AG-GRPO: Answer-Guided GRPO for Masked Diffusion Language Models

Juhyeong Kim*, **Gyunyeop Kim***, **Sangwoo Kang**†
Department of Computing, Gachon University
{stephano12, gyop0817, swkang}@gachon.ac.kr

Abstract

Reinforcement learning with verifiable rewards (RLVR) typically evaluates only final outcomes, providing limited learning signal about whether the generated reasoning is consistent with the correct answer. As a result, even when ground-truth answers are available during training, on-policy rollouts can repeatedly produce reasoning that is inconsistent with the answer. We propose Answer-Guided Group Relative Policy Optimization (AG-GRPO) for masked diffusion language models (dLLMs), which generate text through iterative masked-token restoration. AG-GRPO combines standard answer-free (AF) rollouts, sampled without access to the ground-truth answer, with answer-guided (AG) rollouts. In AG rollouts, the model generates reasoning conditioned on an anchored ground-truth answer suffix, and then re-predicts the answer from the generated reasoning for reward computation. We compute group-relative advantages over the combined AF/AG rollout set, allowing answer-guided training signals to improve the answer-free policy used at test time. Across mathematics, puzzle-solving, and code-generation benchmarks, AG-GRPO consistently improves over the pretrained dLLM and prior RL method for masked dLLMs. We further analyze optimization dynamics to study how shared group-relative advantages support signal transfer and affect convergence. Our code is available at `https://github.com/JuHyng/ag_grpo`.

1 Introduction

Recent work on large language models (LLMs) has emphasized not only final-answer accuracy but also the reliability of reasoning on challenging tasks such as mathematics, puzzles, and programming (Wei et al., 2023; Lightman et al., 2023; DeepSeek-AI et al., 2025). Reinforcement learning with verifiable rewards (RLVR) has become a widely used approach for improving reasoning-heavy behavior by assigning rewards through outcome-based verification, such as exact match, constraint satisfaction, or program execution (Shao et al., 2024; DeepSeek-AI et al., 2025; Schulman et al., 2017). However, RLVR typically evaluates only final outcomes. As a result, even when ground-truth answers are available during training, it provides limited signal about whether the generated reasoning is consistent with the correct answer, and on-policy rollouts can repeatedly sample similar failure modes early in training (Liu et al., 2025a).

A central challenge is that, under standard left-to-right autoregressive decoding, reasoning tokens cannot access future answer tokens during rollout generation. This limits how ground-truth answers can be incorporated into the sampling process itself. In contrast, masked diffusion language models (dLLMs) generate text through iterative masked-token restoration, allowing suffix information to influence prefix restoration (Nie et al., 2025). This makes masked dLLMs a natural setting for incorporating answer information directly into rollout generation.

In this work, we propose Answer-Guided Group Relative Policy Optimization (AG-GRPO) for masked dLLMs. AG-GRPO combines answer-free (AF) rollouts, which match the test-time condition, with answer-guided (AG) rollouts, which condition generation on an anchored ground-truth answer. To ensure that reward computation depends on the generated reasoning rather than the fixed suffix, AG rollouts re-predict the answer from the generated reasoning. AG-GRPO then computes group-relative advantages over a shared set of AF and AG rollouts, allowing answer-guided training signals to improve the AF policy used at test time.

Our contributions are as follows.

- We define answer-guided rollouts for masked dLLMs using suffix anchoring, followed by

*Equal contribution.
†Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 37175–37191
July 2-7, 2026

answer re-prediction for outcome-based verification.

- We propose AG-GRPO, which computes group-relative advantages over a shared set of answer-guided and answer-free rollouts to improve the answer-free policy used at test time.
- Experiments on mathematics, puzzle-solving, and code-generation tasks show that AG-GRPO improves performance across tasks, and further analyses examine its training dynamics and signal transfer.

2 Preliminaries

2.1 Masked Diffusion Language Model

A masked diffusion language model (masked dLLM) generates text by iteratively restoring masked tokens. Given a partially masked sequence as input, a mask predictor (denoiser) f_θ predicts token distributions at masked positions.

Let $\mathcal{V}$ denote the vocabulary and $\langle\text{mask}\rangle$ the mask token. Given a sequence $x \in (\mathcal{V} \cup \{\langle\text{mask}\rangle\})^{|x|}$ obtained by masking a subset of tokens from the original sequence x_0, the mask predictor outputs, at each position k, a categorical distribution over $\mathcal{V}$, denoted $f_\theta^k(\cdot \mid x)$.

