PBPO: PREFIX-BLOCK POLICY OPTIMIZATION FOR DIFFUSION LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion Large Language Models (dLLMs) commonly decode responses by proceeding autoregressively across blocks while iteratively denoising multiple tokens within each block. This structure makes block-level policy optimization a natural adaptation of autoregressive policy optimization. However, the exact likelihood of a completed block is intractable because it marginalizes over the intermediate states and reveal orders of the denoising process. To realize the block-level formulation, we construct a prefix-block state and use a single model evaluation to define a factorized one-step surrogate for the block likelihood. The resulting block surrogate ratio is a product of its token surrogate ratios, allowing variation at individual positions to propagate across the entire block before clipping. We therefore introduce Prefix-Block Policy Optimization (PBPO), which preserves block-level conditioning but applies surrogate importance weighting and clipping separately to each token. PBPO further diversifies repeated updates by resampling corrupted views of the prompt and revealed prefix, a design motivated by masked-dLLM pre-training. Under our analysis assumptions, block-product weighting amplifies the variance of gradient contributions relative to token-local weighting. Across two LLaDA-family initializations, six benchmarks, and three generation lengths, PBPO achieves competitive performance against existing dLLM RL methods.

1 INTRODUCTION

Reinforcement learning with verifiable rewards has substantially improved the reasoning capabilities of autoregressive large language models (LLMs) (Shao et al., 2024; DeepSeek-AI, 2025; Yu et al., 2025). Masked diffusion language models (dLLMs) provide a complementary generation paradigm: rather than predicting strictly from left to right, they iteratively denoise masked positions and can update multiple tokens in parallel. Recent works such as LLaDA (Nie et al., 2025) and LLaDA-1.5 (Zhu et al., 2026) scale this paradigm to open-ended language generation and employ block-wise decoding, which combines autoregressive dependencies across blocks with parallel denoising within each block. Effectively adapting RL post-training to this decoding structure remains an important challenge, which lies in likelihood evaluation, as autoregressive policies admit a unique left-to-right factorization, so each generated token has a tractable conditional likelihood and importance sampling ratio. In contrast, for a dLLM, its response likelihood marginalizes over all denoising trajectories and is therefore intractable to evaluate, preventing a direct adaptation of autoregressive policy optimization objectives.

Recent work addresses this challenge through different approximations and reformulations. d1 uses one-step token predictions under a mean-field approximation (Zhao et al., 2025), GDPO estimates a sequence-level diffusion lower bound (Rojas et al., 2025), wd1 removes explicit policy ratios in favor of reward-weighted likelihood optimization (Tang et al., 2025), and DMPO casts post-training as distribution matching to a reward-modified target (Zhu et al., 2025). These approaches primarily differ in how they approximate the intractable response likelihood or or reformulate the intractable objective. We study a complementary question induced by block-wise decoding: once a surrogate is chosen, at what granularity should policy change be measured and clipped?

As block-wise generation proceeds autoregressively across blocks, the most direct adaptation is to treat each completed block as the optimization unit conditioned on the previously generated blocks. Its exact conditional likelihood remains intractable because the block itself is produced by multi-

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

step denoising. We therefore define a *prefix-block state* that reveals the completed prefix blocks while masking the current and future blocks, and approximate the completed block with factorized one-step tokens denoising from this shared state. This construction yields a tractable surrogate for a direct block-level policy objective. However, the direct construction also exposes an optimization difficulty. The surrogate ratio of a block is the product of all token surrogate ratios within that block, and this single product weights the aggregate contribution of its tokens. Consequently, a large or small ratio at one position can affect every token update in the block, with within-block variation compounding before clipping. Our analysis characterizes this variance amplification under its stated assumptions, and we observe the corresponding instability when training the direct block-level formulation.

We introduce **Prefix-Block Policy Optimization (PBPO)**, which evaluates every token in the current block conditioning on the same prefix-block state, but clips each token's surrogate ratio independently. When a rollout batch is reused for multiple updates, PBPO also resamples corrupted views of the prompt and revealed prefix. Motivated by masked-dLLM pre-training, this per-update corruption diversifies the conditioning signal without requiring additional computational overhead on rollouts generation. Our main contributions are:

- We formulate two policy optimization objectives for dLLMs: a direct block-level objective based on a factorized one-step likelihood surrogate, and PBPO, which retains prefix-block conditioning while applying clipped surrogate ratios at token level.
- We analyze why multiplicative block-level ratios can amplify gradient variance under our assumptions and empirically demonstrate that PBPO provides more stable optimization than the direct block-level adaptation.
- We evaluate PBPO across two LLaDA-family initializations and six benchmarks under varied generation lengths and denoising settings, where it achieves competitive performance compared to existing dLLM RL methods.

2 Preliminaries

2.1 Masked Diffusion Language Models

Foundational masked diffusion models specialize discrete diffusion to an absorbing-state corruption process (Shi et al., 2024; Sahoo et al., 2024; Ou et al., 2025). Let $\mathcal{V}$ denote the token vocabulary, $x_0 = (x_0^1, \ldots, x_0^L) \in \mathcal{V}^L$ a clean sequence of length L, and $[\texttt{MASK}]$ the absorbing mask token. At diffusion time $t \in [0, 1]$, the forward process independently masks each token with probability $1-\alpha_t$, where the noise schedule $\alpha_t \in [0, 1]$ decreases from $\alpha_0 = 1$ to $\alpha_1 = 0$, producing a corrupted sequence x_t according to $q_{t|0}(x_t \mid x_0) = \prod_{k=1}^{L} \left(\alpha_t \, \mathbf{1}[x_t^k = x_0^k] + (1 - \alpha_t) \, \mathbf{1}[x_t^k = [\texttt{MASK}]]\right)$. As t increases, x_t therefore transitions from clean data to a fully masked sequence. The reverse process is parameterized by a bidirectional denoising model f_θ that predicts the clean token at every masked position. For absorbing diffusion, the negative evidence lower bound reduces to a weighted cross-entropy over these predictions (Shi et al., 2024; Sahoo et al., 2024). Modern dLLMs scale this formulation using large Transformer models, for instance, LLaDA (Nie et al., 2025) adopts the linear schedule $\alpha_t = 1 - t$, yielding

$$\mathcal{L}_{\mathrm{DCE}}(\theta) = -\mathbb{E}_{\substack{x_0 \sim p_{\mathrm{data}},\, t \sim \mathcal{U}[0,1], \\ x_t \sim q_{t|0}(\cdot|x_0)}} \left[\frac{1}{t} \sum_{k=1}^{L} \mathbf{1}[x_t^k = [\texttt{MASK}]] \log f_\theta(x_0^k \mid x_t) \right], \tag{1}$$

where p_{data} is the distribution of clean sequences. Equation 1 is an upper bound on the negative log-likelihood of the induced generative distribution. For conditional language generation, let q and o denote the prompt and response, and let $\pi_\theta(o \mid q)$ denote the conditional response distribution induced by the complete reverse process. Modern dLLMs apply the masking and denoising processes only to o: the response begins fully masked and is iteratively revealed through denoising. Therefore, the conditional response likelihood is $\pi_\theta(o \mid q) = \mathbb{E}_\sigma \left[\prod_{m=1}^{L} f_\theta \left(o^{\sigma_m} \mid q, o^{\sigma_{<m}} \right) \right]$, where $\sigma = (\sigma_1, \ldots, \sigma_L)$ is a token-reveal order, and positions in $\sigma_{<m}$ are revealed and all remaining response positions are masked. Thus, $\log \pi_\theta(o \mid q)$ is the log of a marginal over reveal orders, making its exact evaluation intractable.

2.2 Reinforcement Learning for Large Language Models

Reinforcement learning with verifiable rewards (RLVR) improves LLM reasoning by optimizing sequence-level feedback (Ouyang et al., 2022; Shao et al., 2024; DeepSeek-AI, 2025). PPO (Schulman et al., 2017) stabilizes policy updates through clipped importance ratios, while GRPO (Shao et al., 2024) estimates advantages from a group of responses and avoids training a separate value model. To be more specific, given a prompt $q \sim \mathcal{D}$, the behavior policy π_{old} samples G responses $\{o_i\}_{i=1}^{G} \overset{\text{i.i.d.}}{\sim} \pi_{\text{old}}(\cdot \mid q)$, and each response receives a reward $R(q, o_i)$. Following prior work (Liu et al., 2025; Zhao et al., 2025), we use the unnormalized group-relative advantage $A_i = R(q, o_i) - \frac{1}{G}\sum_{j=1}^{G} R(q, o_j)$. An autoregressive policy admits the exact left-to-right factorization $\pi_\theta(o_i \mid q) = \prod_{k=1}^{|o_i|} \pi_\theta\left(o_i^k \mid q, o_i^{<k}\right)$. Each generated token therefore has a unique conditioning state $(q, o_i^{<k})$, and its importance sampling ratio is $r_i^k(\theta) = \frac{\pi_\theta\left(o_i^k \mid q, o_i^{<k}\right)}{\pi_{\text{old}}\left(o_i^k \mid q, o_i^{<k}\right)}$. The corresponding clipped objective is

$$J_{\text{RL}}(\theta) = \mathbb{E}_{q\sim\mathcal{D},\{o_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{k=1}^{|o_i|}\min\left(r_i^k(\theta)A_i, \text{clip}\left(r_i^k(\theta), 1-\epsilon_l, 1+\epsilon_r\right)A_i\right)\right]. \tag{2}$$

Clipping limits the contribution of ratios outside the interval $[1-\epsilon_l, 1+\epsilon_r]$. This notation includes symmetric clipping in PPO and GRPO as well as the asymmetric Clip-Higher rule introduced by DAPO (Yu et al., 2025).

Policy optimization for masked dLLMs. As shown in Section 2.1, the conditional response likelihood $\pi_\theta(o_i \mid q)$ is intractable, resulting in policy optimization objectives involving $\pi_\theta(o_i \mid q)$ intractable. d1 addresses this challenge using one-step token predictions with a mean-field approximation (Zhao et al., 2025). The next section addresses the same central challenge for block-wise generation by constructing a tractable block-likelihood approximation, which serves as the starting point for PBPO.

