instruct (Nie et al., 2025) by integrating it with two terminal-feedback MDLM policy-optimization methods, diffu-GRPO (Zhao et al., 2025b) and SPG (Wang et al., 2026). Under matched dominant training costs—multi-step diffusion rollouts and optimizer updates—DISPO consistently improves both base optimizers on math reasoning (GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2024)) and symbolic planning (Sudoku, Countdown (Zhao et al., 2025b)). Ablations attribute these gains to same-state branching and newly filled-token updates, confirming that the improvements come from the state-conditioned credit assignment (§ 4.4).

2. Preliminaries

2.1. Terminal-feedback policy optimization for LMs

We view a language model as a conditional policy $\pi_\theta(o \mid q)$ over completions o given a prompt q. In *terminal-feedback* policy optimization, a scalar reward $R(q, o)$ is evaluated on the *final* completion, and learning updates are driven by likelihood-ratio (policy-gradient) objectives at the *sequence level*. This bandit-style view is broadly applicable, but it attributes the same terminal signal to all decoding decisions, yielding coarse credit assignment.

A common instantiation is *group-based* policy optimization (Shao et al., 2024). Given G sampled completions $o_1, \ldots, o_G \sim \pi_{\theta_{\text{old}}}(\cdot \mid q)$ and terminal rewards $r_i = R(q, o_i)$, define the group baseline $\bar{r} = \frac{1}{G}\sum_{i=1}^{G} r_i$ and advantages $A_i = r_i - \bar{r}$. A concise likelihood-ratio objective (optionally with a KL penalty to a reference policy π_{ref}) is

$$\mathbb{E}_q\left[\frac{1}{G}\sum_{i=1}^{G}\underbrace{\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}}_{\rho_i(\theta)} A_i - \beta\,\mathrm{KL}(\pi_\theta(\cdot \mid q) \,\|\, \pi_{\text{ref}}(\cdot \mid q))\right].$$

We use this terminal-feedback formulation as a convenient baseline and as a plug-in component for our method.

2.2. Surrogate likelihood for terminal-feedback optimization of MDLMs

Masked diffusion language models (MDLMs) generate text by iteratively refining a partially-masked sequence, unmasking tokens in an arbitrary order. This makes the exact sequence likelihood $\log \pi_\theta(o \mid q)$ intractable, which prevents directly applying standard terminal-feedback objectives such as § 2.1. To enable *terminal-feedback* policy optimization for MDLMs, Zhao et al. (2025b) introduces a tractable *surrogate likelihood* $\tilde{\pi}_\theta$ (yielding an MDLM instantiation often referred to as diffu-GRPO).

Given a prompt–completion pair (q, o), we sample a random prompt-masking pattern m to obtain a corrupted prompt $q^{(m)}$. We then form a fully masked completion $\tilde{o}$, run a single denoising step to obtain per-position token distributions $p_\theta(\cdot \mid q^{(m)}, \tilde{o})$, and define the surrogate log-probability

$$\log \tilde{\pi}_\theta(o \mid q) \;=\; \mathbb{E}_m\left[\sum_{j=1}^{|o|} \log p_\theta\big(o_j \mid q^{(m)}, \tilde{o}\big)\right]. \quad (1)$$

In terminal-feedback MDLM policy optimization, $\tilde{\pi}_\theta$ replaces π_θ inside likelihood ratios and KL terms, allowing MDLMs to be trained as conditional policies from scalar rewards on final completions.

2.3. Rethinking MDLM decision points for RL

In an episodic MDP, the policy gradient theorem (Williams, 1992; Sutton et al., 1999) states that for a stochastic policy $\pi_\theta(a_t \mid s_t)$,

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right],$$

where s_t and a_t are the state and action, and G_t denotes the return from step t. Each decision (s_t, a_t) contributes an explicit term $\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t$. This decomposition enables fine-grained credit assignment across steps.

Most RL formulations for LMs (autoregressive or diffusion) collapse decoding to a *terminal-reward bandit*, where reward evaluation and gradient attribution are both driven by the final completion (Shao et al., 2024; Zhao et al., 2025b). MDLMs, however, provide an anytime completion interface: at an intermediate masked state, a single forward pass yields logits for *all* masked positions. Hence we can view the masked sequence as the state s_t and its mask fillings as the action a_t; filling deterministically yields a completion to score with the same terminal reward, and cached logits enable multiple same-state counterfactual completions without additional rollout forward passes. This state–action view motivates our diffusion-state policy optimization in § 3.

3. Diffusion-State Policy Optimization

We present ***Diffusion-State Policy Optimization*** (**DiSPO**), a state-wise credit-assignment objective for MDLM policy optimization (Algo. 1). DiSPO branches at selected intermediate masked states using rollout-cached logits, scores the branched completions with the base terminal reward, and updates only newly filled tokens. By reusing intermediate states and masked-position logits from rollout, it adds same-state comparisons without additional multi-step diffusion rollouts and optimizer steps. *Notably*, DiSPO defines only the state-wise objective and treats the terminal-feedback PO objective as a replaceable base optimizer; for clarity, we first describe DiSPO independently of the terminal objective and then instantiate with diffu-GRPO (Zhao et al., 2025b).

Algorithm 1 Diffusion-State Policy Optimization

Require: initial parameters θ, reward func. R, dataset $\mathcal{D}$, trajectory size K, branch size Z, timestep sampler $\omega(t)$, weights $\alpha_{\text{step}}, \alpha_{\text{term}}$, objectives $\mathcal{L}_{\text{term}}(\cdot)$, $\mathcal{L}_{\text{step}}(\cdot)$, learning rate η, batch size B.
repeat
  initialize: $\theta_{\text{old}} \leftarrow \theta$, $\mathcal{L} \leftarrow 0$
  sample prompts: $\{q_b\}_{b=1}^{B} \sim \mathcal{D}$
  for each prompt q_b **do**
    rollout: sample K denoising trajectories with θ_{old}; cache logits at each visited state $s_{k,t} = (q_b, x_{k,t})$; obtain terminal completions $\{o_k\}_{k=1}^{K}$.
    terminal rewards: $\{R_k\}_{k=1}^{K} \leftarrow \{R(q_b, o_k)\}_{k=1}^{K}$
    $\mathcal{L} \leftarrow \mathcal{L} + \alpha_{\text{term}}\, \mathcal{L}_{\text{term}}(\{(q_b, o_k, R_k)\}_{k=1}^{K}; \tilde{\pi}_\theta, \tilde{\pi}_{\theta_{\text{old}}})$
    sample timesteps: $\mathcal{T}_{\text{sub}} \sim \omega(t)$
    select states: $\mathcal{S}_b \leftarrow \{1, \ldots, K\} \times \mathcal{T}_{\text{sub}}$
    ▷ branching reuses cached logits; no additional rollout passes
    for each selected $(k, t) \in \mathcal{S}_b$ **do**
      same-state branching: sample actions $\{a_{k,t,z}\}_{z=1}^{Z}$ from cached logits at $s_{k,t}$
      form completions: $o_{k,t,z} = \text{FILL}(x_{k,t}, a_{k,t,z})$
      step rewards: $\{R_{k,t,z}\}_{z=1}^{Z} \leftarrow \{R(q_b, o_{k,t,z})\}_{z=1}^{Z}$
      pack outcomes: $\mathcal{B}_{k,t} \leftarrow \{(a_{k,t,z}, R_{k,t,z})\}_{z=1}^{Z}$
      step-wise logps: $\{\log \tilde{\pi}_\theta(a_{k,t,z}|s_{k,t})\}_{z=1}^{Z}$, and $\{\log \tilde{\pi}_{\theta_{\text{old}}}(a_{k,t,z}|s_{k,t})\}_{z=1}^{Z}$
      step-wise loss: $\mathcal{L} \leftarrow \mathcal{L} + \alpha_{\text{step}}\, \mathcal{L}_{\text{step}}(s_{k,t}, \mathcal{B}_{k,t}; \tilde{\pi}_\theta, \tilde{\pi}_{\theta_{\text{old}}})$
    end for
  end for
  update: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$ ▷ matched number of update steps
until convergence

3.1. RL Formulation

States. For a prompt $q \sim \mathcal{D}$ and behavior parameters θ_{old}, a multi-step MDLM sampler produces a trajectory of partially-masked sequences $x_{k,1}, x_{k,2}, \ldots, x_{k,T}$, where T is the number of denoising steps and k indexes trajectories. Each $x_{k,t} \in (\mathcal{V} \cup \{\texttt{[MASK]}\})^L$ is a length-L sequence with a subset of positions masked. Let

$$M_{k,t} = \{i : x_{k,t,i} = \texttt{[MASK]}\}, \quad U_{k,t} = [L] \setminus M_{k,t},$$

and define the diffusion state at step t as $s_{k,t} = (q, x_{k,t})$.

Actions and policy factorization. At state $s_{k,t}$, the MDLM outputs logits for all $i \in M_{k,t}$, inducing per-position token distributions $\pi_\theta(\cdot \mid s_{k,t}, i)$. We define the action as the joint filling of the currently masked positions, $a_{k,t} = \{a_{k,t,i}\}_{i \in M_{k,t}}$, with the factorized policy

$$\pi_\theta(a_{k,t} \mid s_{k,t}) = \prod_{i \in M_{k,t}} \pi_\theta\big(a_{k,t,i} \mid s_{k,t}, i\big). \tag{2}$$

Applying an action yields a completion deterministically: $o = \text{FILL}(x_{k,t}, a_{k,t})$, which replaces $x_{k,t,i}$ by $a_{k,t,i}$ for $i \in M_{k,t}$ and keeps $U_{k,t}$ fixed. This state–action view makes it natural to evaluate intermediate decisions using the same terminal reward $R(q, o)$. Moreover, DiSPO attributes gradients only to the action distribution $\pi_\theta(a_{k,t} \mid s_{k,t})$, i.e., the newly filled tokens on $M_{k,t}$, treating previously filled tokens as part of the state. In practice, likelihood ratios and KL terms are computed with a tractable masked-token surrogate introduced in § 3.3.

3.2. Per-state Branching and Rewards

For each prompt $q \sim \mathcal{D}$, we first sample K behavior trajectories by running the multi-step MDLM sampler with frozen parameters θ_{old}, obtaining $\{x_{k,1:T}\}_{k=1}^{K}$ and terminal completions $o_k = x_{k,T}$. During rollout, we cache the logits for each masked position $i \in M_{k,t}$ at every visited state $s_{k,t}$.