Let $\mathcal{M}(x) := \{k \mid x^k = \langle\text{mask}\rangle\}$ be the set of masked positions. Training optimizes f_θ to reconstruct the original tokens at masked positions:

$$\mathcal{L}_{\text{diff}} = -\mathbb{E}_{x_0,x}\left[\sum_{k\in\mathcal{M}(x)} \log f_\theta^k\big(x_0^k \mid x\big)\right]. \quad (1)$$

At inference, the prompt is kept fixed, the generation segment is initialized as fully masked, and denoising is applied iteratively under a fixed schedule to produce the completion. More detailed formulation is provided in Appendix B.

2.2 Block-Wise Denoising Inference

Following Nie et al. (2025), we use block-wise denoising as the default inference procedure for long generations. The generation span is divided into multiple blocks, which are restored from left to right, yielding a semi-autoregressive decoding procedure.

Given a prompt q, generation length L, block length B, and total number of denoising steps S, the number of blocks is $M = L/B$, the number of steps per block is $T = S/M$, and the number of tokens restored per step is $N = L/S$. For block $m = 1, \ldots, M$, the index set is

$$\mathcal{I}_m = \big\{|q| + (m-1)B + 1, \ldots, |q| + mB\big\}. \quad (2)$$

Let $\tau = 1, \ldots, T$ index the denoising steps within block m, and let $x_{(m,\tau)}$ denote the full sequence state after step τ. Let $x_{(m,0)}$ be the state at the beginning of block m. Inference starts from

$$x_{(1,0)} = q \oplus \langle\text{mask}\rangle^L \quad (3)$$

and transitions across blocks as $x_{(m,0)} = x_{(m-1,T)}$.

At each step τ, the mask predictor computes the predicted token and its confidence at each position based on the previous state $x_{(m,\tau-1)}$:

$$\hat{x}^k_{(m,\tau)} = \arg\max_{v\in\mathcal{V}} f_\theta^k\big(v \mid x_{(m,\tau-1)}\big), \quad (4)$$

$$c^k_{(m,\tau)} = f_\theta^k\big(\hat{x}^k_{(m,\tau)} \mid x_{(m,\tau-1)}\big). \quad (5)$$

Among the still-masked positions in $\mathcal{I}_m$, we select the N positions with the highest confidence, denoted by $\mathcal{U}^{(m,\tau)}$, and update only those positions:

$$x^k_{(m,\tau)} = \begin{cases} \hat{x}^k_{(m,\tau)}, & k \in \mathcal{U}^{(m,\tau)}, \\ x^k_{(m,\tau-1)}, & \text{otherwise.} \end{cases} \quad (6)$$

Repeating this procedure over all blocks yields the final output $x_{(M,T)}$. Intuitively, generation proceeds left to right across blocks, while within each block tokens are restored over multiple discrete steps according to model confidence. In Section 3, we instantiate this default procedure differently for answer-free and answer-guided rollouts.

3 Answer-Guided GRPO for Masked dLLMs

In this section, we introduce Answer-Guided GRPO (AG-GRPO), which incorporates the ground-truth answer into rollout generation for masked dLLMs. Unlike standard RLVR, the answer is used not only for reward computation but also as a conditioning signal during generation.

AG-GRPO uses two rollout conditions for each prompt–answer pair: answer-free (AF) and answer-guided (AG). The AF condition matches test-time inference, whereas the AG condition conditions generation on an anchored ground-truth answer suffix. Each rollout condition produces a completion, and group-relative advantages are computed over the resulting shared AF/AG set so that relative signals from AG completions can improve the answer-free policy used at test time.

3.1 Rollout Sampling

We define two rollout conditions for each prompt–answer pair (q, y), where y is the ground-truth answer: answer-free (AF) and answer-guided (AG). Each rollout condition produces a completion. Completions from both rollout conditions are sampled from the same current rollout policy $\pi_{\theta_{\text{old}}}$.

Answer-Free (AF) completions. Under the AF rollout condition, completions are sampled from $\pi_{\theta_{\text{old}}}$ under the same condition as test time (i.e., without using the answer), and therefore directly reflect the final evaluation behavior:

$$o_{\text{AF}} \sim \pi_{\theta_{\text{old}}}\Big(\cdot \;\Big|\; q \oplus \langle\text{mask}\rangle^{L}\Big), \tag{7}$$

where the prompt q is kept fixed, the entire generation segment of length L is initialized as fully masked, and the default block-wise denoising procedure described in Section 2.2 is applied. The completion is defined as the portion after the prompt in the final restored sequence:

$$o_{\text{AF}} = x_{(M,T)}^{|q|+1:|q|+L}. \tag{8}$$

Answer-Guided (AG) completions. Under the AG rollout condition, a completion is generated in two stages: (1) generate a reasoning prefix by fixing the answer as a suffix anchor, and (2) predict the answer conditioned on the generated reasoning. This uses y not only as a verification target but also as a conditioning signal during generation.