3 Methodology

We introduce **Prefix-Block Policy Optimization (PBPO)**, a policy optimization method for RL post-training of masked diffusion language models (dLLMs). PBPO is motivated by the block-wise decoding procedure used by modern dLLMs (Nie et al., 2025; Zhu et al., 2026). Specifically, the procedure proceeds autoregressively across blocks and iteratively denoise multiple positions within the current block. Therefore, the denoised response is therefore naturally viewed as a sequence of block-level actions, where each block is conditioned on the previously generated blocks. The most direct adaptation of autoregressive policy optimization is to replace its token-level action with this block-level action.

However, realizing this block-level formulation requires evaluating the conditional likelihood of each completed block, which is intractable. Specifically, as a block is produced through multi-step denoising, obtaining its likelihood requires marginalizing over all possible intermediate masked states and token-reveal orders that can produce the same completed block. We therefore approximate the block likelihood with a single forward pass conditioned on a partially masked response, where the previously generated blocks are unveiled, and the current and future blocks remain masked. We refer to this conditioning input, as the *prefix-block state*, which is formalized in Section 3.1. This approximation makes the direct block-level policy-optimization formulation tractable (Section 3.2). Although block-level policy optimization is the natural starting point, its importance ratio introduces a distinct optimization difficulty: the ratio for a block is the product of the ratios of all tokens within that block. In Section 3.4, our analyses further show that this multiplicative structure can amplify the variance of policy-gradient estimates, and also result in unstable optimization in practice. Together, these findings motivate the final PBPO objective, which is elaborated in Section 3.3. PBPO retains the prefix-block conditioning states used by the block-level formulation, but decomposes each block ratio and applies importance weighting and clipping separately to its tokens. It thereby preserves alignment with block-autoregressive generation while avoiding a single multiplicative importance weight for an entire block.

3.1 PREFIX-BLOCK STATES

As motivated above, approximating the likelihood of each block requires a conditioning input associated with the corresponding block boundary. Let $\pi_{\theta'}(o^k \mid q, z)$ denote the categorical probability assigned to the k-th token by one forward pass of a masked dLLM with parameters θ', conditioned on prompt q and a partially masked response z. A masked position in z is represented by the special token `[MASK]`. Let B_l denote the predefined block length. Given a response o_i, we divide it into $B_i = \left\lceil \frac{|o_i|}{B_l} \right\rceil$ consecutive blocks. The b-th block spans token positions $(b-1)B_l + 1$ through $\min(bB_l, |o_i|)$. For each $b \in \{1, \ldots, B_i\}$, we define the corresponding *prefix-block state* as

$$z_i^{<b} = o_i^{1:(b-1)B_l} \oplus \underbrace{\texttt{[MASK]} \ldots \texttt{[MASK]}}_{|o_i|-(b-1)B_l}, \tag{3}$$

where $\oplus$ denotes concatenation. The state $z_i^{<b}$ reveals the previously generated $(b-1)$ blocks and masks every position in the current and future blocks. For the first block, no response tokens have yet been revealed, so its prefix-block state is fully masked: $z_i^{<1} = \underbrace{\texttt{[MASK]} \ldots \texttt{[MASK]}}_{|o_i|}$.

The prefix-block state therefore represents the conditioning input immediately before block b is denoised. Thus, the sequence of prefix-block states preserves the autoregressive dependency across blocks.

3.2 BLOCK-LEVEL POLICY OPTIMIZATION

Using the prefix-block state $z_i^{<b}$, we approximate the intractable conditional likelihood of the b-th block with a single model forward pass. As a masked dLLM predicts all positions in the block in parallel, so we define the surrogate block probability as the factorized product as follows

$$\pi_* \left(o_i^{(b-1)B_l+1:\min(bB_l,|o_i|)} \mid q, z_i^{<b} \right) = \exp \left(\sum_{k=(b-1)B_l+1}^{\min(bB_l,|o_i|)} \log \pi_* \left(o_i^k \mid q, z_i^{<b} \right) \right), \tag{4}$$

where $*$ denotes either the current policy π_θ or the behavior policy π_{old}. The surrogate retains the block-autoregressive conditioning structure because each block is evaluated from its corresponding $z_i^{<b}$. Applying the same approximation to both policies gives the block-level surrogate ratio

$$r_i^b = \frac{\pi_\theta \left(o_i^{(b-1)B_l+1:\min(bB_l,|o_i|)} \mid q, z_i^{<b} \right)}{\pi_{\text{old}} \left(o_i^{(b-1)B_l+1:\min(bB_l,|o_i|)} \mid q, z_i^{<b} \right)} = \prod_{k=(b-1)B_l+1}^{\min(bB_l,|o_i|)} \frac{\pi_\theta \left(o_i^k \mid q, z_i^{<b} \right)}{\pi_{\text{old}} \left(o_i^k \mid q, z_i^{<b} \right)}. \tag{5}$$

Using the group-relative sequence-level advantage $A_i = R(q, o_i) - \frac{1}{G} \sum_{j=1}^{G} R(q, o_j)$, where details for task-specific reward definitions used in our experiments are provided in Appendix A.1, the block-level PPO-style objective can be formulated as

$$J_{\text{block}}(\theta) = \mathbb{E}_{q \sim \mathcal{D}, \{o_i\}_{i=1}^{G} \sim \pi_{\text{old}}(\cdot|q)} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{B_i} \sum_{b=1}^{B_i} \min \left(r_i^b A_i, \text{clip} \left(r_i^b, 1-\epsilon_l, 1+\epsilon_r \right) A_i \right) \right]. \tag{6}$$

The next subsection retains its prefix-block conditioning while revising the granularity of importance weighting.

3.3 PREFIX-BLOCK POLICY OPTIMIZATION

The block-level objective (Equation 6) follows the generation structure, but its product ratio couples the policy changes of all tokens within a block. PBPO removes this coupling while preserving the prefix-block conditioning introduced above. Specifically, all tokens within a block remain conditioned on the same $z_i^{<b}$, but each token receives its own importance ratio and clipping operation. PBPO therefore separates the *conditioning granularity*, which remains block-level, from the *importance-weighting granularity*, which is token-level.

Algorithm 1 Prefix-Block Policy Optimization (PBPO)

Require: Initial masked dLLM policy π_0, prompt distribution $\mathcal{D}$, reward function R, group size G, and block length B_l, block-diffusion sampling strategy $\mathcal{S}$, inner updates μ, masking probabilities p_q and p_z, and clipping coefficients ϵ_l, ϵ_r.

Initialize the trainable policy $\pi_\theta \leftarrow \pi_0$
while not converged **do**
  Set the behavior policy $\pi_{\text{old}} \leftarrow \pi_\theta$
  Sample a prompt $q \sim \mathcal{D}$, and G responses $\{o_i\}_{i=1}^{G} \sim \pi_{\text{old}}(\cdot \mid q)$ using $\mathcal{S}$
  Compute $R(q, o_i)$ and $A_i \leftarrow R(q, o_i) - \frac{1}{G}\sum_{j=1}^{G} R(q, o_j), \forall i \in \{1, \ldots, G\}$
  for $u = 1, \ldots, \mu$ **do**
    for $i = 1, \ldots, G$ **do**
      Construct $\widetilde{q}_i^{(u)} \leftarrow \mathcal{M}_{p_q}(q)$
      Partition o_i into $B_i \leftarrow \lceil |o_i|/B_l \rceil$ consecutive blocks
      for $b = 1, \ldots, B_i$ **do**
        Construct the corrupted prefix-block state $\widetilde{z}_i^{<b,(u)} \leftarrow$ Equation 8
        Compute $r_i^{t,(u)}$ for every token t in block $b \leftarrow$ Equation 9

$$\widehat{J}_{\text{PBPO}}^{(u)}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\Big(r_i^{t,(u)}A_i, \text{clip}\Big(r_i^{t,(u)}, 1-\epsilon_l, 1+\epsilon_r\Big)A_i\Big).$$

      Update θ by applying one optimizer step to minimize $-\widehat{J}_{\text{PBPO}}^{(u)}(\theta)$
return π_θ

For the t-th token of response o_i, let its block index be $b_i^t = \left\lceil \frac{t}{B_l} \right\rceil$. PBPO defines the token-level surrogate ratio under the corresponding prefix-block state as $r_i^t = \frac{\pi_\theta\left(o_i^t \mid q, z_i^{<b_i^t}\right)}{\pi_{\text{old}}\left(o_i^t \mid q, z_i^{<b_i^t}\right)}$ Thus, PBPO optimizes the following objective

$$J_{\text{PBPO}}(\theta) = \mathbb{E}_{q\sim\mathcal{D}\{o_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\left(r_i^t A_i, \text{clip}\left(r_i^t, 1-\epsilon_l, 1+\epsilon_r\right)A_i\right)\right]. \tag{7}$$

Equation 7 clips each r_i^t independently, as a policy change at one position therefore affects that token's surrogate term rather than being multiplied into a shared ratio for the entire block. PBPO remains distinct from autoregressive token-level policy optimization through its conditioning states. An autoregressive model evaluates token o_i^t from the token prefix $o_i^{<t}$, whereas PBPO evaluates it from the prefix-block state $z_i^{<b_i^t}$. Tokens in the same block share this state, while tokens in different blocks condition on different revealed prefixes. Consequently, token-level importance weighting does not require token-by-token forward passes, as a single dLLM forward pass produces the probabilities for all tokens that share a prefix-block state.