We then select a set of state indices $\mathcal{S} \subseteq \{(k, t)\}$ to update. For each selected $(k, t) \in \mathcal{S}$, we form a *state-conditioned* group by branching from the state: using cached logits at $s_{k,t}$, we resample Z actions

$$a_{k,t,1}, \ldots, a_{k,t,Z} \sim \pi_{\theta_{\text{old}}}(\cdot \mid s_{k,t}) = \prod_{i \in M_{k,t}} \pi_{\theta_{\text{old}}}(\cdot \mid s_{k,t}, i),$$

and obtain completions $o_{k,t,z} = \text{FILL}(x_{k,t}, a_{k,t,z})$. Since branching reuses rollout-cached logits, generating these Z completions incurs no additional *rollout* forward passes.

All completions share the same terminal reward function,

$$R_{k,t,z} = R(q, o_{k,t,z}),$$

yielding per-state baselines and group-relative advantages:

$$\bar{R}_{k,t} = \frac{1}{Z}\sum_{z=1}^{Z} R_{k,t,z}, \quad A_{k,t,z} = R_{k,t,z} - \bar{R}_{k,t}. \tag{3}$$

These advantages capture state-conditioned preferences over alternative fillings under a fixed intermediate state.

3.3. Step-level Surrogate Log-probabilities

To optimize masked-token actions at intermediate states, we need tractable log-probabilities to form likelihood ratios for the diffusion-state policy. Following Zhao et al. (2025b), we use a one-step masked-token surrogate $\tilde{\pi}_\theta$ (Eq. 1), and adapt it to a *state-wise definition* that accounts only for the currently masked positions.

State-wise masked-token surrogate. For each state $s_{k,t} = (q, x_{k,t})$, sample a random prompt-masking pattern m to obtain a corrupted prompt $q^{(m)}$ and define $s_{k,t}^{(m)} = (q^{(m)}, x_{k,t})$. Running a single denoising step on $s_{k,t}^{(m)}$ yields per-position distributions $\tilde{\pi}_\theta(\cdot \mid s_{k,t}^{(m)}, i)$ for all $i \in M_{k,t}$. For an action sample $a_{k,t,z}$, define the state-wise surrogate log-probability by summing over actionable positions:

$$\log \tilde{\pi}_\theta(a_{k,t,z} \mid s_{k,t}) := \mathbb{E}_m\left[\sum_{i \in M_{k,t}} \log \tilde{\pi}_\theta\big(a_{k,t,z,i} \mid s_{k,t}^{(m)}, i\big)\right]. \tag{4}$$

This makes the update token-local: completion rewards affect only the newly filled tokens.

Implementation note. We estimate the expectation over m with the same Monte Carlo scheme as Zhao et al. (2025b). All log-probabilities, likelihood ratios, and KL terms in our objectives are computed using $\tilde{\pi}_\theta$.

3.4. Training Objective

For clarity, we present the *unclipped* likelihood-ratio objectives; our implementation uses standard clipping and a KL penalty to a reference surrogate policy $\tilde{\pi}_{\text{ref}}$.

State-wise DiSPO objective. For each selected state $(k,t) \in \mathcal{S}$, we have Z branched actions with rewards $\{(a_{k,t,z}, R_{k,t,z})\}_{z=1}^{Z}$ and group-relative advantages from Eq. 3. Using the state-wise surrogate in Eq. 4, define

$$\rho_{k,t,z}(\theta) = \exp\Big(\log \tilde{\pi}_\theta(a_{k,t,z} \mid s_{k,t}) - \log \tilde{\pi}_{\theta_{\text{old}}}(a_{k,t,z} \mid s_{k,t})\Big).$$

The state-wise loss by DiSPO for state (k,t) is

$$\mathcal{L}_{\text{step}}^{(k,t)}(\theta) = -\frac{1}{Z}\sum_{z=1}^{Z} \rho_{k,t,z}(\theta)\, A_{k,t,z}, \tag{5}$$

and the aggregated state-wise loss is

$$\mathcal{L}_{\text{step}}(\theta) = \sum_{(k,t)\in\mathcal{S}} \mathcal{L}_{\text{step}}^{(k,t)}(\theta). \tag{6}$$

Combination with a base terminal objective. DiSPO is agnostic to the choice of terminal-feedback optimizer. Let $\mathcal{L}_{\text{base}}(\theta)$ denote a terminal-feedback MDLM PO objective computed from the same rollout completions and terminal reward function. We optimize

$$\mathcal{L}(\theta) = \alpha_{\text{base}}\, \mathcal{L}_{\text{base}}(\theta) + \alpha_{\text{step}}\, \mathcal{L}_{\text{step}}(\theta), \tag{7}$$

where $\alpha_{\text{base}}, \alpha_{\text{step}} \in \mathbb{R}_{\geq 0}$. This separates the proposed state-wise objective from the replaceable terminal-feedback base optimizer. We set $\alpha_{\text{base}} = 1$ in experiments.

diffu-GRPO instantiation. For controlled comparisons with prior MDLM policy optimization, we instantiate $\mathcal{L}_{\text{base}}$ with diffu-GRPO (Zhao et al., 2025b). For each prompt q, rollout completions $\{o_k\}_{k=1}^{K}$ are scored by $r_k = R(q, o_k)$ and converted to group-relative advantages $A_k = r_k - \bar{r}_q$ where $\bar{r}_q = \frac{1}{K}\sum_{k=1}^{K} r_k$. Using the sequence-level surrogate log-probabilities $\log \tilde{\pi}_\theta(q, o_k)$ from Eq. 1, define

$$\rho_k(\theta) = \exp\Big(\log \tilde{\pi}_\theta(q, o_k) - \log \tilde{\pi}_{\theta_{\text{old}}}(q, o_k)\Big).$$

The diffu-GRPO base loss is

$$\mathcal{L}_{\text{base}}(\theta) = -\frac{1}{K}\sum_{k=1}^{K} \rho_k(\theta)\, A_k. \tag{8}$$

Other terminal-feedback objectives, such as SPG, can be used by replacing $\mathcal{L}_{\text{base}}$ in Eq. 7.

3.5. Theoretical Analysis

We summarize the theoretical properties of DiSPO (Proofs in Appendix A). We here consider the unclipped likelihood-ratio objective without the KL penalty for simplicity.

Fixed-state policy gradient. First, DiSPO provides a valid policy-gradient signal for fixed-state mask filling. For a fixed denoising step, conditioning on intermediate masked states induced by the frozen behavior policy, the expected gradient of the step-wise loss is aligned with the policy gradient of a fixed-state expected-return objective, up to the constant factor $c_Z = (Z-1)/Z$ induced by the within-state group baseline. See Theorem A.1 in Appendix A.1.

Mixed objective. Second, the step-wise objective can be combined with a terminal-feedback base optimizer without introducing an ad-hoc learning signal. In expectation, the combined loss optimizes a single mixed objective consisting of the base terminal objective and the timestep-averaged fixed-state objectives. See Theorem A.2 in Appendix A.2.

Variance-motivated design choices. Finally, we provide simplified variance analyses that motivate the two main design choices in DiSPO: updating only newly filled tokens

Table 1. **DiSPO on LLaDA.** Exact-match accuracy (%) on planning and math reasoning, evaluated with $N_{\text{gen}} \in \{128, 256, 512\}$. Matched-compute comparisons are made within each base optimizer: diffu-GRPO vs. DiSPO$_{\text{diffu-GRPO}}$ and SPG vs. DiSPO$_{\text{SPG}}$, with the same training N_{gen}=256, multi-step diffusion rollout budget, and optimizer steps. † indicates a non-matched-compute reward-shaping baseline included for reference. ‡ indicates results reported by Zhao et al. (2025b).

	Sudoku				Countdown				GSM8K				MATH500			
Methods \ N_{gen}	**128**	**256**	**512**	**Avg.**	**128**	**256**	**512**	**Avg.**	**128**	**256**	**512**	**Avg.**	**128**	**256**	**512**	**Avg.**
LLaDA-8B-Inst. (Nie et al., 2025)‡	11.7	6.7	5.5	8.0	20.7	19.5	16.0	18.7	68.7	76.7	78.2	74.5	26.0	32.4	36.2	31.5
diffu-GRPO (Zhao et al., 2025b)‡	18.4	12.9	11.0	14.1	33.2	31.3	37.1	33.9	72.6	79.8	81.9	78.1	33.2	37.2	39.2	36.5
DiSPO$_{\text{diffu-GRPO}}$	**34.7**	**30.2**	**29.4**	**31.4**	**49.6**	**49.6**	**62.5**	**53.9**	**74.8**	**81.9**	**82.9**	**79.9**	**34.0**	**39.2**	**40.2**	**37.8**
SPG (Wang et al., 2026)	26.6	26.0	26.1	26.2	67.6	66.8	67.2	67.2	77.3	80.9	81.5	79.9	33.6	38.4	40.2	37.4
DiSPO$_{\text{SPG}}$	**43.2**	**44.0**	**43.4**	**43.5**	**68.0**	**68.8**	**68.8**	**68.5**	**79.0**	**81.7**	**81.8**	**80.8**	**35.2**	**39.4**	**41.0**	**38.5**
SAPO (Xie et al., 2025)†	22.4	20.3	16.1	19.6	51.6	52.0	56.3	53.3	72.9	82.2	82.4	79.2	32.0	40.0	38.4	36.8

Table 2. **DiSPO on LLaDA-SFT.** Under the same evaluation and matched-compute setup as Table 1, DiSPO uses a conservative step weight (α_{step}=0.1) and still improves SFT + diffu-GRPO across benchmarks. ‡ indicates results reported by Zhao et al. (2025b).

	Sudoku				Countdown				GSM8K				MATH500			
Methods \ N_{gen}	**128**	**256**	**512**	**Avg.**	**128**	**256**	**512**	**Avg.**	**128**	**256**	**512**	**Avg.**	**128**	**256**	**512**	**Avg.**
LLaDA-8B-Inst. (Nie et al., 2025)‡	11.7	6.7	5.5	8.0	20.7	19.5	16.0	18.7	68.7	76.7	78.2	74.5	26.0	32.4	36.2	31.5
SFT (Muennighoff et al., 2025)‡	16.5	8.5	4.6	9.9	20.3	14.5	23.8	19.5	66.5	78.8	81.1	75.5	26.2	32.6	34.8	31.2
SFT & diffu-GRPO (Zhao et al., 2025b)‡	22.1	16.7	9.5	16.1	34.8	32.0	**42.2**	36.3	73.2	81.1	82.1	78.8	33.8	38.6	**40.2**	37.5
SFT & **DiSPO**$_{\text{diffu-GRPO}}$	**26.4**	**25.0**	**11.7**	**21.0**	**37.5**	**32.8**	40.2	**36.8**	**75.0**	**82.3**	**83.2**	**80.2**	**34.8**	**39.2**	**40.2**	**38.1**

and averaging over same-state branches. Under standard independence assumptions, token-local updates reduce variance by excluding non-actionable positions, while averaging Z same-state branches yields the usual $O(1/Z)$ variance reduction. See Propositions A.3 and A.4 in Appendix A.3.