(1) Answer-guided reasoning. The answer is anchored at the suffix of the generation segment. To reduce trivial cues from answer length, the anchor is augmented with a margin of length m by appending m padding tokens:

$$\tilde{y} = y \oplus \langle\text{pad}\rangle^{m} \tag{9}$$

To allow the suffix anchor to constrain reasoning tokens throughout the generation segment, we use a single-block denoising setting with $M = 1$ and $B = L$, so that all masked positions in the reasoning prefix are eligible for restoration at every denoising step, while the anchored suffix remains fixed. We analyze this choice in Section 5.5 and Appendix E.1.

With the suffix $\tilde{y}$ fixed, the remaining $L - |\tilde{y}|$ tokens are restored to obtain reasoning r_j for $j = 1, \ldots, G_{AG}$:

$$r_j \sim \pi_{\theta_{\text{old}}}\Big(\cdot \;\Big|\; q \oplus \langle\text{mask}\rangle^{(L-|\tilde{y}|)} \oplus \tilde{y}\Big) \tag{10}$$

(2) Answer re-prediction. To compute verifiable rewards, the answer is predicted after removing the anchored suffix. In stage (1), $\tilde{y}$ is used only to guide reasoning and should not be used directly for reward computation. Otherwise, the model could receive high reward from the fixed suffix even when the generated reasoning is uninformative or incorrect. Accordingly, rewards are computed from the correctness of the answer predicted from r_j. In stage (2), the fixed answer is removed and the answer tokens are generated conditioned on $q \oplus r_j$ to obtain a_j. This is formulated as a denoising problem that restores a masked segment of length $|\tilde{y}|$ appended after $q \oplus r_j$:

$$a_j \sim \pi_{\theta_{\text{old}}}\Big(\cdot \;\Big|\; q \oplus r_j \oplus \langle\text{mask}\rangle^{|\tilde{y}|}\Big), \tag{11}$$

For this answer re-prediction stage, we use the inference procedure in Section 2.2 with $M = 1$ and $B = |\tilde{y}|$, and define the number of discrete denoising steps as a separate hyperparameter S_{ans}, which is typically much smaller than S.

Finally, an AG completion is defined as the concatenation of the reasoning and the predicted answer:

$$o_{\text{AG}_j} := r_j \oplus a_j \tag{12}$$

In summary, under the AG rollout condition, the model first generates a reasoning prefix conditioned on an anchored answer suffix and then re-predicts the answer from that reasoning to construct the final completion.

3.2 Group-Relative Advantage

A key idea of AG-GRPO is to compute group-relative advantages over a shared set of AF and AG completions. For a single prompt–answer pair (q, y), we group AF completions $\{o_{\text{AF}_i}\}_{i=1}^{G_{\text{AF}}}$ and AG completions $\{o_{\text{AG}_j}\}_{j=1}^{G_{\text{AG}}}$ into a single set, and define GRPO's group-relative advantage over this shared set. Denote the total number of completions by $G := G_{\text{AF}} + G_{\text{AG}}$.

Let R_{AF_i} and R_{AG_j} denote scalar rewards obtained from a task-specific reward function for each completion. The group-mean reward for a given input (q, y) is defined as

$$\bar{R} = \frac{1}{G}\left(\sum_{i=1}^{G_{\text{AF}}} R_{\text{AF}_i} + \sum_{j=1}^{G_{\text{AG}}} R_{\text{AG}_j}\right), \tag{13}$$

which serves as a common baseline across all completions for the same prompt. Following (Liu et al.,

Algorithm 1 Answer-Guided Group Relative Policy Optimization (AG-GRPO)

Require: Reference policy π_{ref}, Reward function $\mathcal{R}$, Dataset $\mathcal{D}$, Rollout counts G_{AF} and G_{AG}, Max completion length L, Margin m, Prompt masking probability p_{mask}, KL coefficient β, Update iterations μ
Ensure: Optimized policy π_θ

Initialize policy $\pi_\theta \leftarrow \pi_{\text{ref}}$
while not converged **do**
  $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
  Sample a prompt-answer pair (q, y) from $\mathcal{D}$