3.3.1 Diversified Conditioning via Per-Update Masking

Masked dLLMs are pre-trained to predict clean tokens from randomly corrupted inputs. For example, LLaDA samples a noise level and constructs a corresponding partially masked sequence during pre-training (Nie et al., 2025). This training paradigm motivates exposing a fixed rollout through different corrupted conditioning views during policy optimization, with the goal of diversifying the training signal. We therefore resample the conditioning context at every update. Following d1 (Zhao et al., 2025), we independently resample a random prompt mask. We additionally sample one prefix masking pattern for each response; for block b, we restrict this pattern to its $(b-1)$ previously generated blocks. To be more specific, let $\mathcal{M}_p(x)$ denote the resulting masked sequence of x, in which each token is independently replaced with `[MASK]` with probability p. Suppose $\{o_i\}_{i=1}^{G} \sim \pi_{\text{old}}(\cdot \mid q)$ has been collected and is reused for μ policy-update iterations. We use p_q and p_z to denote the masking probabilities for the prompt and revealed prefix, respectively. For each response o_i and update iteration $u \in \{1, \ldots, \mu\}$, each prompt token is independently masked with

probability p_q, so the corrupted prompt is constructed as $\widetilde{q}_i^{(u)} = \mathcal{M}_{p_q}(q)$. Then, we sample one response-wide masking pattern by selecting each response position independently with probability p_z. For obtaining token's likelihood in block b, we apply only the portion of this pattern corresponding to the revealed prefix $o_i^{1:(b-1)B_l}$. The corrupted prefix-block state for block b can be obtained as

$$\widetilde{z}_i^{<b,(u)} = (\mathcal{M}_{p_z}(o_i))^{1:(b-1)B_l} \oplus \underbrace{\texttt{[MASK]} \ldots \texttt{[MASK]}}_{|o_i|-(b-1)B_l}, \tag{8}$$

which additionally corrupts only the revealed prefix; the current and future blocks remain fully masked as in the original prefix-block state $z_i^{<b}$. The resulting PBPO importance ratio at update iteration u is

$$r_i^{t,(u)} = \frac{\pi_\theta\left(o_i^t \mid \widetilde{q}_i^{(u)}, \widetilde{z}_i^{<b_i^t,(u)}\right)}{\pi_{\text{old}}\left(o_i^t \mid \widetilde{q}_i^{(u)}, \widetilde{z}_i^{<b_i^t,(u)}\right)}. \tag{9}$$

Per-update masking therefore converts one rollout batch into multiple conditioning views without additional rollout generation. Empirically, *only applying prompt masking* and *masking both prompt and prefix-block states* consistently yield better results than clean prompt and clean prefix-block states, as reported in Table 2 in Section 4.3.

The complete algorithmic training procedure of PBPO is presented in Algorithm 1.

3.4 Why Token-Level Importance Weighting?

The block-level objective (Equation 6) and PBPO use the same prefix-block conditioning but differ in the granularity of importance weighting. We now analyze how this distinction affects gradient variance. Theorem 1 formalizes the comparison (the full proof deferred to Appendix C), showing that the multiplicative block ratio couples the contributions of all tokens in a block: an unusually large or small ratio at one position can influence the update contributed by the entire block. PBPO instead confines each importance weight to its corresponding token, avoiding this within-block propagation. Empirically, we observe the same behavior. In Figure 1, under identical training configurations, the block-level formulation becomes unstable and collapses earlier during training, while PBPO demonstrates substantially more stable optimization.

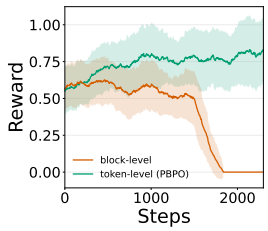

Figure 1: Reward dynamics on MATH500, trained upon LLaDA-8B-Instruct. PBPO demonstrates stable training, while block-level objective collapses early during training.

Theorem 1 (Blockwise Importance Weighting Produces Noisier Gradient Contributions)**.** *Given a question q, and a group of G rollouts $\{o_i\}_{i=1}^G \sim \pi_{\text{old}}(\cdot \mid q)$. For any b-th block of a rollout o_i, let the block-wise gradient contribution of the b-th block of o_i, $o_i^{(b-1)B_l+1:bB_l}$, in Equation 6 be*

$$g_{\text{block}} = \left[\prod_{k=(b-1)B_l+1}^{bB_l} \frac{\pi_\theta(o_i^k \mid q, z_i^{<b})}{\pi_{\text{old}}(o_i^k \mid q, z_i^{<b})}\right] \left[\sum_{k=(b-1)B_l+1}^{bB_l} \nabla_\theta \log \pi_\theta(o_i^k \mid q, z_i^{<b})\right], \tag{10}$$

and let the gradient contribution of the tokens $o_i^{(b-1)B_l+1:bB_l}$ in Equation 7 be

$$g_{\text{token}} = \sum_{k=(b-1)B_l+1}^{bB_l} \frac{\pi_\theta(o_i^k \mid q, z_i^{<b})}{\pi_{\text{old}}(o_i^k \mid q, z_i^{<b})} \nabla_\theta \log \pi_\theta(o_i^k \mid q, z_i^{<b}). \tag{11}$$

Under the additional assumptions stated in Appendix C, we have

$$\text{Var}\left[g_{\text{block}}\right] > \text{Var}\left[g_{\text{token}}\right]. \tag{12}$$

Thus, block-wise importance weighting in Equation 6 induces noisier gradient contributions than the token-wise importance weighting used by PBPO.

Table 1: Main results across six reasoning and coding benchmarks at generation lengths 128, 256, and 512. Best result per column within each backbone block is in **bold**, and the second-best is underlined.

	GSM8K			MATH500			HumanEval			MBPP			Countdown			Sudoku		
Baseline / Seq Len	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**
LLaDA-8B-Instruct	69.37	77.94	79.61	26.80	32.80	34.20	23.17	31.71	30.49	35.41	39.69	40.86	19.92	17.97	17.97	9.33	5.74	5.12
d1	72.55	78.01	81.20	29.00	34.40	37.60	29.88	35.37	**37.80**	41.25	42.41	41.63	30.08	23.44	30.08	21.29	14.98	9.76
wd1	73.01	80.06	81.50	32.80	34.80	36.20	27.44	34.15	37.20	42.80	42.02	42.80	48.83	51.17	49.22	24.40	24.16	22.97
GDPO	75.66	79.91	80.59	31.40	34.80	37.40	23.17	**37.20**	36.59	42.41	41.25	42.80	**62.50**	**73.05**	**73.05**	28.09	28.09	26.12
DMPO	75.59	80.74	82.79	29.80	35.40	38.20	24.39	35.98	**37.80**	43.19	43.58	43.58	50.00	57.81	64.84	21.15	17.13	15.50
PBPO (Ours)	**76.72**	**84.08**	**84.38**	**34.80**	**36.80**	**41.40**	**30.49**	**37.20**	**37.80**	**43.97**	**48.25**	**47.47**	52.73	64.06	64.45	**34.93**	**38.13**	**38.13**
LLaDA-1.5	71.27	79.38	82.11	28.40	32.20	38.40	21.95	30.49	34.76	36.19	44.36	38.91	21.09	18.36	21.09	13.78	10.10	8.33
d1	74.15	81.35	80.74	28.00	36.20	36.00	26.22	33.54	33.54	40.86	**46.69**	42.02	29.69	19.14	33.20	21.05	15.69	11.53
wd1	73.09	78.77	81.12	30.00	34.80	36.40	23.78	35.98	31.71	40.08	40.47	42.41	42.97	56.25	48.05	24.31	23.16	22.73
GDPO	74.60	78.39	79.38	30.20	34.40	38.40	23.78	36.59	**35.98**	38.91	45.91	41.25	51.56	67.58	**71.88**	25.50	26.70	24.26
DMPO	49.81	50.87	51.33	31.40	35.80	**39.00**	25.00	39.63	35.37	42.41	44.36	43.19	49.61	58.98	69.92	26.12	24.98	23.88
PBPO (Ours)	**78.24**	**83.47**	**84.61**	**33.40**	**42.00**	**39.00**	**31.71**	**41.46**	**35.98**	**42.80**	**46.69**	**44.36**	**57.03**	**69.14**	69.53	**37.75**	**39.33**	**36.17**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Datasets and evaluation metrics. We evaluate PBPO on six benchmarks spanning mathematical reasoning, planning, and coding. For (1) mathematical reasoning, we train on the GSM8K's training split and MATH dataset (Hendrycks et al., 2021) (without the subset of 500 problems from MATH500 (Lightman et al., 2023)) and evaluate on the GSM8K test set (Cobbe et al., 2021) and MATH500 (Lightman et al., 2023), respectively. For planning, we train on three-number TinyZero Countdown problems (Pan et al., 2025) and a 4×4 Sudoku split, then evaluate on 256 held-out Countdown problems and a solution-disjoint Sudoku set (Arel, 2025; Wang et al., 2025). For coding, we train on KodCode-Light-RL-10K (Xu et al., 2025) and evaluate on HumanEval's 164 problems (Chen et al., 2021) and the 257 tasks in sanitized MBPP (Austin et al., 2021). We report accuracy using final-answer matching for GSM8K and MATH500, valid target-reaching expressions for Countdown, and complete-puzzle correctness for Sudoku; for HumanEval and MBPP, we report execution-based `pass@1`. Further dataset, split, and training reward details are provided in Appendix A.1.

Base models and baselines. We initialize RL training from two masked diffusion language models, **LLaDA-8B-Instruct** (Nie et al., 2025) and **LLaDA-1.5** (Zhu et al., 2026). We compare PBPO against recent approaches for RL post-training of dLLMs, including d1 (Zhao et al., 2025), wd1 (Tang et al., 2025), GDPO (Rojas et al., 2025), and DMPO (Zhu et al., 2025).

Training setup. We perform task-specific RL post-training and train a separate policy for each benchmark and initialization checkpoint. Following d1 (Zhao et al., 2025), we use Low-Rank Adaptation (LoRA) (Hu et al., 2022) for parameter-efficient optimization. Unless otherwise specified, all training runs share the same rollout generation process. We use confidence-based block-wise decoding proposed in LLaDA (Nie et al., 2025), which unmasks the two positions with the highest confidence within the current block at each denoising step. Complete training configurations, rollout generation details, and hyperparameters are provided in Appendix A.2.

Evaluation protocol. For each RL run, we evaluate the *final saved checkpoint* after the prescribed number of optimization steps, with GDPO as the only exception. As GDPO collapses early during training, we evaluate its last saved checkpoint immediately before the collapse. Following prior dLLM RL works (Zhao et al., 2025; Tang et al., 2025), we report results at maximum completion lengths of 128, 256, and 512. For a maximum generation length of N, we use $N/2$ diffusion steps and block length $B_l = 32$, utilizing LLaDA's confidence-based unmasking strategy, as described above. During evaluation, we use temperature 0.0 for deterministic decoding.