4. Experiments

This section evaluates whether state-conditioned credit assignment improves terminal-feedback PO for MDLMs. We plug DiSPO into two base optimizers, diffu-GRPO (Zhao et al., 2025b) and SPG (Wang et al., 2026), by adding the state-wise objective in Algorithm 1. For controlled comparisons, we match multi-step diffusion rollouts, sequence-level group size, and RL update steps. DiSPO forms branches only from logits cached in those rollouts, so gains reflect the added state-wise signal rather than extra rollout compute.

4.1. Setup

Models. We evaluate LLaDA-8B-Instruct (Nie et al., 2025) and its s1k-supervised fine-tuned variant (Muennighoff et al., 2025). We follow Zhao et al. (2025b) for the model setup and denoising schedule. See Appendix C.3 for details.

Benchmarks. Following Zhao et al. (2025b), we report exact-match accuracy on math reasoning benchmarks (GSM8K (Cobbe et al., 2021); MATH500 (Lightman et al., 2024)) and symbolic planning benchmarks (Sudoku; Countdown (Zhao et al., 2025b)). See Appendix C.1 for details.

Optimization. For each base optimizer, we match the dominant training budgets between the baseline and its DiSPO-augmented variant, including N_{gen}=256, the denoising schedule (with block size 32), the rollout budget, and the number of RL update steps. For SPG, we use the original paper's practical masking and mixed negative-advantage estimator settings, fixed for both SPG and DiSPO$_{\text{SPG}}$. We use the same terminal reward evaluator for both terminal and state-wise terms, without reward shaping. DiSPO uses Z=2 and samples one timestep per rollout from a late-biased polynomial distribution with degree k=4. We sweep $\alpha_{\text{step}} \in \{0.1, 0.5, 0.9\}$. See Appendix C.3 for details.

Evaluation. We follow the protocol of diffu-GRPO (Zhao et al., 2025b); we evaluate at $N_{\text{gen}} \in \{128, 256, 512\}$ using zero-shot greedy decoding. For diffusion decoding, we use $N_{\text{gen}}/2$ denoising steps with a block size of 32 tokens. Additional details are provided in Appendix C.2.

4.2. Main results

Table 1 shows that DiSPO improves both base optimizers, diffu-GRPO and SPG, on LLaDA-8B-Instruct across benchmarks. The gains are largest on planning tasks and generally persist across evaluation budgets $N_{\text{gen}} \in \{128, 256, 512\}$. Table 2 further shows that the same state-wise objective also improves after SFT, even with the conservative α_{step}=0.1.

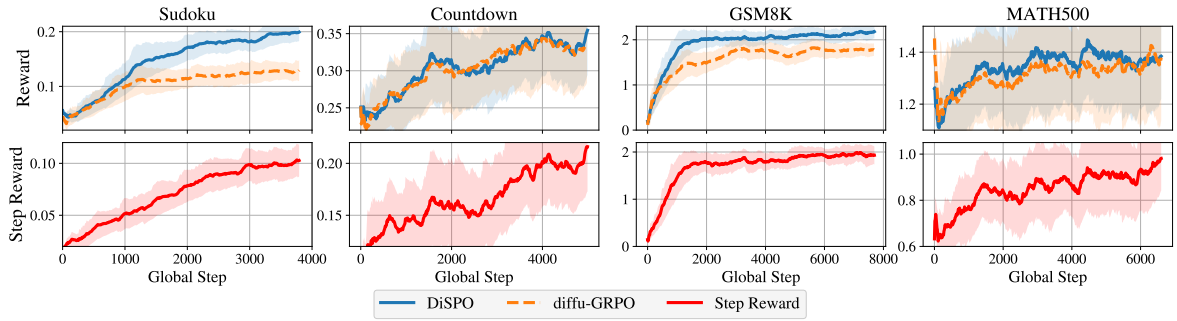


Figure 2. **Reward curves.** Terminal reward curves (*top*) and step reward curves (*bottom*) during policy optimization with DiSPO instantiated by diffu-GRPO. Across tasks, DiSPO reaches higher terminal rewards earlier and maintains them over training. Step rewards exhibit relatively smaller magnitudes but follow trends as terminal rewards, indicating their role as a complementary training signal.

Table 3. **Ablation on step-wise estimator design.** Gradient scope (Prop. A.3) and number of same-state drafts Z (Prop. A.4). All values represent the exact-match accuracy (%) on Sudoku.

Perspectives \ N_{gen}	128	256	512	Avg.
Grad. ▷ All tokens	22.7	21.4	**20.6**	21.6
Grad. ▷ Action only	**26.6**	**23.3**	20.0	**23.3**
# Drafts ▷ $Z = 2$	**26.6**	**23.3**	20.0	**23.3**
# Drafts ▷ $Z = 3$	25.1	22.8	**21.0**	23.0
# Drafts ▷ $Z = 4$	**26.6**	22.1	19.6	22.8

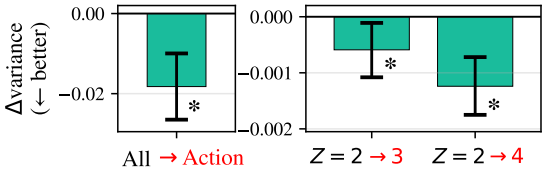


Figure 3. **Variance reduction of the step-wise gradients on Sudoku.** ***Left***: Updating only actions (vs. all tokens) reduces variance at Z=2. ***Right***: Increasing Z reduces variance. Error bars: paired 95% bootstrap CIs.

All matched comparisons use the same multi-step diffusion rollout budget, sequence-level group size, optimizer steps, and terminal reward evaluator. Thus, the improvements are best explained by finer state-conditioned credit assignment rather than additional rollout compute or reward shaping.

To probe the underlying training dynamics, Figure 2 plots both terminal rewards (*top*) and step rewards (*bottom*) on LLaDA-8B-Instruct. DiSPO reaches higher terminal rewards earlier and sustains them throughout training; while step rewards vary over a smaller range, they track the up and downs of terminal reward, consistent with a state-wise signal that aligns with terminal feedback along the trajectory.

4.3. Analysis: State-wise Estimator Design

Propositions A.3 and A.4 motivate two design choices for the state-wise estimator: i) updating only actionable tokens and ii) averaging over Z same-state branches. We test both on 4×4 Sudoku through controlled gradient-variance measurements and task ablations.

Table 4. **Ablation on optimization recipe.** Timestep sampling, loss components, and step-loss intensity α_{step}. All values represent the exact-match accuracy (%) on Sudoku.

Perspectives \ N_{gen}	128	256	512	Avg.
Timestep ▷ Uniform	25.1	23.0	17.5	21.9
Timestep ▷ Early-focused	24.2	18.1	16.0	19.4
Timestep ▷ Late-focused	**26.6**	**23.3**	**20.0**	**23.3**
Loss ▷ $\mathcal{L}_{\text{base}}$ only	18.4	12.9	11.0	14.1
Loss ▷ $\mathcal{L}_{\text{step}}$ only	24.9	18.4	17.7	20.3
Loss ▷ Both	**26.6**	**23.3**	**20.0**	**23.3**
Intensity ▷ $\alpha_{\text{step}} = 0.0$	18.4	12.9	11.0	14.1
Intensity ▷ $\alpha_{\text{step}} = 0.1$	26.6	23.3	20.0	23.3
Intensity ▷ $\alpha_{\text{step}} = 0.5$	**34.7**	**30.2**	**29.4**	**31.4**
Intensity ▷ $\alpha_{\text{step}} = 0.9$	28.8	27.8	26.3	27.6

Gradient variance. Using fixed intermediate states from the final LLaDA-8B-Instruct Sudoku checkpoint, we resample branches and estimate the trace covariance of the state-wise gradient, $\mathrm{trCov}(\hat{g})$. Figure 3 confirms both predictions: i) action-only updates reduce variance relative to all-token updates, and ii) larger Z reduces variance under action-only updates.

Impact on Task performance. Table 3 shows that all-token updates underperform action-only updates, supporting the state–action decomposition. Increasing Z from 2 to 4 sometimes improves individual evaluation budgets but gives no monotonic average gain, so we use Z=2 by default.

4.4. Ablations: Optimization Choices

Timestep sampling. Table 4 (*top*) compares uniform sampling with late- and early-biased polynomial samplers (k=4). Late-biased sampling performs best, suggesting that states closer to the final completion provide more reliable credit-assignment signals than very early masked states.

Objective combination. Table 4 (*middle*) shows that $\mathcal{L}_{\text{step}}$ alone outperforms $\mathcal{L}_{\text{base}}$ alone, while their combination performs best, indicating thier complementary nature.

Step-wise weight. Table 4 (*bottom*) shows that all tested $\alpha_{\text{step}} \in \{0.1, 0.5, 0.9\}$ improve over α_{step}=0, with the best result at α_{step}=0.5. Thus, the state-wise objective is consistently useful, but its weight should be balanced against the terminal objective.

4.5. Diagnostics of Intermediate-State Branching

Training rollout block size. During optimization, the base optimizer generates samples using block-wise diffusion rollouts; our main setup uses block size = 32. Because this choice affects which intermediate states and logits are exposed to DiSPO, we vary the training rollout block size in Table 5. DiSPO remains effective across block sizes, indicating robustness to the rollout schedule used during policy optimization.

Reward variation across branches. DiSPO relies on reward differences among alternative fillings from the same intermediate state. As a *conservative* diagnostic, we *uniformly* sample states, although our main experiments use a late-biased polynomial sampler. Table 6 shows that, even under this uniform sampling, 17.8% of states yield nonzero reward variation across branches, with the fraction increasing sharply later in denoising. This supports our premise that intermediate rollout states contain useful credit-assignment signals that terminal-only training leaves unexploited. The increase at later denoising steps is also consistent with our late-biased timestep sampler.

Cached-branch stability. Since branches are sampled from rollout-cached logits, they could in principle become stale as the policy changes. However, cached logits are used only to generate branch candidates; optimization still uses the usual current-vs.-old surrogate likelihood ratio, $\rho = \tilde{\pi}_\theta / \tilde{\pi}_{\theta_{\text{old}}}$. Replay diagnostics show that the step-wise log-ratio $\log \rho$ remains comparable to the terminal log-ratio, with very low clip fractions (Table 9 in Appendix G). This suggests that cached-branch reuse does not introduce observable optimization instability.