  Step 1: Rollout Sampling
  Generate answer-free completions $\{o_{\text{AF}_i}\}_{i=1}^{G_{\text{AF}}}$ via $o_{\text{AF}_i} \sim \pi_{\theta_{\text{old}}}(\cdot \mid q \oplus \langle\text{mask}\rangle^L)$
  for $j = 1, \ldots, G_{\text{AG}}$ **do**
    Construct anchor with margin $\tilde{y} \leftarrow y \oplus \langle\text{pad}\rangle^m$
    Sample answer-guided reasoning $r_j \sim \pi_{\theta_{\text{old}}}(\cdot \mid q \oplus \langle\text{mask}\rangle^{(L-|\tilde{y}|)} \oplus \tilde{y})$
    Re-predict answer $a_j \sim \pi_{\theta_{\text{old}}}(\cdot \mid q \oplus r_j \oplus \langle\text{mask}\rangle^{|\tilde{y}|})$
    Construct completion $o_{\text{AG}_j} \leftarrow r_j \oplus a_j$
  end for

  Step 2: Shared Group-Relative Advantage Estimation
  Compute rewards $R_{\text{AF}_i} = \mathcal{R}(o_{\text{AF}_i}, y)$ and $R_{\text{AG}_j} = \mathcal{R}(o_{\text{AG}_j}, y)$ for all i, j
  Compute shared group mean baseline $\bar{R} \leftarrow \frac{1}{G_{\text{AF}}+G_{\text{AG}}}\left(\sum_{i=1}^{G_{\text{AF}}} R_{\text{AF}_i} + \sum_{j=1}^{G_{\text{AG}}} R_{\text{AG}_j}\right)$
  Compute advantages $A_{\text{AF}_i} \leftarrow R_{\text{AF}_i} - \bar{R}$, $A_{\text{AG}_j} \leftarrow R_{\text{AG}_j} - \bar{R}$

  Step 3: Policy Optimization
  for $n = 1, \ldots, \mu$ **do**
    Sample randomly masked context $q' \sim \text{Mask}(q; p_{\text{mask}})$
    Compute policy losses $\mathcal{L}_{\text{AF}}$ and $\mathcal{L}_{\text{AG}}$ via clipped objective in Eq. (21)–(23)
    Compute KL divergence penalty $\mathcal{L}_{\text{KL}}$ using Eq. (24)
    Update θ by minimizing $\mathcal{L}_{\text{AG-GRPO}} = \mathcal{L}_{\text{AF}} + \mathcal{L}_{\text{AG}} + \beta\mathcal{L}_{\text{KL}}$
  end for
end while

2025b; Zhao et al., 2025), we use unnormalized advantages. Thus, the advantages for AF and AG completions are

$$A_{\text{AF}_i} = R_{\text{AF}_i} - \bar{R}, \qquad i = 1, \ldots, G_{\text{AF}}, \quad (14)$$

$$A_{\text{AG}_j} = R_{\text{AG}_j} - \bar{R}, \qquad j = 1, \ldots, G_{\text{AG}}. \quad (15)$$

To make the effect of the shared baseline explicit, define $\mu_{\text{AF}} := \mathbb{E}[R(o_{\text{AF}}, y)]$ and $\mu_{\text{AG}} := \mathbb{E}[R(o_{\text{AG}}, y)]$ as the expected rewards of AF and AG completions, respectively. Then

$$\mathbb{E}[\bar{R}] = \frac{G_{\text{AF}}}{G}\mu_{\text{AF}} + \frac{G_{\text{AG}}}{G}\mu_{\text{AG}}. \quad (16)$$

Accordingly, the expected advantages satisfy

$$\mathbb{E}[A_{\text{AF}}] = \mu_{\text{AF}} - \mathbb{E}[\bar{R}] = \frac{G_{\text{AG}}}{G}\left(\mu_{\text{AF}} - \mu_{\text{AG}}\right), \quad (17)$$

$$\mathbb{E}[A_{\text{AG}}] = \mu_{\text{AG}} - \mathbb{E}[\bar{R}] = \frac{G_{\text{AF}}}{G}\left(\mu_{\text{AG}} - \mu_{\text{AF}}\right). \quad (18)$$

Because AG completions condition on the ground-truth answer y during rollout generation, they can achieve higher rewards than AF completions, especially early in training. When $\mu_{\text{AG}} > \mu_{\text{AF}}$, the shared baseline induces negative expected advantage for AF completions and positive expected advantage for AG completions. Under shared policy parameters, this creates a contrastive signal between AF and AG updates, allowing stronger AG rollouts to affect updates applied to the answer-free policy used at test time. We analyze the merit of grouping AF and AG completions together in Section 5.2, and discuss the ideal AG/AF ratio in Appendix E.2.