4.2 MAIN RESULTS

PBPO achieves competitive performance. In Tables 1, across all configurations, PBPO outperforms d1, wd1, GDPO, and DMPO in 27 settings and ties the leading baseline in five more. In the other four settings, PBPO remains competitive, ranking second or third. This performance

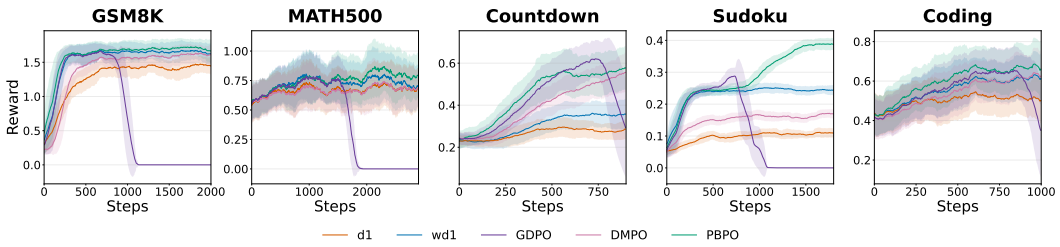

Figure 2: Reward dynamics during five RL post-training runs initialized from LLaDA-8B-Instruct. We report the mean and standard deviation over a rolling window of 50 training steps.

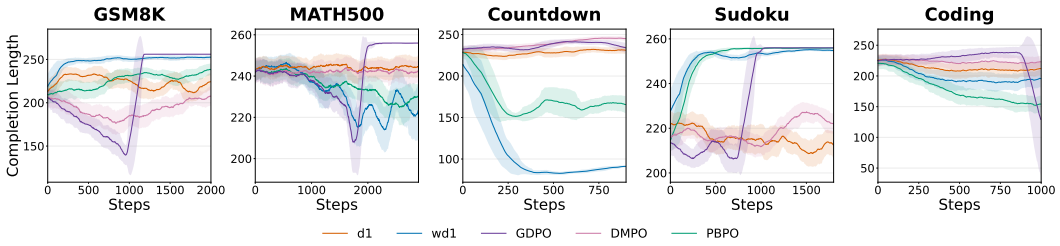

Figure 3: Completion length dynamics during five RL post-training runs initialized from LLaDA-8B-Instruct. We report the mean and standard deviation over a rolling window of 50 training steps.

is consistent across both model initializations and task domains: PBPO leads or ties throughout GSM8K, MATH500, Sudoku, HumanEval, and MBPP, while remaining competitive on Countdown. Averaged across the three generation lengths, PBPO's margins over the runner-up on GSM8K, MATH500, Sudoku, HumanEval, and MBPP are 2.00, 2.20, 9.63, 0.20, and 3.11 percentage points, respectively, with LLaDA-8B-Instruct. With LLaDA-1.5, the corresponding margins are 3.08, 2.60, 12.06, 2.44, and 0.52 percentage points. Qualitative PBPO generations from both model initializations are provided in Appendix D.

Training dynamics. Figures 2 and 4 show that PBPO improves steadily and maintains stable optimization throughout training with both model initializations. It finishes with the highest reward across all five training runs, complementing its competitive final evaluation performance in Table 1.

Effective utilization of the generation budget. Figures 3 and 5 show that PBPO couples improving reward with effective use of the generation budget. Although PBPO converges to the highest completion-length regime alongside wd1 and GDPO on Sudoku, it achieves substantially higher reward and evaluation accuracy than both. Tables 5 and 6 reinforce this efficiency at evaluation time: PBPO pairs strong accuracy with compact effective responses, ranking as the shortest or second-shortest method in 29 configurations. On the coding benchmarks, it produces the shortest responses in 11 of 12 settings and the second-shortest in the remaining one while attaining the highest `pass@1` throughout. These results show that PBPO yields strong task performance with effective generation.

Robustness across various denoising configurations. We further vary the block length B_l and evaluate both confidence-based and random token-selection strategies, which were proposed in LLaDA (Nie et al., 2025). Complete configuration details and empirical results are provided in Appendix B.3. Across the 30 combinations of six benchmarks and five denoising configurations, PBPO achieves the best result in 20, ties the leading baseline in two, ranks second in five, and ranks third in three. PBPO achieves the best result on GSM8K and Sudoku under every configuration, while the two confidence-based block-length variants nearly preserve its default mean accuracy across all six benchmarks (53.22% versus 53.68%). Under random token selection, PBPO ranks first or second in 17 of 18 comparisons. Its effective sequence length also ranks first or second in 20 configurations, demonstrating robust task performance together with effective generation across various decoding configurations.

4.3 Ablation Study

We next examine the effect of the conditioning diversification strategy introduced in Section 3.3.1. We compare three ablations: (1) PBPO with a clean prompt and clean prefix; (2) PBPO with per-

Table 2: Ablation across three PBPO variants on Countdown and MATH500 at generation lengths 128, 256, and 512, with LLaDA-1.5 as the base model. Best result per column is in **bold**, and second-best result is highlighted with underline.

	Countdown			MATH500		
Baseline / Seq Len	**128**	**256**	**512**	**128**	**256**	**512**
PBPO + clean prompt + clean prefix	46.09	47.27	51.56	30.80	32.60	36.20
PBPO + prompt masking + clean prefix	47.27	57.81	62.89	**33.40**	**42.00**	**39.00**
PBPO + prompt masking + prefix masking	**57.03**	**69.14**	**69.53**	31.00	35.60	38.20

update prompt masking but a clean prefix; and (3) PBPO with per-update masking of both the prompt and revealed prefix. For the masked variants, each eligible token is independently replaced by `[MASK]` with probability 0.15. We conduct this ablation on MATH500 and Countdown using LLaDA-1.5 as the initialization checkpoint.

As shown in Table 2, introducing per-update corruption consistently improves over using a clean prompt and clean prefix across both benchmarks and all three generation lengths. This supports the motivation in Section 3.3.1, as exposing the dLLM to multiple corrupted conditioning views provides a more effective way to reuse a fixed rollout batch during policy optimization. The effect of additionally masking the revealed prefix is task-dependent. On Countdown, prompt-and-prefix masking substantially outperforms prompt-only masking, improving accuracy by 9.76, 11.33, and 6.64 percentage points at generation lengths 128, 256, and 512, respectively. On MATH500, prompt-only masking performs better than additionally masking the prefix, although both masked variants outperform the fully clean configuration. These results suggest that conditioning diversification is consistently beneficial, while the preferred amount of corruption in the revealed prefix can depend on the reasoning task.

5 Related Works

Masked diffusion language models. Masked diffusion language models specialize discrete diffusion to absorbing-state corruption, with training objectives that reduce to weighted denoising losses over masked positions (Shi et al., 2024; Sahoo et al., 2024). RADD further connects absorbing diffusion to clean-data conditional prediction and any-order autoregressive modeling (Ou et al., 2025). Building on these foundations, LLaDA and LLaDA-1.5 scale masked diffusion to large language models and use block-wise decoding to combine parallel denoising with dependencies across completed blocks (Nie et al., 2025; Zhu et al., 2026). PBPO focuses on RL post-training under this block-wise generation structure.

Policy optimization for masked dLLMs. Because exact response likelihoods are intractable, existing methods adopt different optimization surrogates. d1 applies group-relative policy optimization using one-step token predictions under a mean-field approximation (Zhao et al., 2025). wd1 instead avoids explicit importance ratios and optimizes a reward-weighted token likelihood (Tang et al., 2025). GDPO constructs a sequence-level ratio from a diffusion evidence lower bound and uses structured estimation to manage its cost and variance (Rojas et al., 2025), whereas DMPO replaces clipped ratio optimization with matching to a reward-modified target distribution (Zhu et al., 2025). PBPO retains clipped surrogate ratios and addresses the distinct question of how these ratios should be assigned within the block-wise structure.

6 Conclusion

We propose Prefix-Block Policy Optimization (PBPO) for RL post-training of dLLMs. Starting from the block-level adaptation of autoregressive policy optimization, PBPO approximates intractable completed-block likelihoods with factorized one-step predictions from a prefix-block state. Whereas the block-level objective multiplies the token surrogate ratios into one shared weight, PBPO retains the same prefix-block conditioning while applying surrogate importance weighting and clipping separately to each token, yielding more stable training. Across two LLaDA-family initializations, six benchmarks, and varied denoising configurations, PBPO achieves competitive performance against existing dLLM RL methods while maintaining stable reward improvement. PBPO still relies on a one-step approximation to the exact multi-step denoising likelihood; extending the formulation to broader dLLM families and decoding procedures is therefore an important direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to polish writing. We have not used generative AI tools for generating research ideas, retrieval and discovery, or generate synthetic datasets. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper did not involve human subjects, practices to data set releases, potentially harmful insights, methodologies and applications, potential conflicts of interest and sponsorship, discrimination/bias/fairness concerns, privacy and security issues, legal compliance, and research integrity issues.

REPRODUCIBILITY STATEMENT

The necessary implementation details are listed in Section 4.1 and Appendix A. The authors will release the source code upon paper publication.

REFERENCES

Arel. Arel's sudoku generator. https://www.ocf.berkeley.edu/~arel/sudoku/main.html, 2025.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

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. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/ARXIV.2501.12948. URL https://doi.org/10.48550/arXiv.2501.12948.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *CoRR*, abs/2503.20783, 2025. doi: 10.48550/ARXIV.2503.20783. URL https://doi.org/10.48550/arXiv.2503.20783.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

Jingyang Ou, Shen Nie, Kaiwen Xue, Fengqi Zhu, Jiacheng Sun, Zhenguo Li, and Chongxuan Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=sMyXP8Tanm.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

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

Kevin Rojas, Jiahe Lin, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, Molei Tao, and Wei Deng. Improving reasoning for diffusion language models via group diffusion policy optimization. *CoRR*, abs/2510.08554, 2025. doi: 10.48550/ARXIV.2510.08554. URL https://doi.org/10.48550/arXiv.2510.08554.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander M. Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL https://doi.org/10.48550/arXiv.2402.03300.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and generalized masked diffusion for discrete data. In *Advances in Neural Information Processing Systems*, 2024.

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. wd1: Weighted policy optimization for reasoning in diffusion language models. *CoRR*, abs/2507.08838, 2025. doi: 10.48550/ARXIV.2507.08838. URL https://doi.org/10.48550/arXiv.2507.08838.