Table 5. **Block size of rollout trajectories at the *training* on Sudoku.** Improvements have been observed in different rollout sizes.

Block size / N_{gen}	128	256	512	Avg.
(no training)	11.7	6.7	5.5	8.0
8	29.5	26.9	26.3	27.6
32 (default)	34.7	30.2	29.4	31.4
128	32.4	30.8	32.4	31.9

Table 6. **Fraction of reward-distinguishable intermediate states**; 64 prompts, 6 generations per prompt, 8 states per trajectory, 128 sampling steps. Timestep bins are defined over normalized denoising progress.

Timestep bin	Reward-distinguishable states (%)
early $[0.0, 0.3)$	4.5
mid $[0.3, 0.7)$	9.2
late $[0.7, 1.0]$	43.8
overall	17.8

Table 7. **Additional operation counts per prompt with DiSPO.** Z: # branches/state, $\mathcal{S}$: selected states, N_m: # monte-carlo prompt masks for $\mathbb{E}_m[\cdot]$ in the surrogate.

Compute items	Δ w/ DiSPO
Diffusion rollout forward	0
Optimizer update steps	0
One-step surrogate (terminal)	0
Reward evaluations	$\lvert\mathcal{S}\rvert Z$
One-step surrogate (step-wise)	$2N_m\lvert\mathcal{S}\rvert$

4.6. Compute Cost Breakdown

Algorithmic overhead. Table 7 summarizes the additional operations introduced by DiSPO. The dominant training costs—multi-step diffusion rollouts and optimizer update steps—are unchanged. DiSPO adds only lightweight extras: reward evaluations for same-state branches and *one-step* surrogate calls for selected states.

Implementation-level wall-clock overhead. Although DiSPO leaves the dominant algorithmic costs unchanged, our current implementation is, in practical, slower in wall-clock time (∼0.4× baseline throughput at Z=2 and $|\mathcal{S}|$=K) due to non-optimized repeated policy/surrogate calls and kernel/IO. This overhead is implementation-dependent rather than intrinsic to the rollout/update budget; likely optimizations are discussed in Appendix F. We therefore evaluate wall-clock-matched training in Fig. 4. Even under this stricter comparison, DiSPO catches up and surpasses the terminal-feedback baseline within the same time budget, suggesting that improved credit assignment offsets the current implementation overhead.

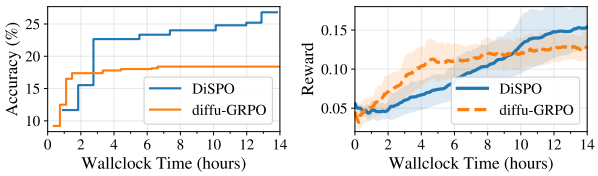
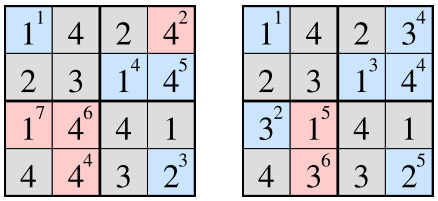


Figure 4. **Wall-clock-matched training curves on LLaDA-8B-Instruct for Sudoku.** Accuracy (N_{gen}=128) and reward vs. training time. DiSPO surpasses diffu-GRPO within the budget.



(a) diffu-GRPO *(b)* DiSPO

Figure 5. **Qualitative example of premature commitment on Sudoku.** Gray cells are inputs; red and blue cells denote incorrect and correct fillings, respectively, with superscripts indicating fill order. diffu-GRPO makes an early constraint-violating fill, while DiSPO keeps the partial assignment consistent.

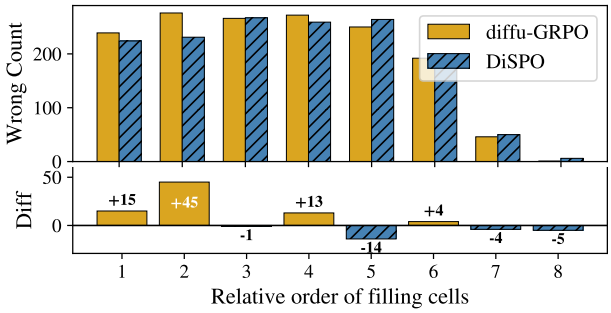


Figure 6. **Premature commitments on Sudoku.** The first-violation time is the earliest denoising step at which the filled cells violate Sudoku constraints. DiSPO shifts violations to later steps than diffu-GRPO, indicating fewer premature commitments.

4.7. Error Analysis: Premature Commitments

We analyze premature commitment errors on Sudoku using the final LLaDA-8B-Instruct checkpoint with $N_{\text{gen}}=128$. Figure 5 shows a representative problem instance at the same denoising step index for both methods: diffu-GRPO has already made a constraint-violating fill, whereas DiSPO maintains a consistent partial assignment.

We define the *first-violation time* as the earliest denoising step at which a filled cell violates a Sudoku constraint. Figure 6 shows that DiSPO shifts first violations later in denoising, suggesting fewer premature, hard-to-correct commitments.

5. Related Work

Terminal-feedback policy optimization for MDLMs. Most RL fine-tuning methods for masked diffusion LMs optimize a scalar terminal reward on the final completion, using PPO/GRPO-style objectives with diffusion surrogate likelihoods (Zhao et al., 2025b; Wang et al., 2026; Tang et al., 2026; Yang et al., 2025; Rojas et al., 2025; Zekri & Boulle, 2026; Zhao et al., 2025a). DiSPO is complementary: it compares alternative mask fillings at fixed intermediate states via a state-wise objective.

Terminal reward shaping. SAPO uses denoising states to shape terminal rewards with step-aware bonuses (Xie et al., 2025). This is complementary to DiSPO: SAPO designs denser rewards along realized denoising processes, whereas DiSPO optimizes filling actions under a fixed masked state. We include SAPO as a non-matched-compute reference in Table 1, with compute details in Appendix E.

Trajectory-level objectives. MDPO aligns the denoising process by comparing different denoising trajectories across steps and favoring trajectories whose intermediate predictions improve toward the terminal outcome (He et al., 2025). In contrast, DiSPO has the different optimization unit: it fixes an intermediate masked state and compares alternative fillings of the current masks.

Surrogate likelihoods and gradient estimators. Complementary work improves surrogate likelihoods or policy-gradient estimators for diffusion LMs (Wang et al., 2026; Zekri & Boulle, 2026; Rojas et al., 2025; Wang et al., 2025). DiSPO can directly incorporate improved surrogates/estimators while keeping the state-wise objective unchanged.

6. Conclusion

We proposed DiSPO, a plug-in state-wise objective for policy optimization in masked diffusion language models. By fixing intermediate masked states and comparing alternative mask fillings sampled from rollout-cached logits, DiSPO turns intermediate rollout information into fine-grained credit-assignment signals without additional multi-step diffusion rollouts. The resulting objective admits a policy-gradient interpretation and can be combined with terminal-feedback base optimizers. On LLaDA-8B-Instruct, DiSPO improves both diffu-GRPO and SPG across math and planning benchmarks under matched rollout and update budgets, showing that intermediate-state action optimization is an effective complement to terminal-feedback training.

Limitations and discussion. Future work includes extending DiSPO beyond the diffu-GRPO and SPG instantiations studied here, for example to other terminal-feedback objectives such as GDPO (Rojas et al., 2025) and to stronger surrogate likelihood or gradient estimators. We also use the same unshaped terminal reward for both terminal and state-wise updates, leaving open how fixed-state action optimization interacts with reward-shaping methods such as SAPO (Xie et al., 2025). Although DiSPO adds no multi-step diffusion rollouts or optimizer steps, our current implementation incurs wall-clock overhead from repeated reward and surrogate evaluations, suggesting room for systems-level optimization (Appendix F). Finally, while DiSPO may improve MDLM reasoning and planning, stronger LMs can also increase misuse risks; our experiments are limited to public benchmarks with no deployment or user data.

Acknowledgement

This work was partially supported by JSPS KAKENHI Grant Number 25H01137 and JST K Program Japan Grant Number JPMJKP24C3.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning by improving policy optimization for masked diffusion language models. There are many potential societal consequences of improved language-model reasoning capabilities, including both beneficial applications and possible misuse; we do not believe our method introduces unique ethical concerns beyond those common to this line of research.

References

Arriola, M., Sahoo, S. S., Gokaslan, A., Yang, Z., Qi, Z., Han, J., Chiu, J. T., and Kuleshov, V. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x.

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

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gong, S., Zhang, R., Zheng, H., Gu, J., Jaitly, N., Kong, L., and Zhang, Y. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.

He, H., Renz, K., Cao, Y., and Geiger, A. Mdpo: Overcoming the training-inference divide of masked diffusion language models. *arXiv preprint arXiv:2508.13148*, 2025.

Israel, D., Broeck, G. V. d., and Grover, A. Accelerating diffusion llms via adaptive parallel decoding. *arXiv preprint arXiv:2506.00413*, 2025.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Liu, Z., Yang, Y., Zhang, Y., Chen, J., Zou, C., Wei, Q., Wang, S., and Zhang, L. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Luxembourg, O., Permuter, H., and Nachmani, E. Plan for speed–dilated scheduling for masked diffusion language models. *arXiv preprint arXiv:2506.19037*, 2025.

Ma, X., Yu, R., Fang, G., and Wang, X. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. s1: Simple test-time scaling. In Christodoulopoulos, C., Chakraborty, T., Rose, C., and Peng, V. (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Nie, S., Zhu, F., You, Z., Zhang, X., Ou, J., Hu, J., Zhou, J., Lin, Y., Wen, J.-R., and Li, C. Large language diffusion models, 2025. URL https://arxiv.org/abs/2502.09992.

Oba, D., Bollegala, D., Kaneko, M., and Okazaki, N. Stopping computation for converged tokens in masked diffusion-LM decoding. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=PzhNnMepgl.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Rojas, K., Lin, J., Rasul, K., Schneider, A., Nevmyvaka, Y., Tao, M., and Deng, W. Improving reasoning for diffusion language models via group diffusion policy optimization. *arXiv preprint arXiv:2510.08554*, 2025.

Sahoo, S., Arriola, M., Schiff, Y., Gokaslan, A., Marroquin, E., Chiu, J., Rush, A., and Kuleshov, V. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Shi, J., Han, K., Wang, Z., Doucet, A., and Titsias, M. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Tang, X., Dolga, R., Yoon, S., and Bogunovic, I. wd1: Weighted policy optimization for reasoning in diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=L2rfd2Czbj.

Wang, C., Rashidinejad, P., Su, D., Jiang, S., Wang, S., Zhao, S., Zhou, C., Shen, S. Z., Chen, F., Jaakkola, T., Tian, Y., and Liu, B. SPG: Sandwiched policy gradient for masked diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=18j5Q49GwN.

Wang, G., Turok, G., Schiff, Y., Arriola, M., and Kuleshov, V. d2: Improved techniques for training reasoning diffusion language models. *arXiv preprint arXiv:2509.21474*, 2025.

Wei, Q., Zhang, Y., Liu, Z., Liu, D., and Zhang, L. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv preprint arXiv:2506.10848*, 2025.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Wu, C., Zhang, H., Xue, S., Liu, Z., Diao, S., Zhu, L., Luo, P., Han, S., and Xie, E. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.

Xie, S., Kong, L., Song, X., Dong, X., Chen, G., Xing, E. P., and Zhang, K. Step-aware policy optimization for reasoning in diffusion large language models. *arXiv preprint arXiv:2510.01544*, 2025.

Yang, J., Chen, G., Hu, X., and Shao, J. Taming masked diffusion language models via consistency trajectory reinforcement learning with fewer decoding step. *arXiv preprint arXiv:2509.23924*, 2025.

Ye, J., Xie, Z., Zheng, L., Gao, J., Wu, Z., Jiang, X., Li, Z., and Kong, L. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Zekri, O. and Boulle, N. Fine-tuning discrete diffusion models with policy gradient methods. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=rXFzVRZsbt.

Zhao, H., Liang, D., Tang, W., Yao, D., and Kallus, N. DiFFPO: Training diffusion LLMs to reason fast and furious via reinforcement learning. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025a. URL https://openreview.net/forum?id=zmDXUAmjRn.

Zhao, S., Gupta, D., Zheng, Q., and Grover, A. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025b.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., and Irving, G. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Theoretical Analysis

DISPO treats intermediate diffusion states as decision points and directly optimizes their mask fillings via policy gradients. Under the masked-token surrogate policy $\tilde{\pi}_\theta$ (Eq. 4), we show that the step-wise loss provides a *principled* policy-gradient estimator for a fixed-state objective (Theorem A.1). We further show that combining step-wise and terminal losses corresponds in expectation to optimizing a single objective (Theorem A.2). We also provide simple variance-reduction mechanisms that justify token-local updates and same-state averaging (§ A.3). For clarity, we analyze the *unclipped* likelihood-ratio objective without KL.

A.1. Step-wise Objective and Step-level Policy Gradient

Fix a denoising step index t. Let $d_t^{\text{old}}(\cdot \mid q)$ denote the distribution over intermediate states at step t induced by rolling out the frozen behavior model θ_{old} on prompt q. Using the state–action view in § 3.1, define

$$J_{\text{step},t}(\theta) := \mathbb{E}_{q\sim\mathcal{D}}\mathbb{E}_{s_t\sim d_t^{\text{old}}(\cdot|q)}\mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_t)}\big[R\big(q, \textsc{Fill}(x_t, a)\big)\big], \tag{9}$$

where $s_t = (q, x_t)$ and $\textsc{Fill}(x_t, a)$ deterministically completes x_t by filling its masked positions with a. Let $\mathcal{L}_{\text{step}}^{(t)}(\theta)$ denote the corresponding unclipped step-wise loss at timestep t, computed as in Eq. 5 with Z same-state branches.

Theorem A.1 (Step-level policy gradient)**.** *Assume (i) $d_t^{\text{old}}(\cdot \mid q)$ is independent of θ, and (ii) $\tilde{\pi}_\theta(\cdot \mid s)$ is differentiable and normalized for each s. Using the within-group mean baseline in Eq. 3, in the unclipped likelihood-ratio setting the expected gradient of the step-wise loss is aligned with the policy gradient of $J_{\text{step},t}(\theta)$:*

$$\mathbb{E}\big[-\nabla_\theta \mathcal{L}_{step}^{(t)}(\theta)\big] = \frac{Z-1}{Z}\,\nabla_\theta J_{\text{step},t}(\theta). \tag{10}$$

Proof idea. Write the step-wise loss (Eq. 5) as a score-function estimator for $\tilde{\pi}_\theta(a \mid s_t)$ using action-only log-probabilities on the current mask. The likelihood-ratio form gives the standard importance-weighted policy gradient; the within-group mean baseline induces only the constant factor $(Z-1)/Z$ in expectation and reduces variance. Full details are in Appendix B.1.

A.2. Mixed Objective and Overall Policy Gradient

A key property of DISPO is that adding step-wise updates does not introduce an ad-hoc signal: in the *unclipped, exact likelihood-ratio* setting, the expected gradient of the combined loss (Eq. 7) equals the gradient of a single well-defined objective.

Terminal surrogate objective. Define the terminal (sequence-level) surrogate objective

$$J_{\text{seq}}(\theta) := \mathbb{E}_{q\sim\mathcal{D},\, o\sim\tilde{\pi}_\theta(\cdot|q)}\big[R(q, o)\big]. \tag{11}$$

A standard likelihood-ratio argument implies that the unclipped terminal loss $\mathcal{L}_{\text{base}}(\theta)$ is an unbiased policy-gradient estimator for $J_{\text{seq}}(\theta)$ (Shao et al., 2024).

Mixed objective. Let $c_Z := (Z-1)/Z$ denote the constant scaling induced by the within-group mean baseline in Eq. 3. With timestep sampling $\omega(t)$ and the step-wise objectives $\{J_{\text{step},t}(\theta)\}$ (Eq. 9), define

$$J_{\text{mix}}(\theta) := \alpha_{\text{step}}\, c_Z \sum_t \omega(t)\, J_{\text{step},t}(\theta) + \alpha_{\text{base}}\, J_{\text{seq}}(\theta). \tag{12}$$

Theorem A.2 (Overall policy gradient)**.** *Under the assumptions of Theorem A.1 (with the within-group mean baseline) and the standard likelihood-ratio assumptions for $J_{seq}(\theta)$,*

$$\mathbb{E}\big[-\nabla_\theta \mathcal{L}(\theta)\big] = \nabla_\theta J_{mix}(\theta), \tag{13}$$

where $\mathcal{L}(\theta) = \alpha_{step}\mathcal{L}_{step}(\theta) + \alpha_{\text{base}}\mathcal{L}_{\text{base}}(\theta)$.

Proof idea. Condition on a timestep t: Theorem A.1 gives $\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{step}}^{(t)}(\theta)] = c_Z\, \nabla_\theta J_{\text{step},t}(\theta)$. Averaging over $t \sim \omega(t)$ yields the step term $c_Z \sum_t \omega(t) \nabla_\theta J_{\text{step},t}(\theta)$. Then add $\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{base}}(\theta)] = \nabla_\theta J_{\text{seq}}(\theta)$ and use linearity of expectation. See Appendix B.2 for details.

A.3. Variance Reduction for the State-wise Estimator

DISPO uses two simple mechanisms that reduce the variance of DISPOś state-wise gradient estimates: (i) restricting gradients to actionable (masked) tokens, and (ii) averaging over Z candidates per state via cached-logit resampling.

Token-local updates. Fix a state $s = (q, x)$ of length L with masked set M of size $m = |M|$, and let R be the terminal reward of the resulting completion. Let $g_i := \nabla_\theta \log \tilde{\pi}_\theta(a_i \mid s, i)$ denote a per-position score term. Compare an estimator that propagates through all positions, $\hat{g}_{\text{full}} := \sum_{i=1}^{L} g_i\, R$, to the action-only estimator, $\hat{g}_{\text{sub}} := \sum_{i \in M} g_i\, R$.

Proposition A.3 (Variance reduction by partial updates)**.** *Assume $\{g_i\}$ are independent, zero-mean with $\mathrm{Var}[g_i] = \sigma^2$, and R is independent of $\{g_i\}$ with $\mathbb{E}[R^2] < \infty$. Then*

$$\mathrm{Var}[\hat{g}_{sub}] \le \frac{m}{L}\,\mathrm{Var}[\hat{g}_{full}].$$

Proof idea. Under the assumptions, the variance of the sum scales linearly with the number of included positions, giving the factor m/L. See Appendix B.3.

Same-state averaging via cached-logit branching. At a fixed state s, DISPO can resample Z independent branches and average their gradient contributions. With any baseline $b(s)$ that is independent of each sampled action, the variance of the averaged estimator decreases at the standard $O(1/Z)$ rate. Cached-logit branching avoids additional *rollout* forward passes to obtain these Z samples.

Proposition A.4 (Variance vs. number of branches)**.** *Fix a state s and draw i.i.d. actions $a_1, \ldots, a_Z \sim \tilde{\pi}_\theta(\cdot \mid s)$ with rewards R_z. Let $b(s)$ be any baseline independent of each a_z, and define the per-branch contribution $\hat{g}_z := (R_z - b(s))\,\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s)$ with finite variance. Let $\hat{g} := \frac{1}{Z}\sum_{z=1}^{Z} \hat{g}_z$ be the averaged estimator. Then $\mathrm{Var}[\hat{g}] = O(1/Z)$.*

Proof idea. The estimator is the average of Z i.i.d. per-branch terms under an action-independent baseline, so its variance shrinks by $1/Z$. See Appendix B.4.

Remark (group-relative baseline used in practice). Our implementation uses the within-group mean baseline (Eq. 3), which introduces dependence across branches but typically further reduces variance in practice; we empirically verify variance reduction from increasing Z (§ 4).

B. Proofs

B.1. Proof of Theorem A.1

We prove that the (unclipped) step-level loss yields a principled (scaled) policy-gradient estimator for the step-wise objective $J_{\text{step},t}(\theta)$ when treating the surrogate $\tilde{\pi}_\theta$ as the policy.