Chenyu Wang, Paria Rashidinejad, DiJia Su, Song Jiang, Sid Wang, Siyan Zhao, Cai Zhou, Shannon Zejiang Shen, Feiyu Chen, Tommi S. Jaakkola, Yuandong Tian, and Bo Liu. SPG: sandwiched policy gradient for masked diffusion language models. *CoRR*, abs/2510.09541, 2025. doi: 10.48550/ARXIV.2510.09541. URL https://doi.org/10.48550/arXiv.2510.09541.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 6980–7008. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-ACL.365. URL https://doi.org/10.18653/v1/2025.findings-acl.365.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: an open-source LLM reinforcement learning system at scale. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/52190a0362148d179f1cbd9080956872-Abstract-Conference.html.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 11425–11460. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.ACL-LONG.524. URL https://doi.org/10.18653/v1/2026.acl-long.524.

Yuchen Zhu, Wei Guo, Jaemoo Choi, Petr Molodyk, Bo Yuan, Molei Tao, and Yongxin Chen. Enhancing reasoning for diffusion llms via distribution matching policy optimization. *CoRR*, abs/2510.08233, 2025. doi: 10.48550/ARXIV.2510.08233. URL https://doi.org/10.48550/arXiv.2510.08233.

A Additional Experimental Details

A.1 Datasets and Reward Functions

We follow the task-specific verifiable rewards used in prior dLLM RL work (Zhao et al., 2025; Tang et al., 2025), while specifying the exact parsing and scoring rules used in our implementation. All reward functions operate on the decoded response string o. When a task uses M reward components, the sequence-level reward is their equal-weight sum,

$$R(q, o) = \sum_{m=1}^{M} R_m(q, o), \tag{13}$$

where R_m denotes the m-th task-specific component. Countdown and Sudoku use a single component. PBPO subsequently converts $R(q, o)$ into the group-relative advantage in Section 2.2; no additional reward normalization is applied.

GSM8K. We perform RL training on the standard GSM8K training split and evaluate on the held-out test split. Let $a(o)$ be the whitespace-trimmed substring after the final `<answer>` tag and before the next `</answer>` tag. If an opening answer tag is absent, the parser falls back to the response content preceding a closing answer tag, if present. The ground-truth answer y is the trimmed substring following the `####` delimiter in the GSM8K reference solution. The training reward is

$$R_{\text{GSM8K}}(q, o) = R_{\text{XML}}(o) + R_{\text{soft}}(o) + R_{\text{strict}}(o) + R_{\text{int}}(o) + R_{\text{correct}}(o, y), \tag{14}$$

with the following five components:

- **XML structure.** R_{XML} adds 0.125 when each of the following literal delimiters occurs exactly once: `<reasoning>\n`, `\n</reasoning>\n`, `\n<answer>\n`, and `\n</answer>`. The trailing-content adjustment follows the implementation literally. If `\n<answer>\n` occurs exactly once, it subtracts 0.001 times the number of characters returned after splitting on `\n</answer>\n`; if that closing delimiter is absent, this operation returns the full response. A second check, activated when `\n</answer>` occurs exactly once, subtracts $0.001(L-1)$, where L is the length of the suffix after that delimiter. This operation does not verify that the omitted character is a newline: it adds 0.001 when the suffix is empty and otherwise ignores one trailing character. The two adjustments are added when both checks apply.
- **Soft format.** $R_{\text{soft}} = 0.5$ when the response matches the code-like pattern `<reasoning>...</reasoning>\s*<answer>...</answer>` from the beginning of the string, and is zero otherwise. Because the regular expression does not enable multiline matching, the content inside each tag pair must lie on a single line.
- **Strict format.** $R_{\text{strict}} = 0.5$ only for the exact layout `<reasoning>\n...\n</reasoning>` followed by `\n<answer>\n...\n</answer>\n`; otherwise, it is zero. Here, each ellipsis again matches only single-line content. Consequently, the soft and strict format indicators cannot both be active for the same response under these regular expressions.
- **Integer answer.** $R_{\text{int}} = 0.5$ when every character of $a(o)$ is a decimal digit, and is zero otherwise.
- **Answer correctness.** $R_{\text{correct}}(o, y) = 2\,\mathbf{1}[a(o) = y]$, where equality is an exact string comparison after trimming surrounding whitespace.

Because the soft and strict patterns are mutually exclusive, the largest executed reward is 3.5, attained by a correct integer answer satisfying the strict layout and all four XML checks. Trailing content can further reduce the XML component. Evaluation reports the percentage of examples for which the extracted final answer exactly matches the ground-truth answer.

MATH500. For RL training, we load the training split of the MATH500 dataset and evaluate on its held-out split. We first extract the final `\boxed{}` expression from the response, or the final `\fbox{}` expression when no `\boxed` expression occurs; if neither is found, the full response is

retained as the candidate. We remove the outer box and apply the same procedure to the reference solution. The candidate and reference are then normalized using a deterministic string procedure. Specifically, the normalizer removes newline characters and literal spaces, inverse spacing `\!`, `\left` and `\right`, degree markers, escaped dollar and percentage symbols, and all content beginning at the exact substring `\text{` when that substring occurs once. If it occurs multiple times, normalization fails and the comparator falls back to raw-string equality. The procedure also canonicalizes doubled backslashes, `\tfrac`, and `\dfrac`; repairs simple fraction and square-root syntax; converts a simple a/b expression and 0.5 to fractional notation; normalizes leading decimals; and drops a variable name of at most two characters preceding a single equality. Let $\mathcal{N}(o)$ and $\mathcal{N}(y)$ denote the resulting normalized candidate and reference strings, respectively.

For the format component, let $h_{\text{ans}}(o)$ indicate that the literal opening tag `<answer>` occurs; a closing tag is not required by the implemented split operation. Moreover, let $h_{\text{box}}(o)$ indicate the literal control-character sequence tested by the implementation, within the extracted answer content when $h_{\text{ans}}(o) = 1$ and within the full response otherwise. This sequence is `U+0008oxed` (a backspace followed by `oxed`), rather than the printed LaTeX command `\boxed`; consequently, $h_{\text{box}}(o) = 0$ for ordinary model outputs containing `\boxed`. The executed reward is

$$R_{\text{MATH}}(q, o) = 2\,\mathbf{1}[\mathcal{N}(o) = \mathcal{N}(y)] + 0.25 + 0.5h_{\text{ans}}(o) + 0.25h_{\text{box}}(o). \tag{15}$$

Accordingly, correctness is determined by equality after deterministic string normalization rather than by a computer-algebra solver. For ordinary decoded text, the format component is 0.75 when `<answer>` is present and 0.25 otherwise, giving a maximum attainable reward of 2.75; the literal program maximum is 3.0 if the control-character condition is met. Evaluation applies the same normalized-answer comparison.

Countdown. We use the training split of the Countdown dataset from TinyZero (Pan et al., 2025), restricted to problems containing three input numbers. After shuffling with the experiment seed, we reserve the final 500 examples and train on the remainder. Following d1 (Zhao et al., 2025), we evaluate on a fixed set of 256 synthetically generated three-number Countdown problems. The parser extracts the final substring enclosed by `<answer>` and `</answer>`. Let $e(o)$ denote this expression, $T \in \mathbb{R}$ the target, and $\mathcal{N}_q$ the multiset of three input numbers. The verifier extracts every maximal digit sequence in $e(o)$, converts each sequence to an integer, and requires the resulting multiset to equal $\mathcal{N}_q$. It then evaluates expressions containing only digits, whitespace, decimal points, parentheses, and the operators $+$, $-$, $*$, and $/$ in an environment without Python built-ins. The reward used in our runs is

$$R_{\text{Countdown}}(q, o) = \begin{cases} 1, & \text{if an expression is extracted, uses exactly } \mathcal{N}_q, \\ & \text{evaluates successfully, and differs from } T \text{ by less than } 10^{-5}, \\ 0, & \text{if no answer expression can be extracted,} \\ 0.1, & \text{otherwise.} \end{cases} \tag{16}$$

Thus, the 0.1 fallback also covers an extracted expression with an invalid number multiset, unsupported characters, an evaluation error, or an incorrect value; it is not restricted to well-formed expressions that merely miss the target.

Sudoku. We use the 4×4 Sudoku dataset generated by Arel (Arel, 2025), whose original training set contains one million unique puzzles spanning all 288 possible 4×4 solution grids. Following (Wang et al., 2025), we construct a solution-disjoint train–test split to prevent leakage through memorization of complete solutions. Specifically, we randomly select 200 solution grids and include all puzzles associated with them in the training pool, resulting in $694{,}006$ puzzles. The loader shuffles this pool with the experiment seed, reserves its final 500 examples, and therefore uses $693{,}506$ puzzles for RL training. The test set contains 256 puzzles sampled from the remaining 88 solution grids, ensuring that no test solution appears in the RL training set.

For a puzzle string $p \in \{0, 1, 2, 3, 4\}^{16}$, let $g \in \{1, 2, 3, 4\}^{16}$ denote its ground-truth solution and $\mathcal{E}(p) = \{j \in \{1, \ldots, 16\} : p_j = 0\}$ the indices of originally empty cells. The parser takes the final answer-tag span, removes every non-digit character, and obtains a candidate digit string $s(o)$. Candidates shorter than 16 digits are right-padded with zeros, whereas candidates longer than 16 digits are truncated; denote the resulting length-16 string by $\bar{s}(o)$. If no nonempty answer can be extracted, the reward is zero. Otherwise,

$$R_{\text{Sudoku}}(q, o) = \frac{1}{|\mathcal{E}(p)|} \sum_{j \in \mathcal{E}(p)} \mathbf{1}[\bar{s}_j(o) = g_j]\,. \tag{17}$$

Only originally empty cells contribute: copied clues receive no credit, and there is no separate reward for satisfying row, column, or subgrid constraints. Evaluation is stricter than the dense training reward and reports the fraction of puzzles for which the complete solution is correct.

Code generation. We perform coding RL on the training portion of KodCode-Light-RL-10K (Xu et al., 2025). Following a deterministic split with seed 42, we use 90% of its training split; each example provides a coding problem, a reference solution, and synthetic unit tests. The coding reward combines the XML-structure component defined for GSM8K with a code-specific component that evaluates both output placement and functional correctness.