Setup. Fix a timestep index t. Recall the step-wise objective (Eq. 9)

$$J_{\text{step},t}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\mathbb{E}_{s_t\sim d_t^{\text{old}}(\cdot|q)}\mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_t)}\big[R\big(q, \textsc{Fill}(x_t, a)\big)\big],$$

where $s_t = (q, x_t)$ and a fills the masked positions at s_t. For brevity, define the terminal reward as a function of (s_t, a):

$$\mathcal{R}(s_t, a) := R\big(q, \textsc{Fill}(x_t, a)\big).$$

Policy-gradient form of $\nabla_\theta J_{\text{step},t}(\theta)$. Under Assumption (i) of Theorem A.1, the state distribution $d_t^{\text{old}}(\cdot \mid q)$ is induced by the frozen behavior model θ_{old} and is independent of θ, so we may move the gradient through the outer expectations:

$$\nabla_\theta J_{\text{step},t}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\mathbb{E}_{s_t\sim d_t^{\text{old}}(\cdot|q)}\left[\nabla_\theta \mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_t)}\big[\mathcal{R}(s_t, a)\big]\right]. \tag{14}$$

Under Assumption (ii) (differentiable, normalized $\tilde{\pi}_\theta$), the score-function identity gives

$$\nabla_\theta \mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_t)}\big[\mathcal{R}(s_t, a)\big] = \mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_t)}\big[\mathcal{R}(s_t, a)\,\nabla_\theta \log \tilde{\pi}_\theta(a \mid s_t)\big].$$

Substituting into Eq. 14 yields

$$\nabla_\theta J_{\text{step},t}(\theta) = \mathbb{E}_{q\sim\mathcal{D}}\mathbb{E}_{s_t\sim d_t^{\text{old}}(\cdot|q)}\mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_t)}\big[\mathcal{R}(s_t,a)\,\nabla_\theta \log \tilde{\pi}_\theta(a \mid s_t)\big]. \tag{15}$$

Expected gradient of the step-level loss. At a selected state $s_{k,t}$, DiSPO draws Z branched action samples $a_{k,t,1:Z}$ from the frozen behavior policy $\tilde{\pi}_{\theta_{\text{old}}}(\cdot \mid s_{k,t})$ and evaluates $\mathcal{R}(s_{k,t}, a_{k,t,z})$ for each branch. Ignoring clipping (as stated in the theorem), the per-state step loss is

$$\mathcal{L}_{\text{step}}^{(k,t)}(\theta) = -\frac{1}{Z}\sum_{z=1}^{Z} \rho_{k,t,z}(\theta)\, A_{k,t,z}, \qquad \rho_{k,t,z}(\theta) := \frac{\tilde{\pi}_\theta(a_{k,t,z} \mid s_{k,t})}{\tilde{\pi}_{\theta_{\text{old}}}(a_{k,t,z} \mid s_{k,t})},$$

where $A_{k,t,z}$ is the group-relative advantage using the within-group mean baseline (Eq. 3), i.e., $A_{k,t,z} = R_{k,t,z} - \bar{R}_{k,t}$ with $\bar{R}_{k,t} = \frac{1}{Z}\sum_{j=1}^{Z} R_{k,t,j}$. Since $\tilde{\pi}_{\theta_{\text{old}}}$ is frozen, $\nabla_\theta \rho_{k,t,z}(\theta) = \rho_{k,t,z}(\theta)\,\nabla_\theta \log \tilde{\pi}_\theta(a_{k,t,z} \mid s_{k,t})$. Therefore,

$$-\nabla_\theta \mathcal{L}_{\text{step}}^{(k,t)}(\theta) = \frac{1}{Z}\sum_{z=1}^{Z} \rho_{k,t,z}(\theta)\, A_{k,t,z}\, \nabla_\theta \log \tilde{\pi}_\theta(a_{k,t,z} \mid s_{k,t}). \tag{16}$$

Taking expectation over the branched samples $a_{k,t,1:Z} \sim \tilde{\pi}_{\theta_{\text{old}}}(\cdot \mid s_{k,t})$ and using importance weighting yields

$$\mathbb{E}\big[-\nabla_\theta \mathcal{L}_{\text{step}}^{(k,t)}(\theta) \mid q, s_{k,t}\big] = \mathbb{E}_{a_{1:Z}\sim\tilde{\pi}_\theta(\cdot|s_{k,t})}\left[\frac{1}{Z}\sum_{z=1}^{Z}(R_z - \bar{R})\,\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s_{k,t})\right], \tag{17}$$

where $R_z = \mathcal{R}(s_{k,t}, a_z)$ and $\bar{R} = \frac{1}{Z}\sum_{j=1}^{Z} R_j$. Expanding the baseline term,

$$\mathbb{E}\left[\frac{1}{Z}\sum_{z=1}^{Z} \bar{R}\,\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s_{k,t})\right] = \frac{1}{Z^2}\sum_{z=1}^{Z}\sum_{j=1}^{Z} \mathbb{E}[R_j\,\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s_{k,t})]\,.$$

For $j \neq z$, R_j is independent of a_z and $\mathbb{E}[\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s_{k,t})] = 0$, so cross terms vanish. Only the $j = z$ terms remain, yielding

$$\mathbb{E}\left[\frac{1}{Z}\sum_{z=1}^{Z} \bar{R}\,\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s_{k,t})\right] = \frac{1}{Z}\,\mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_{k,t})}\big[\mathcal{R}(s_{k,t},a)\,\nabla_\theta \log \tilde{\pi}_\theta(a \mid s_{k,t})\big].$$

Therefore,

$$\mathbb{E}\big[-\nabla_\theta \mathcal{L}_{\text{step}}^{(k,t)}(\theta) \mid q, s_{k,t}\big] = \frac{Z-1}{Z}\,\mathbb{E}_{a\sim\tilde{\pi}_\theta(\cdot|s_{k,t})}\big[\mathcal{R}(s_{k,t},a)\,\nabla_\theta \log \tilde{\pi}_\theta(a \mid s_{k,t})\big].$$

Taking outer expectations over $q \sim \mathcal{D}$ and $s_{k,t} \sim d_t^{\text{old}}(\cdot \mid q)$ yields

$$\mathbb{E}\big[-\nabla_\theta \mathcal{L}_{\text{step}}^{(t)}(\theta)\big] = \frac{Z-1}{Z}\,\nabla_\theta J_{\text{step},t}(\theta),$$

which matches Theorem A.1. □

Remark on the group baseline. If a leave-one-out baseline is used (independent of each branch action), the estimator is unbiased. With the within-group mean baseline in Eq. 3 used in our implementation, the expected gradient is scaled by the constant factor $(Z-1)/Z$, which can be absorbed into α_{step} or the learning rate.

B.2. Proof of Theorem A.2

We prove that the expected gradient of the combined loss equals the gradient of the mixed objective. Throughout, we treat the masked-token surrogate $\tilde{\pi}_\theta$ as the policy being optimized and work in the unclipped setting stated in § A.2.

Recall the mixed objective and combined loss. Let $c_Z := (Z-1)/Z$ denote the constant scaling induced by the within-group mean baseline in the step-wise estimator. The mixed objective (Eq. 12) is

$$J_{\text{mix}}(\theta) = \alpha_{\text{step}}\, c_Z \sum_t \omega(t)\, J_{\text{step},t}(\theta) + \alpha_{\text{base}}\, J_{\text{seq}}(\theta),$$

where $J_{\text{step},t}(\theta)$ is the step-wise objective (Eq. 9) and $J_{\text{seq}}(\theta)$ is the terminal surrogate objective (Eq. 11). The combined training loss is

$$\mathcal{L}(\theta) = \alpha_{\text{step}}\, \mathcal{L}_{\text{step}}(\theta) + \alpha_{\text{base}}\, \mathcal{L}_{\text{base}}(\theta).$$

By linearity of differentiation,

$$\nabla_\theta J_{\text{mix}}(\theta) = \alpha_{\text{step}}\, c_Z \sum_t \omega(t)\, \nabla_\theta J_{\text{step},t}(\theta) + \alpha_{\text{base}}\, \nabla_\theta J_{\text{seq}}(\theta). \tag{18}$$

It therefore suffices to show that $\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{step}}(\theta)] = c_Z \sum_t \omega(t)\, \nabla_\theta J_{\text{step},t}(\theta)$ and $\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{base}}(\theta)] = \nabla_\theta J_{\text{seq}}(\theta)$.

Step term. Recall that $\mathcal{L}_{\text{step}}(\theta)$ is computed by selecting timesteps according to $\omega(t)$ and applying the per-state step loss $\mathcal{L}_{\text{step}}^{(k,t)}(\theta)$ (Eq. 5) on the corresponding intermediate state(s). Condition on a particular timestep t being selected. Under the assumptions of Theorem A.1, we have

$$\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{step}}(\theta) \mid t] = c_Z\, \nabla_\theta J_{\text{step},t}(\theta),$$

where the expectation is over $q \sim \mathcal{D}$, $s_t \sim d_t^{\text{old}}(\cdot \mid q)$, and the branched action samples at that state. Taking expectation over $t \sim \omega(t)$ yields

$$\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{step}}(\theta)] = c_Z \sum_t \omega(t)\, \nabla_\theta J_{\text{step},t}(\theta). \tag{19}$$

Terminal term. By definition,

$$J_{\text{seq}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}} \mathbb{E}_{o\sim\tilde{\pi}_\theta(\cdot|q)} \big[R(q,o)\big].$$

A standard score-function (REINFORCE) argument gives

$$\nabla_\theta J_{\text{seq}}(\theta) = \mathbb{E}_{q,o\sim\tilde{\pi}_\theta} \big[R(q,o)\, \nabla_\theta \log \tilde{\pi}_\theta(o \mid q)\big],$$

and subtracting any baseline that depends on q but not on o preserves unbiasedness. The terminal GRPO loss $\mathcal{L}_{\text{base}}(\theta)$ is precisely the corresponding baseline-centered (and, in practice, importance-weighted) estimator; in the unclipped setting,

$$\mathbb{E}[-\nabla_\theta \mathcal{L}_{\text{base}}(\theta)] = \nabla_\theta J_{\text{seq}}(\theta), \tag{20}$$

which is the standard PPO/GRPO identity (see, e.g., Shao et al. (2024)).

Combine the terms. Combining Eq. 19 and Eq. 20 and using linearity of expectation,

$$\mathbb{E}[-\nabla_\theta \mathcal{L}(\theta)] = \alpha_{\text{step}}\, c_Z \sum_t \omega(t)\, \nabla_\theta J_{\text{step},t}(\theta) + \alpha_{\text{base}}\, \nabla_\theta J_{\text{seq}}(\theta) = \nabla_\theta J_{\text{mix}}(\theta),$$

where the last equality follows from Eq. 18 with the definition of $J_{\text{mix}}(\theta)$ above. This proves Theorem A.2. □

Remark (KL-regularized case). If the main text includes a KL penalty, the same argument applies after adding the term $-\lambda\, \mathbb{E}_q[\mathrm{KL}(\tilde{\pi}_\theta(\cdot \mid q) \,\|\, \tilde{\pi}_{\text{ref}}(\cdot \mid q))]$ to $J_{\text{mix}}(\theta)$ and the corresponding penalty $\lambda\, \mathcal{L}_{\text{KL}}(\theta)$ to $\mathcal{L}(\theta)$; differentiation under the expectation yields the additional gradient term.

B.3. Proof of Proposition A.3

We prove the variance reduction claim for the token-local (action-only) estimator under the stated independence assumptions.

Setup. Fix a diffusion state $s = (q, x)$ of length L with masked set $M \subseteq [L]$ of size $m = |M|$. Let R denote the (scalar) terminal reward associated with the filled candidate, and define per-position score terms

$$g_i \;:=\; \nabla_\theta \log \tilde{\pi}_\theta(a_i \mid s, i),$$

where a_i is the token sampled at position i (for $i \in M$) and $\tilde{\pi}_\theta(\cdot \mid s, i)$ is the one-step surrogate distribution at position i. Consider the two estimators (as in § A.3)

$$\hat{g}_{\text{full}} \;=\; \sum_{i=1}^{L} g_i\, R, \qquad \hat{g}_{\text{sub}} \;=\; \sum_{i\in M} g_i\, R.$$

Variance calculation. Under the assumptions of Proposition A.3, $\{g_i\}_{i=1}^{L}$ are independent, zero-mean with $\mathrm{Var}[g_i] = \sigma^2$, and R is independent of all g_i with $\mathbb{E}[R^2] < \infty$. First, independence across positions implies

$$\mathrm{Var}\left[\sum_{i\in S} g_i R\right] = \sum_{i\in S} \mathrm{Var}[g_i R] \quad \text{for any index set } S \subseteq [L].$$

Second, since g_i is independent of R and $\mathbb{E}[g_i] = 0$,

$$\mathrm{Var}[g_i R] = \mathbb{E}[g_i^2 R^2] - (\mathbb{E}[g_i R])^2 = \mathbb{E}[g_i^2]\mathbb{E}[R^2] = \sigma^2\, \mathbb{E}[R^2].$$

Therefore,

$$\mathrm{Var}[\hat{g}_{\text{full}}] = \sum_{i=1}^{L} \sigma^2 \mathbb{E}[R^2] = L\,\sigma^2 \mathbb{E}[R^2], \qquad \mathrm{Var}[\hat{g}_{\text{sub}}] = \sum_{i\in M} \sigma^2 \mathbb{E}[R^2] = m\,\sigma^2 \mathbb{E}[R^2].$$

Taking the ratio yields

$$\mathrm{Var}[\hat{g}_{\text{sub}}] = \frac{m}{L}\,\mathrm{Var}[\hat{g}_{\text{full}}],$$

which implies the stated inequality in Proposition A.3 since $m \le L$. □

Remark (vector-valued gradients). If each g_i is vector-valued, the same argument applies coordinate-wise. Equivalently, writing $\mathrm{tr}\,\mathrm{Cov}[\cdot]$ for the trace of the covariance matrix, independence implies $\mathrm{tr}\,\mathrm{Cov}[\hat{g}_{\text{sub}}] = \frac{m}{L}\,\mathrm{tr}\,\mathrm{Cov}[\hat{g}_{\text{full}}]$ under the same assumptions.

B.4. Proof of Proposition A.4

We prove the $O(1/Z)$ variance reduction from averaging Z (approximately) independent branches at a fixed state.

Setup. Fix a state s and draw i.i.d. actions $a_1, \ldots, a_Z \sim \tilde{\pi}_\theta(\cdot \mid s)$ with associated rewards R_z. Let $b(s)$ be any baseline independent of each sampled action a_z, and define the per-branch contribution

$$\hat{g}_z \;:=\; (R_z - b(s))\,\nabla_\theta \log \tilde{\pi}_\theta(a_z \mid s).$$

Define the averaged estimator

$$\hat{g} \;:=\; \frac{1}{Z}\sum_{z=1}^{Z} \hat{g}_z,$$

and assume $\hat{g}_z$ have finite variance.

Variance of an average. For the scalar case, independence gives

$$\mathrm{Var}[\hat{g}] = \mathrm{Var}\left[\frac{1}{Z}\sum_{z=1}^{Z}\hat{g}_z\right] = \frac{1}{Z^2}\sum_{z=1}^{Z}\mathrm{Var}[\hat{g}_z] = \frac{1}{Z}\,\mathrm{Var}[\hat{g}_z],$$

where $\mathrm{Var}[\hat{g}_z]$ is the same for all z since $\{\hat{g}_z\}_{z=1}^{Z}$ are i.i.d. Thus $\mathrm{Var}[\hat{g}] = O(1/Z)$.

For the vector-valued case, the same argument holds for covariance matrices. Under independence,

$$\mathrm{Cov}[\hat{g}] = \mathrm{Cov}\left[\frac{1}{Z}\sum_{z=1}^{Z}\hat{g}_z\right] = \frac{1}{Z^2}\sum_{z=1}^{Z}\mathrm{Cov}[\hat{g}_z] = \frac{1}{Z}\,\mathrm{Cov}[\hat{g}_z],$$

where the last equality uses that $\{\hat{g}_z\}_{z=1}^{Z}$ are i.i.d. and hence have identical covariance. Therefore $\mathrm{tr}\,\mathrm{Cov}[\hat{g}] = \frac{1}{Z}\,\mathrm{tr}\,\mathrm{Cov}[\hat{g}_z] = O(1/Z)$. □

Remark (group-relative baseline used in practice). Our implementation uses the within-group mean baseline (Eq. 3), which introduces dependence across branches. Proposition A.4 formalizes the standard $1/Z$ reduction for action-independent baselines; empirically, increasing Z reduces variance in our setting as well (§ 4).

C. Experimental Details

This appendix provides additional details on benchmarks, reward functions, evaluation protocols, and training/implementation settings. We follow the d1/diffu-GRPO setup where applicable (Zhao et al., 2025b), reusing its SFT procedure and programmatic reward evaluators. For policy optimization, we instantiate the terminal-feedback base objective with diffu-GRPO and SPG, and compare each baseline against its DiSPO-augmented variant under matched compute-relevant quantities. Run-specific hyperparameters, optimizer schedules, seeds, and infrastructure settings will be provided in the released configuration files.

C.1. Benchmarks, Evaluation Protocol, and Reward Functions

We evaluate on four reasoning benchmarks following Zhao et al. (2025b). For each task, we report exact-match accuracy and use a programmatic verifier to compute the scalar terminal reward $R(q, o)$ from prompt q and model completion o. The reward functions are task-specific and reused from the d1/diffu-GRPO implementation without modification: https://github.com/dllm-reasoning/d1. Importantly, DiSPO uses the same terminal reward evaluator for both the sequence-level/base objective and the state-wise objective; we do not introduce additional reward shaping or custom step-wise rewards.

GSM8K. GSM8K (Cobbe et al., 2021) is a grade-school math word-problem benchmark. It uses exact match on the final numeric answer after applying the same answer-extraction and normalization rules. We use the training data publicly available https://huggingface.co/datasets/openai/gsm8k. *Reward* is computed by considering multiple axes, i.e., format reward (max. 1.625) and correctness reward (max. 2.0).

MATH500. MATH500 (Lightman et al., 2024) is a subset of MATH focusing on competition-level problems. *Reward* is computed by considering the two axes, i.e., format reward (max 1.0) and correctness reward (max 2.0)

Sudoku. 4×4 Sudoku tasks is synthetic benchmark for planning. We use the training data publicly available https://github.com/Black-Phoenix/4x4-Sudoku-Dataset. As for the evaluation data, we use the synthetically created problem set by using specific implementation https://www.ocf.berkeley.edu/~arel/sudoku/main.html. Reward is the fraction of empty cells filled correctly.

Countdown. Countdown is an arithmetic-planning benchmark, whose training data publicly available https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4. *Evaluation* parses the output into an arithmetic expression, verifies that it uses the provided numbers under the task rules, and checks that the expression evaluates exactly to the target value. *Reward* is computed by considering the two axes, i.e., nearly-correct answers obtain 0.1 and the perfect answers gain 1.0.

C.2. Inference Details

For inference, we use semi-autoregressive decoding with 32-token blocks (Arriola et al., 2025), unmasking the 2 highest-confidence tokens per step within each block. We evaluate at $N_{\text{gen}} \in \{128, 256, 512\}$ using zero-shot prompting and greedy decoding.

C.3. Training and Implementation Details

Model variants and SFT. We evaluate two model variants: LLaDA-8B-Instruct (Nie et al., 2025) and its s1k-supervised fine-tuned variant (Muennighoff et al., 2025). For the s1k variant, we reuse the SFT recipe from d1, including the data, schedule, and hyperparameters: https://github.com/dllm-reasoning/d1. We keep the architecture, tokenizer, and denoising schedule fixed unless explicitly varied in diagnostic experiments.

Base optimizers and matched budgets. We instantiate the terminal-feedback base objective with diffu-GRPO (Zhao et al., 2025b) and SPG (Wang et al., 2026). For diffu-GRPO, we follow the d1 training recipe. For SPG, we implement the SPG objective in the d1 codebase so that data loading, reward evaluation, denoising, and training infrastructure remain shared. For each base optimizer and task, we train the baseline and its DiSPO-augmented variant with matched compute-relevant quantities, including N_{gen}=256, rollout budget, sequence-level group size, denoising schedule, and number of RL update steps. The diffu-GRPO update steps are $3800/5000/7700/6600$ for Sudoku/Countdown/GSM8K/MATH500, respectively. The SPG update steps are $4000/4600/4600/2600$ for Sudoku/Countdown/GSM8K/MATH500, respectively. Run-specific settings are provided in the released configuration files.

Other settings. Unless stated otherwise, DiSPO uses lightweight branching settings: branch size Z=2 per selected intermediate state and one selected timestep per trajectory, $|T_{\text{sub}}|$=1. Timesteps are sampled from a late-biased polynomial distribution $\omega(t)$ with exponent k=4. We set α_{base}=1 and sweep the step-wise weight over $\alpha_{\text{step}} \in \{0.1, 0.5, 0.9\}$; run-specific choices are recorded in the released configuration files.

SPG-specific settings. We implement SPG in the d1 codebase using the practical settings described in Wang et al. (2026). For Monte Carlo estimation, we use the block-wise masking strategy: each sample randomly selects one block, keeps previous blocks mostly clean (with optional light perturbation on prompt/clean tokens), fully masks later blocks, and applies random masking within the selected block. For negative-advantage traces, we use the mixture estimator from SPG, blending the EUBO and ELBO terms with coefficient w when $A < 0$. We keep these SPG-specific choices fixed for both the SPG baseline and $\text{DiSPO}_{\text{SPG}}$.

Optimization. Following Zhao et al. (2025b), we use LoRA with a rank $r = 128$ and scaling factor $\alpha = 64$, and AdamW optimizer (Loshchilov & Hutter, 2019). We basically set hyperparameters as follows: $\beta_1 = 0.9, \beta_2 = 0.99$, weight decay 0.1, learning rate $3\times$ or 5×10^{-6}, gradient clipping 0.2. We train on 4 NVIDIA H100-94G GPUs with batch size 6 per GPU, gradient accumulation steps of 4, and sequence length of 256 tokens.