More precisely, let $c(o)$ denote the program extracted from the first literal code block matching ```` ```python\n...``` ```` in response o. Let $h_{\text{place}}(o)$ indicate that this entire code block occurs inside the first `<answer>...</answer>` span. For a valid KodCode test specification, let $\mathcal{T} = \{\tau_j\}_{j=1}^{J}$ denote the J unit tests associated with the problem, and let $\text{Pass}(c(o), \tau_j) \in \{0, 1\}$ indicate whether executing $c(o)$ with test τ_j terminates successfully. The resulting reward is

$$R_{\text{Code}}(q, o) = R_{\text{XML}}(o) + 0.5\, h_{\text{place}}(o) + \frac{1}{J} \sum_{j=1}^{J} \text{Pass}(c(o), \tau_j), \tag{18}$$

where the test-pass fraction is defined as zero when no executable test is available. Thus, a correctly formatted response whose program passes every test receives a maximum reward of 2.0: up to 0.5 from XML structure, 0.5 from placing the Python code block inside the answer tags, and 1.0 from functional correctness.

Before execution, the verifier identifies the expected function name from a literal statement of the form `from solution import <name>` or, otherwise, from the first assertion that invokes a function. It extracts the first function defined by the generated program and renames that definition to the expected name when necessary. The verifier then rejects programs containing any blocked module or keyword from `os`, `sys`, `subprocess`, `shutil`, `socket`, `psutil`, `ctypes`, `pathlib`, `builtins`, and `__import__`. Unsafe programs and outputs without a function definition receive no functional-correctness credit, although a correctly placed code block retains its 0.5 placement reward. Finally, each unit test is executed in a separate worker process with a 1-GB memory limit and a one-second deadline; exceptions, timeouts, and missing test results are counted as failures. During evaluation on HumanEval and MBPP, we instead report the stricter execution-based `pass@1` metric described in Section 4.1.

A.2 Training and Implementation Details

Here, we provide the complete rollout, optimization, and implementation configurations required for reproducibility.

Rollout generation. For each prompt, we sample $G = 6$ responses with maximum completion length 256 and temperature 0.9. Following LLaDA (Nie et al., 2025) and prior dLLM RL work (Zhao et al., 2025; Tang et al., 2025), generation proceeds block-by-block with block length $B_l = 32$ over 128 denoising steps. At each step, the two currently masked positions with the highest predicted-token confidence within the active block are revealed, while all remaining positions in that block stay masked. Generation advances to the next block after the active block is fully revealed. Each resulting rollout batch is reused for $\mu = 4$ policy-update iterations.

We use a per-device training batch size of 6 with 2 gradient accumulation steps. Every training run is conducted on 8 NVIDIA H200 GPUs.

Following d1 (Zhao et al., 2025), we use LoRA with rank $r = 128$, scaling factor $\alpha = 64$, and dropout 0.05. LoRA adapters are applied to the query, key, value, and output projections in the attention layers, as well as the up, down, and gate projections in the feed-forward layers. We optimize the LoRA parameters using AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.99$, weight decay 0.1, learning rate 3×10^{-6}, and maximum gradient norm 0.2. The learning rate follows a `constant_with_warmup` schedule with warmup ratio 10^{-4}.

For PBPO with per-update masking, the clean prompt, clean revealed prefixes, generated responses, and behavior policy π_{old} remain fixed while a rollout batch is reused. At each policy-update iteration,

Table 3: Dataset-specific PBPO training hyperparameters for policies initialized from LLaDA-8B-Instruct.

Hyperparameter	GSM8K	MATH500	Countdown	Sudoku	KodCode-Light-RL-10K
Training steps	2,000	2,900	900	1,800	1,000
Prompt masking probability p_q	0.15	0.15	0.15	0.15	0.15
Prefix masking probability p_z	0	0	0.15	0	0.15
ϵ_l	0.5	0.2	0.2	0.2	0.2
ϵ_r	0.5	0.28	0.28	0.28	0.28

Table 4: Dataset-specific PBPO training hyperparameters for policies initialized from LLaDA-1.5.

Hyperparameter	GSM8K	MATH500	Countdown	Sudoku	KodCode-Light-RL-10K
Training steps	2,200	2,400	1,000	1,900	1,000
Prompt masking probability p_q	0.15	0.15	0.15	0.15	0.15
Prefix masking probability p_z	0	0	0.15	0	0.15
ϵ_l	0.2	0.2	0.2	0.2	0.2
ϵ_r	0.28	0.28	0.28	0.28	0.28

a fresh masking pattern is independently sampled and applied to the same clean prompt and prefix sequences, as described in Section 3.3.1.

The optimization, batching, rollout, and LoRA settings are shared across all experiments. Tables 3 and 4 report the dataset-specific hyperparameters separately for policies initialized from LLaDA-8B-Instruct and LLaDA-1.5, respectively.

B Additional Empirical Results

B.1 Training Dynamics with LLaDA-1.5

Figures 4 and 5 provide the reward and completion length trajectories for the five training runs when PBPO and the baselines are initialized from LLaDA-1.5.

B.2 Effective Sequence Length Across Benchmarks

Reasoning benchmarks. Table 5 reports the average effective sequence length of the checkpoints evaluated in Table 1 across the four reasoning benchmarks.

Coding benchmarks. Table 6 reports the corresponding average effective sequence lengths on HumanEval and MBPP for both initialization checkpoints.

B.3 Robustness Across Various Evaluation Configurations

We further evaluate LLaDA-1.5 and its RL-post-trained variants under five additional decoding configurations. All configurations use a maximum generation length of 256 and 128 denoising steps, while varying the block length B_l and token-selection strategy. For each configuration, we report task accuracy and average effective sequence length across GSM8K, MATH500, Countdown, Sudoku, HumanEval, and MBPP.

Low-confidence selection with $B_l = 16$. Tables 7 and 8 report accuracy and average effective sequence length, respectively.

Low-confidence selection with $B_l = 64$. Tables 9 and 10 report accuracy and average effective sequence length, respectively.

Random selection with $B_l = 16$. Tables 11 and 12 report accuracy and average effective sequence length, respectively.

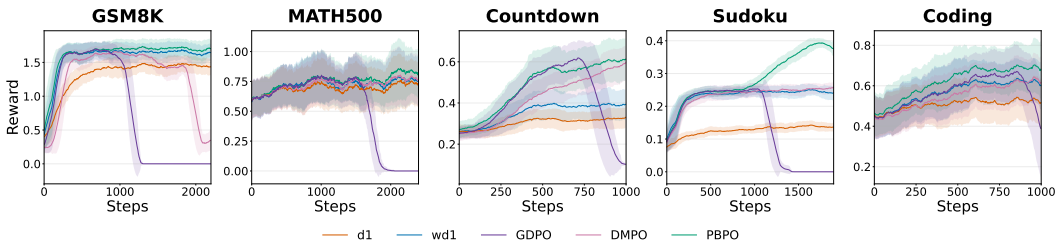

Figure 4: Reward dynamics during five RL post-training runs initialized from LLaDA-1.5. We report the mean and standard deviation over a rolling window of 50 training steps.

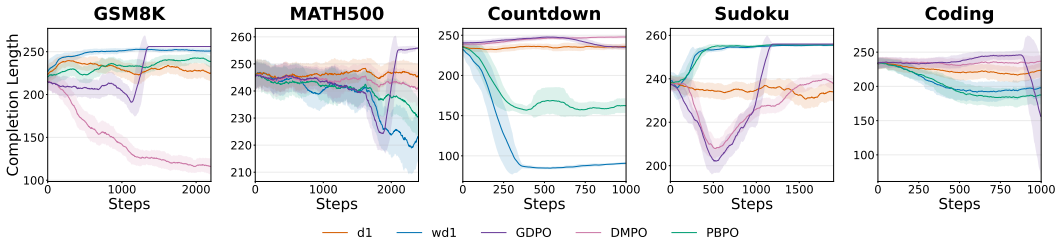

Figure 5: Completion length dynamics during five RL post-training runs initialized from LLaDA-1.5. We report the mean and standard deviation over a rolling window of 50 training steps.

Random selection with $B_l = 32$. Tables 13 and 14 report accuracy and average effective sequence length, respectively.

Random selection with $B_l = 64$. Tables 15 and 16 report accuracy and average effective sequence length, respectively.

Table 5: Effective sequence length for each baseline across four reasoning benchmarks at generation lengths 128, 256, and 512.

	GSM8K			MATH500			Countdown			Sudoku		
Baseline / Seq Len	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**	**128**	**256**	**512**
LLaDA-8B-Instruct	114.31	211.21	256.14	121.62	234.50	399.00	110.49	212.89	409.96	115.42	216.92	432.95
d1	115.85	217.13	275.15	122.05	234.47	392.75	107.49	209.75	397.20	115.54	225.42	435.74
wd1	114.25	226.42	326.66	120.21	209.63	319.50	76.34	80.60	85.30	111.13	236.70	465.87
GDPO	109.46	192.20	204.85	120.45	222.81	367.36	114.25	213.27	351.79	114.17	213.92	355.32
DMPO	114.22	213.28	258.28	121.78	234.64	395.15	110.21	212.17	364.04	114.11	228.31	243.07
PBPO (Ours)	113.78	221.08	281.70	121.62	226.61	359.10	103.37	134.61	159.16	102.94	164.98	244.54
LLaDA-1.5	115.15	214.51	263.16	123.23	237.43	405.48	113.36	214.66	412.64	115.17	222.07	434.49
d1	115.39	218.81	277.47	123.25	237.25	404.29	111.15	212.18	392.13	115.36	225.80	437.36
wd1	114.83	225.94	321.35	121.50	223.21	347.84	84.52	87.63	91.93	113.93	228.88	443.33
GDPO	114.39	213.37	252.97	122.54	234.22	386.49	115.04	217.39	339.80	112.47	220.21	423.35
DMPO	106.20	130.93	157.57	122.65	232.52	385.49	112.77	214.88	346.28	114.55	230.29	445.87
PBPO (Ours)	113.96	223.17	297.08	118.92	224.55	363.46	105.34	136.72	181.89	113.63	220.81	438.93

Table 6: Average effective tokens across two coding benchmarks at generation lengths 128, 256, and 512.

	HumanEval			MBPP		
Baseline / Seq Len	128	256	512	128	256	512
LLaDA-8B-Instruct	120.87	216.43	310.78	118.14	206.96	265.30
Diffu-GRPO	119.16	194.93	246.77	114.72	189.01	241.53
WD1 (WLL NP)	112.15	144.70	165.38	105.12	133.28	138.68
GDPO	116.87	223.54	393.85	115.93	209.94	330.08
DMPO (LLaDA sampler)	116.68	206.83	284.57	113.96	204.62	260.07
PBPO (Ours)	95.81	117.86	127.59	84.72	93.35	97.53
LLaDA-1.5	120.82	220.32	317.95	119.60	210.50	273.32
Diffu-GRPO	118.32	207.38	269.42	117.13	199.17	258.67
WD1 (WLL NP)	112.83	148.90	170.27	104.30	127.45	145.92
GDPO	118.35	229.54	427.01	118.48	222.93	384.49
DMPO (LLaDA sampler)	119.79	226.69	316.39	119.39	214.48	275.40
PBPO (Ours)	111.04	144.02	172.57	103.58	121.60	131.81

Table 7: Accuracy (%) across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 16, and low-confidence remasking. Best values in each benchmark column are in **bold**, and second-best distinct values are underlined.

Baseline	**GSM8K**	**MATH500**	**Countdown**	**Sudoku**	**HumanEval**	**MBPP**
LLaDA-1.5	79.08	33.40	16.02	10.96	32.32	45.53
d1	80.21	35.40	19.53	18.66	37.80	47.08
wd1	80.52	35.60	43.75	23.11	34.76	43.97
GDPO	79.76	36.00	69.14	24.64	35.37	**47.47**
DMPO	47.69	36.60	55.86	25.02	**41.46**	44.75
PBPO (Ours)	**84.38**	**38.40**	**73.83**	**38.09**	37.20	45.53

Table 8: Average effective sequence length in tokens across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 16, and low-confidence remasking.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	214.68	235.92	213.92	215.34	216.71	209.89
d1	218.30	236.67	209.41	224.58	203.32	199.69
wd1	225.21	223.35	87.38	231.79	150.53	132.07
GDPO	213.72	234.65	217.75	208.46	228.23	223.84
DMPO	136.23	233.55	214.05	229.50	226.51	218.67
PBPO (Ours)	222.81	224.18	147.56	219.10	145.73	121.81

Table 9: Accuracy (%) across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 64, and low-confidence remasking. Best values in each benchmark column are in **bold**, and second-best distinct values are underlined.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	80.44	35.80	21.48	7.70	32.32	43.97
d1	79.76	35.40	24.61	15.50	35.37	45.14
wd1	80.14	36.20	53.52	24.35	32.32	40.47
GDPO	78.17	33.20	**69.92**	25.17	34.15	44.75
DMPO	55.34	36.60	60.16	27.13	37.80	45.53
PBPO (Ours)	**83.47**	**41.00**	67.58	**40.33**	**40.24**	**48.64**

Table 10: Average effective sequence length in tokens across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 64, and low-confidence remasking.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	213.00	236.11	213.25	219.71	221.51	213.12
d1	218.31	236.60	211.13	226.96	211.35	201.00
wd1	226.24	224.47	80.70	229.16	145.35	120.46
GDPO	213.24	233.43	215.94	220.77	229.77	227.17
DMPO	129.48	234.84	215.18	229.09	225.58	211.99
PBPO (Ours)	223.12	226.33	123.18	222.70	145.60	121.19

Table 11: Accuracy (%) across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 16, and random remasking. Best values in each benchmark column are in **bold**, and second-best distinct values are underlined.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	70.43	25.80	10.55	9.67	22.56	30.35
d1	74.07	31.00	16.80	16.08	24.39	34.24
wd1	74.45	27.80	35.16	23.68	23.17	36.58
GDPO	76.27	31.20	57.42	23.30	**29.27**	36.19
DMPO	35.86	31.80	44.92	23.88	26.83	31.91
PBPO (Ours)	**79.38**	**33.00**	**64.06**	**36.60**	**29.27**	**37.74**

Table 12: Average effective sequence length in tokens across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 16, and random remasking.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	210.08	236.85	200.35	215.43	218.97	216.00
d1	213.42	233.93	188.89	213.16	208.24	192.67
wd1	225.39	220.64	80.82	227.92	144.90	131.38
GDPO	211.10	230.42	210.82	206.92	216.91	208.46
DMPO	223.87	229.37	206.87	216.55	219.80	214.87
PBPO (Ours)	222.92	221.82	164.72	220.76	148.30	128.40

Table 13: Accuracy (%) across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 32, and random remasking. Best values in each benchmark column are in **bold**, and second-best distinct values are underlined.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	67.55	24.20	9.77	11.29	13.41	24.90
d1	70.13	25.60	18.36	15.79	**19.51**	26.46
wd1	73.84	24.80	37.11	22.87	**19.51**	**28.40**
GDPO	72.86	28.00	**57.81**	24.78	**19.51**	**28.40**
DMPO	33.51	28.80	45.70	24.11	17.68	25.68
PBPO (Ours)	**76.88**	**29.40**	50.39	**35.74**	**19.51**	26.46

Table 14: Average effective sequence length in tokens across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 32, and random remasking.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	208.34	236.18	194.70	216.06	221.29	212.11
d1	212.64	236.13	193.55	216.88	204.54	195.19
wd1	225.28	220.74	79.19	226.87	142.20	119.60
GDPO	211.06	233.40	206.23	204.16	219.20	211.19
DMPO	237.31	230.98	202.77	218.60	222.04	210.03
PBPO (Ours)	223.15	223.46	167.75	219.19	151.60	121.26

Table 15: Accuracy (%) across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 64, and random remasking. Best values in each benchmark column are in **bold**, and second-best distinct values are underlined.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	63.99	22.80	8.59	11.53	15.85	21.01
d1	66.26	25.80	20.31	17.42	14.02	20.62
wd1	68.31	25.40	35.55	23.92	**17.68**	28.02
GDPO	70.58	**29.20**	**60.16**	23.44	15.85	20.23
DMPO	29.04	27.60	45.70	24.35	12.80	24.51
PBPO (Ours)	**75.59**	26.00	51.56	**36.22**	15.85	**30.74**

Table 16: Average effective sequence length in tokens across six benchmarks for LLaDA-1.5 baselines with generation length 256, 128 diffusion steps, block length 64, and random remasking.

Baseline	GSM8K	MATH500	Countdown	Sudoku	HumanEval	MBPP
LLaDA-1.5	207.31	232.23	189.29	210.69	221.56	213.26
d1	213.74	236.24	184.30	212.71	197.27	195.75
wd1	226.47	219.60	71.83	227.71	135.05	114.01
GDPO	212.44	234.97	197.49	202.65	222.63	216.20
DMPO	259.03	234.39	197.01	231.22	218.78	214.25
PBPO (Ours)	223.22	225.75	130.59	219.43	148.45	126.32

C Proof for Theorem 1

This section presents the proof for Theorem 1. The comparison concerns the unclipped, advantage-free gradient contributions displayed in the theorem. Before presenting the proof, we first state the assumptions required for the variance comparison and clarify why they are appropriate for this analysis.

Assumptions. Fix the prompt q, rollout index i, block index b, and prefix-block state $z_i^{<b}$. We consider a full block with $B_l \geq 2$ and define its index set as

$$\mathcal{K}_b = \{(b-1)B_l + 1, \ldots, bB_l\}.$$

All expectations and covariances below are conditional on $(q, z_i^{<b})$ and are taken under the factorized one-step behavior surrogate

$$\prod_{k \in \mathcal{K}_b} \pi_{\text{old}}(o_i^k \mid q, z_i^{<b})$$

used in Equation 4. For each $k \in \mathcal{K}_b$, abbreviate the token surrogate ratio and score vector by

$$\rho_i^k = \frac{\pi_\theta(o_i^k \mid q, z_i^{<b})}{\pi_{\text{old}}(o_i^k \mid q, z_i^{<b})}, \qquad s_i^k = \nabla_\theta \log \pi_\theta(o_i^k \mid q, z_i^{<b}). \tag{19}$$

We assume that the pairs $\{(\rho_i^k, s_i^k)\}_{k \in \mathcal{K}_b}$ are conditionally independent and identically distributed. We further assume that π_θ and π_{old} have common positive support over the finite vocabulary $\mathcal{V}$, that π_θ is differentiable with parameter-independent support, and that the second moments used below are finite.

Because g_{block} and g_{token} are vectors, we interpret the variance in Theorem 1 as total variance:

$$\text{Var}[g] := \text{tr}(\text{Cov}(g)) = \mathbb{E}\left[\|g - \mathbb{E}[g]\|_2^2\right]. \tag{20}$$

By the identical-distribution assumption, the moments

$$m_2 := \mathbb{E}\left[(\rho_i^k)^2\right], \qquad H := \mathbb{E}\left[(\rho_i^k)^2 s_i^k (s_i^k)^\top\right], \qquad c := \mathbb{E}\left[(\rho_i^k)^2 s_i^k\right] \tag{21}$$

do not depend on $k \in \mathcal{K}_b$. For the strict comparison, we additionally assume that $\text{Var}[\rho_i^k] > 0$ and $H \neq 0$.

Remark 1 (Factorized surrogate assumptions)**.** *The conditional independence assumption follows from the factorized one-step block surrogate in Equation 4. It is an assumption about the surrogate probability space used in this comparison, not a claim that tokens produced by the true multi-step denoising trajectory are independent. Identical ratio-score moments across positions are an additional homogeneity assumption that isolates the effect of importance-weighting granularity.*

Remark 2 (Regularity and strictness)**.** *For standard full-softmax policies, all vocabulary tokens have positive probability, so the common-support condition is satisfied. With a finite vocabulary and finite score vectors, the required moments are also finite. The positive-ratio-variance and nonzero-score conditions characterize a nontrivial policy update and are necessary for strictness: if $\pi_\theta = \pi_{\text{old}}$, or if $B_l = 1$, the two contributions can coincide. A partial final block follows by replacing B_l with its actual number of positions, provided that this number is at least two.*

Remark 3 (Advantage and clipping)**.** *Theorem 1 compares the unclipped, advantage-free contributions displayed in its statement. Multiplying both contributions by the same fixed nonzero advantage A_i scales their covariance matrices by A_i^2 and preserves the strict comparison. PPO-style clipping introduces a piecewise dependence on the surrogate ratios, so the covariance identities below do not directly characterize the complete clipped-objective gradients.*

Having stated and justified the assumptions, we now prove the variance comparison.