D. Empirical Measurement of Step-level Gradient Variance

We detail the protocol used to produce Fig. 3, which measures the variance of the step-level gradient estimator via repeated same-state branching.

Quantity measured. Fix an intermediate diffusion state $s = (q, x)$ and draft size Z. Let $\mathcal{L}_{\text{step}}(s; \theta)$ denote the *unclipped* per-state step loss obtained by instantiating Eq. 5 at state s with Z same-state branches, using the state-wise surrogate log-probability (Eq. 4) and group advantages (Eq. 3). We define the step-level gradient estimator as the update direction

$$\hat{g}_{\text{step}}(s) \;:=\; -\nabla_\theta \mathcal{L}_{\text{step}}(s; \theta) \in \mathbb{R}^d,$$

We summarize the variance of this random vector by the trace covariance

$$\mathrm{trCov}\big(\hat{g}_{\text{step}}(s)\big) := \mathrm{tr}\big(\mathrm{Cov}\big[\hat{g}_{\text{step}}(s)\big]\big) = \mathbb{E}\big[\|\hat{g}_{\text{step}}(s)\|^2\big] - \|\mathbb{E}[\hat{g}_{\text{step}}(s)]\|^2 .$$

Empirically, for each fixed state s we repeat same-state branching R times to obtain $\{\hat{g}^{(r)}_{\text{step}}(s)\}_{r=1}^{R}$ and estimate

$$\widehat{\mathrm{trCov}}\big(\hat{g}_{\text{step}}(s)\big) = \frac{1}{R-1} \sum_{r=1}^{R} \Big\| \hat{g}^{(r)}_{\text{step}}(s) - \bar{g}_{\text{step}}(s) \Big\|^2, \qquad \bar{g}_{\text{step}}(s) = \frac{1}{R} \sum_{r=1}^{R} \hat{g}^{(r)}_{\text{step}}(s).$$

Task, checkpoint, and state collection. We run this diagnostic on 4×4 Sudoku for efficiency and use a fixed model checkpoint from DiSPO training (LLaDA optimized with DiSPO; step-3800). We collect intermediate states from 64 prompts and sample 8 states per prompt, yielding 512 candidate states.

State filtering and paired aggregation. To ensure a fair comparison across different conditions (e.g., different Z and action-only vs full-token updates), we fix the same state set for all conditions and retain only non-degenerate states; at the state (i) at least one actionable token contributes to the step loss (denominator > 0), and (ii) among the R repeated branchings, at least one trial yields an advantage > 0. After filtering, we aggregate $\widehat{\mathrm{trCov}}(\hat{g}_{\text{step}}(s))$ over 276 states.

Compared settings. $Z = 2$ with action-only updates is the default setting used in our main experiments (Table 1); Including that, for the Z ablation, we test $Z \in \{2, 3, 4\}$; for the update-target ablation, we test the *full-tokens* and *action-only* variants. Bars in Fig. 3 show the difference in gradient variance from the default ($Z = 2$, action-only), with error bars indicating paired 95% bootstrap CIs over states.

E. Reference Comparison with Terminal Reward Shaping

Difference in objectives. SAPO (Xie et al., 2025) and DiSPO both leverage intermediate diffusion states, but they do so for different purposes. SAPO scores intermediate states to *shape the terminal rewards*, whereas DiSPO directly optimizes *intermediate-state actions*—the mask fillings sampled at a fixed partially-masked state—by comparing same-state counterfactual fillings under the unchanged terminal reward. Hence, the two approaches are best viewed as complementary.

Difference in compute. This difference in design also leads to substantially different compute profiles. As summarized in Table 8, DiSPO adds overhead mainly through extra reward evaluations for same-state branches and one-step surrogate evaluations for step-wise log-probabilities, while matching diffusion-rollout forward passes and update steps. In contrast, SAPO typically incurs additional *multi-step continuation rollouts* from intermediate states to estimate shaping terms, which increases the dominant cost—diffusion-rollout forward passes—often by a large margin. Therefore, SAPO is not only complementary in approach, but can also be considerably more expensive in required rollout compute.

Table 8. **Operation counts per prompt including SAPO.** K: # rollouts, T: # denoising steps/rollout, $\mathcal{S}$: selected states (DiSPO), Z: # branches/state, N_m: # MC prompt masks for $\mathbb{E}_m[\cdot]$, U: # update steps. For SAPO, $\mathcal{S}_{\text{sapo}}$ is the set of intermediate states used for step-aware reward estimation, N_{sapo} is the # continuation rollouts per selected state, and $\bar{T}_{\text{rem}}$ is the average remaining denoising steps.

Compute items	**w/o DiSPO**	**w/ DiSPO**	**w/ SAPO**
Diffusion rollout forward passes	KT	KT	$KT + N_{\text{sapo}}\lvert\mathcal{S}_{\text{sapo}}\rvert\bar{T}_{\text{rem}}$
Optimizer update steps	U	U	U
One-step surrogate (terminal logps)	$2N_mK$	$2N_mK$	$2N_mK$
Reward evaluations	K	$K + \lvert\mathcal{S}\rvert Z$	$K + N_{\text{sapo}}\lvert\mathcal{S}_{\text{sapo}}\rvert$
One-step surrogate (step-wise logps)	0	$2N_m\lvert\mathcal{S}\rvert$	0

Reference comparison in benchmarks. Despite the *mismatch* in objectives and compute, SAPO is a representative step-aware method, so we include a *reference* comparison to contextualize DiSPO. ***This comparison should be interpreted with caution***: the methods optimize different training signals (shaped vs. unshaped terminal rewards) and can require very different rollout compute (Table 8), so the evaluation is not an apples-to-apples matched-budget study. Table 1 reports that, on average, DiSPO outperforms SAPO, despite using substantially less rollout compute.

Takeaway. These results suggest that, among ways to leverage intermediate diffusion states, *directly optimizing intermediate mask-filling actions* can be a particularly effective alternative to using intermediate-state scores only for reward shaping, achieving competitive performance *at a fraction of the rollout compute*. At the same time, since SAPO and DiSPO act on different axes (reward shaping vs. action-level optimization), they are potentially complementary; we leave a careful study of joint training recipes to future work.

F. Discussion on the Wall-clock Overhead

Observed overhead. In our current implementation, DiSPO runs at lower throughput than the terminal-feedback baseline (reported in § 4.6) even at modest settings (e.g., Z=2). This slowdown reflects engineering overhead rather than additional rollout/update compute, since the dominant algorithmic costs (KT rollout forward passes and U updates) are matched.

Likely causes. The main contributors are repeated, non-amortized calls: (i) evaluating multiple same-state branches and (ii) computing step-wise log-probs via one-step surrogate calls at selected states. When executed largely sequentially, these introduce frequent kernel launches, synchronization points, and host-side overhead that can dominate wall-clock time despite modest extra FLOPs.

Optimization opportunities. These costs are not intrinsic to the method and should be reducible by (1) batching same-state branches (amortizing policy/surrogate/reward calls over Z), (2) caching state representations for step-wise terms, and (3) reducing kernel/IO overhead (e.g., call fusion / compilation and fewer synchronization points). We therefore, as a reference, additionally reported wall-clock-matched curves (Fig. 4) to ensure the performance gains (rewards and accuracy) are robust to current implementation overheads.

Relation to inference-time decoding efficiency. This discussion concerns training-time optimization overhead. It is orthogonal to inference-time MDLM acceleration methods, which reduce *inference cost* by shortening denoising schedules, reusing/skipping computation across denoising steps, or early stopping (Luxembourg et al., 2025; Israel et al., 2025; Wei et al., 2025; Ma et al., 2025; Liu et al., 2025; Wu et al., 2025; Oba et al., 2026). In contrast, DiSPO targets *training-time* credit assignment by reusing rollout-cached logits to compare fixed-state mask fillings.

G. Stability diagnosis of intermediate branching

For each checkpoint at global step $s \in \{900, 1900, 3800\}$, we resumed training and replayed $\texttt{outer_batches} = 8$ with $\mu = 12$ inner updates per outer batch (i.e., steps $s \to s + 96$). At every inner update, we computed branch-level $|\log \rho|$ separately for Terminal and Step losses: Terminal uses completion-token likelihood ratios, while Step uses action-token likelihood ratios only. Token-level log-ratios were aggregated within each branch by mean, then pooled over all inner updates.

Table 9 reports pooled median, pooled p99, and clip fraction. Sample sizes are $N_{\text{term}} = 9216$ for all rows, and $N_{\text{step}} = 15480$ ($900 \to 996$), 15584 ($1900 \to 1996$), 15160 ($3800 \to 3896$). For Terminal, the pooled sample size is $N_{\text{term}} = \texttt{outer_batches} \times \mu \times G \times B_{\text{dev}} \times A = 8 \times 12 \times 4 \times 6 \times 4 = 9216$, where G is the number of GPUs, B_{dev} is per-device train batch size, and A is gradient accumulation steps. Equivalently, Step keeps one scalar $|\log \rho|$ per branch after token-mean aggregation over the effective mask ($Z = 2$).

Table 9. **Stability diagnostics of cached-logit reuse on Sudoku.**

Global Steps	Terminal			Step		
	median	**p99**	**clip** (%)	**median**	**p99**	**clip** (%)
900→996	0.0065	0.0480	0.011	0.0053	0.0455	0.000
1900→1996	0.0071	0.0431	0.013	0.0046	0.0425	0.003
3800→3896	0.0065	0.0377	0.016	0.0048	0.0418	0.009

H. Existing Assets and Licenses

Table 10 summarizes the existing assets used in our experiments. We use these assets only for research evaluation and credit the original creators through citations and source links. For assets whose license is not explicitly specified on the source page, we mark the license as "not specified" and refer readers to the original source.

Table 10. **Existing assets used in this work.** Licenses are reported according to the source pages available at the time of writing.

Asset	Use	License / terms
LLaDA-8B-Instruct (Nie et al., 2025)	Base model	MIT
d1 / diffu-GRPO code (Zhao et al., 2025b)	Training recipe, rewards, baselines	Apache-2.0
s1K (Muennighoff et al., 2025)	SFT data	Apache-2.0
GSM8K (Cobbe et al., 2021)	Math benchmark	MIT
MATH500 (Lightman et al., 2024)	Math benchmark	MIT source split via PRM800K
Countdown (Zhao et al., 2025b)	Planning benchmark	Not specified on source page
4×4 Sudoku dataset	Planning benchmark	Not specified on source page