Proof. The common-support assumption and normalization of the token distributions give

$$\mathbb{E}[\rho_i^k] = \sum_{v \in \mathcal{V}} \pi_{\text{old}}(o_i^k = v \mid q, z_i^{<b}) \frac{\pi_\theta(o_i^k = v \mid q, z_i^{<b})}{\pi_{\text{old}}(o_i^k = v \mid q, z_i^{<b})} \tag{22}$$

$$= \sum_{v \in \mathcal{V}} \pi_\theta(o_i^k = v \mid q, z_i^{<b}) = 1. \tag{23}$$

Differentiability and parameter-independent support similarly yield the importance-weighted score identity

$$\mathbb{E}[\rho_i^k s_i^k] = \sum_{v\in\mathcal{V}} \pi_\theta(o_i^k = v \mid q, z_i^{<b}) \nabla_\theta \log \pi_\theta(o_i^k = v \mid q, z_i^{<b}) \tag{24}$$

$$= \nabla_\theta \sum_{v\in\mathcal{V}} \pi_\theta(o_i^k = v \mid q, z_i^{<b}) = 0. \tag{25}$$

Equation 23 also implies

$$m_2 = 1 + \mathrm{Var}[\rho_i^k] \geq 1. \tag{26}$$

Using Equation 19, the token-level contribution in Theorem 1 is

$$g_{\text{token}} = \sum_{k\in\mathcal{K}_b} \rho_i^k s_i^k.$$

Equation 25 gives $\mathbb{E}[g_{\text{token}}] = 0$. Moreover, conditional independence implies that, for distinct $k, \ell \in \mathcal{K}_b$,

$$\mathbb{E}\left[\rho_i^k s_i^k (\rho_i^\ell s_i^\ell)^\top\right] = \mathbb{E}[\rho_i^k s_i^k]\mathbb{E}[\rho_i^\ell s_i^\ell]^\top = 0.$$

Consequently, all cross-position terms vanish and

$$\mathrm{Cov}(g_{\text{token}}) = \sum_{k\in\mathcal{K}_b} \mathbb{E}\left[(\rho_i^k)^2 s_i^k (s_i^k)^\top\right] = B_l H. \tag{27}$$

The block-level contribution can be written in the same notation as

$$g_{\text{block}} = \left(\prod_{j\in\mathcal{K}_b} \rho_i^j\right)\left(\sum_{k\in\mathcal{K}_b} s_i^k\right) \tag{28}$$

$$= \sum_{k\in\mathcal{K}_b} \rho_i^k s_i^k \prod_{\substack{j\in\mathcal{K}_b \\ j\neq k}} \rho_i^j. \tag{29}$$

By conditional independence, together with Equations 23 and 25,

$$\mathbb{E}[g_{\text{block}}] = \sum_{k\in\mathcal{K}_b} \mathbb{E}[\rho_i^k s_i^k] \prod_{\substack{j\in\mathcal{K}_b \\ j\neq k}} \mathbb{E}[\rho_i^j] = 0. \tag{30}$$

Its covariance therefore equals its second moment. Expanding that moment and separating the terms with matching and distinct token indices gives

$$\mathrm{Cov}(g_{\text{block}}) = \sum_{k\in\mathcal{K}_b} \mathbb{E}\left[(\rho_i^k)^2 s_i^k (s_i^k)^\top \prod_{\substack{j\in\mathcal{K}_b \\ j\neq k}} (\rho_i^j)^2\right] \tag{31}$$

$$+ \sum_{\substack{k,\ell\in\mathcal{K}_b \\ k\neq\ell}} \mathbb{E}\left[(\rho_i^k)^2 (\rho_i^\ell)^2 s_i^k (s_i^\ell)^\top \prod_{\substack{j\in\mathcal{K}_b \\ j\neq k,\ell}} (\rho_i^j)^2\right] \tag{32}$$

$$= B_l m_2^{B_l-1} H + B_l(B_l-1) m_2^{B_l-2} cc^\top. \tag{33}$$

Combining Equations 27 and 33 yields

$$\mathrm{Cov}(g_{\text{block}}) - \mathrm{Cov}(g_{\text{token}}) \tag{34}$$

$$= B_l\left(m_2^{B_l-1} - 1\right) H + B_l(B_l-1) m_2^{B_l-2} cc^\top. \tag{35}$$

Both H and $cc^\top$ are positive semidefinite. Since $m_2 \geq 1$ and $B_l \geq 2$, both coefficients on the right-hand side of Equation 35 are nonnegative. Thus,

$$\mathrm{Cov}(g_{\text{block}}) \succeq \mathrm{Cov}(g_{\text{token}}). \tag{36}$$

It remains to establish the strict scalar comparison. Since $\mathrm{Var}[\rho_i^k] > 0$, Equation 26 gives $m_2 > 1$. In addition, the assumed nonzero importance-weighted score second moment implies $H \neq 0$; because $H \succeq 0$, it follows that $\mathrm{tr}(H) > 0$. Taking the trace of Equation 35 therefore gives

$$\mathrm{Var}[g_{\mathrm{block}}] - \mathrm{Var}[g_{\mathrm{token}}] \tag{37}$$

$$= B_l \left(m_2^{B_l - 1} - 1 \right) \mathrm{tr}(H) + B_l(B_l - 1) m_2^{B_l - 2} \|c\|_2^2 > 0. \tag{38}$$

By the total-variance convention in Equation 20, this establishes the inequality in Theorem 1 under the assumptions stated above. □

D Qualitative Examples

We present representative outputs from PBPO initialized from LLaDA-1.5 and LLaDA-8B-Instruct. The responses are reproduced from the saved evaluation generations, with only special end-of-sequence and padding tokens removed.

D.1 MATH500 Examples

Model: LLaDA-1.5 + PBPO
Dataset: MATH500
Generation Sequence Length: 512
Question: What is the probability that a randomly selected integer in the set $\{1, 2, 3, \ldots, 100\}$ is divisible by 2 and not divisible by 3? Express your answer as a common fraction.

Model Response

<reasoning>
First, we determine the total number of integers in the set $\{1, 2, 3, \ldots, 100\}$, which is 100.
Next, we count the number of integers divisible by 2 in this set. The sequence of integers divisible by 2 is $2, 4, 6, \ldots, 100$. This is an arithmetic sequence with the first term $a = 2$ and common difference $d = 2$. The number of terms in this sequence is found by solving

$$a_n = a + (n-1)d \implies 100 = 2 + (n-1) \cdot 2 \implies 100 = 2n \implies n = 50.$$

So, there are 50 integers divisible by 2.
Now, we count the number of integers divisible by both 2 and 3 (i.e., divisible by 6). The sequence of integers divisible by 6 is $6, 12, 18, \ldots, 96$. This is an arithmetic sequence with the first term $a = 6$ and common difference $d = 6$. The number of terms in this sequence is found by solving

$$a_n = a + (n-1)d \implies 96 = 6 + (n-1) \cdot 6 \implies 96 = 6n \implies n = 16.$$

So, there are 16 integers divisible by 6 in the set.
The number of integers divisible by 2 but not divisible by 3 is the total number of integers divisible by 2 minus the number of integers divisible by 6:

$$50 - 16 = 34.$$

Thus, the probability that a randomly selected integer in the set $\{1, 2, 3, \ldots, 100\}$ is divisible by 2 and not divisible by 3 is

$$\frac{34}{100} = \frac{17}{50}.$$

</reasoning>
<answer>

$$\boxed{\frac{17}{50}}$$

</answer>

Model: LLaDA-8B-Instruct + PBPO
Dataset: MATH500
Generation Sequence Length: 512
Question: What is the distance, in units, between the points $(2, -6)$ and $(-4, 3)$? Express your answer in simplest radical form.

Model Response

<reasoning>
To find the distance between two points (x_1, y_1) and (x_2, y_2), we use the distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Given the points $(2, -6)$ and $(-4, 3)$, we substitute these coordinates into the formula:

$$d = \sqrt{((-4) - 2)^2 + (3 - (-6))^2}.$$

Simplify inside the parentheses:

$$= -6 \quad \text{and} \quad 3 + 6 = 9.$$

Square these results:

$$(-6)^2 = 36 \quad \text{and} \quad 9^2 = 81.$$

Add the squares:

$$36 + 81 = 117.$$

Take the square root:

$$d = \sqrt{117}.$$

Simplify $\sqrt{117}$:

$$\sqrt{117} = \sqrt{9 \times 13} = 3\sqrt{13}.$$

Thus, the distance between the points is $\boxed{3\sqrt{13}}$.
</reasoning>
<answer>

$$\boxed{3\sqrt{13}}$$

</answer>

D.2 MBPP EXAMPLES

Model: LLaDA-1.5 + PBPO
Dataset: MBPP
Generation Sequence Length: 512
Question: Write a Python function number is divisible by 11.
Tests: `is_Diff(12345) == False`, `is_Diff(1212112) == True`, and `is_Diff(1212) == False`.

Model Response

<reasoning>
To determine if a number is divisible by 11, we can use the simple divis for 11 rule. The rule states that a number is divisible by 11 if the difference between the sum of the digits in the odd positions and the sum of the digits in the even positions is either 0 or a multiple of 11.
</reasoning>
<answer>

```
def is_Diff(number):
  # Convert the number to a string to easily iterate over digits
  digits = str(number)
  # Initialize sums for odd and even positions
  odd_sum = 0
  even_sum = 0
  # Iterate over the digits of the number
  for i in range(len(digits)):
    if i % 2 == 0:
      even_sum += int(digits[i])
    else:
      odd_sum += int(digits[i])
  # Calculate the difference between the sums
  difference = odd_sum - even_sum
  # Check if the difference is 0 or a multiple of 11
  return difference == 0 or difference % 11 == 0
```

</answer>

Model: LLaDA-8B-Instruct + PBPO
Dataset: MBPP
Generation Sequence Length: 512
Question: Write a function to flatten a list and sum all of its elements.
Tests: `recursive_list_sum(([1,2,[3,4],[5,6])) == 21`, `recursive_list_sum(([7,10,[15,14],[19,41])) == 106`, and `recursive_list_sum(([10,20,[30,40],[50,60])) == 210`.

Model Response

<reasoning>
To flatten a list and sum all of its elements, we need to recursively traverse the list and accumulate the sum of its elements.
</reasoning>
<answer>

```
def recursive_list_sum(lst):
  total = 0
  for item in lst:
    if isinstance(item, list):
      total += recursive_list_sum(item)
    else:
      total += item
  return total
```

</answer>