3.3 Policy Optimization

We optimize the policy π_θ using the group-relative advantages defined above. Because masked dLLMs do not admit an autoregressive chain-rule factorization of sequence likelihood, we follow Zhao et al. (2025) and use a per-token likelihood estimator with a mean-field assumption. Let q' denote a masked prompt obtained by masking each token in q independently with probability p_{mask}.

$$\phi_{\pi_\theta}(o^k \mid q') = f_\theta^k\left(o^k \,\middle|\, q' \oplus \langle\text{mask}\rangle^{|o|}\right). \quad (19)$$

Let $\pi_{\theta_{\text{old}}}$ be the rollout policy used to generate completions and π_θ the current policy. For an arbitrary completion o and position k, we define the per-token ratio as

$$\rho^k(o) = \frac{\phi_{\pi_\theta}(o^k \mid q')}{\phi_{\pi_{\theta_{\text{old}}}}(o^k \mid q')}. \tag{20}$$

Given a completion o and its advantage $A(o)$, we define PPO's clipped objective as the mean over positions:

$$\mathcal{L}_{\text{clip}}(o, A) := -\frac{1}{|o|}\sum_{k=1}^{|o|} \min\Big(\rho^k(o)\,A, \text{clip}(\rho^k(o), 1-\varepsilon, 1+\varepsilon)\,A\Big). \tag{21}$$

Given AF completions $\{o_{\text{AF}_i}\}_{i=1}^{G_{\text{AF}}}$ and AG completions $\{o_{\text{AG}_j}\}_{j=1}^{G_{\text{AG}}}$, with corresponding advantages $\{A_{\text{AF}_i}\}_{i=1}^{G_{\text{AF}}}$ and $\{A_{\text{AG}_j}\}_{j=1}^{G_{\text{AG}}}$, we define the AF and AG losses as

$$\mathcal{L}_{\text{AF}} = \frac{1}{G_{\text{AF}}}\sum_{i=1}^{G_{\text{AF}}} \mathcal{L}_{\text{clip}}(o_{\text{AF}_i}, A_{\text{AF}_i}), \tag{22}$$

$$\mathcal{L}_{\text{AG}} = \frac{1}{G_{\text{AG}}}\sum_{j=1}^{G_{\text{AG}}} \mathcal{L}_{\text{clip}}(o_{\text{AG}_j}, A_{\text{AG}_j}). \tag{23}$$

Following GRPO, we introduce a KL penalty to prevent the policy from drifting too far from the reference policy π_{ref}:

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}\left[\phi_{\pi_\theta}(\cdot \mid q') \,\|\, \phi_{\pi_{\text{ref}}}(\cdot \mid q')\right]. \tag{24}$$

The final objective is

$$\mathcal{L}_{\text{AG-GRPO}} = \mathcal{L}_{\text{AF}} + \mathcal{L}_{\text{AG}} + \beta\,\mathcal{L}_{\text{KL}}. \tag{25}$$

In summary, AG-GRPO uses AF and AG rollout conditions for the same prompt, computes group-relative advantages over the resulting shared set of completions, and optimizes the policy under a combined AF/AG objective. Algorithm 1 summarizes the overall training procedure.

4 Evaluation

4.1 Experimental Settings

Baselines. All methods are initialized from the same pretrained checkpoint, LLaDA-8B-Instruct (Nie et al., 2025). We compare: (1) **LLaDA**, the pretrained model without RL fine-tuning; (2) ***diffu*-GRPO** (Zhao et al., 2025), a GRPO-based RLVR baseline for masked dLLMs; and (3) **AG-GRPO**, our method.

Datasets. We evaluate reasoning and puzzle-solving performance on Sudoku (Cordero, 2025), GSM8K (Cobbe et al., 2021), and MATH-500 (Hendrycks et al., 2021). For all three tasks, we extract the final answer from the model output and report exact match (EM) accuracy against the ground-truth answer. For code generation, we train on Kodcode (Xu et al., 2025) and evaluate on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021b), reporting pass@1, i.e., the fraction of problems solved by a single sampled completion. Detailed dataset descriptions are provided in Appendix C.

Training Setup. Following Zhao et al. (2025), both RL methods use the same rollout budget: a total of G=6 completions per prompt. Since *diffu*-GRPO (Zhao et al., 2025) does not condition on the answer, it corresponds to G_{AF}=6 and G_{AG}=0 in our notation, whereas AG-GRPO uses G_{AF}=3 and G_{AG}=3. Across all experiments, input prompts are truncated to at most 200 tokens. During training, we fix the generation length to L_{train}=256, while at evaluation time we vary L according to the experimental setting. For AG completions, the answer tokens y are wrapped with `<answer>` and `</answer>` to delimit the answer span.

For mathematics and puzzle tasks, we train separate models for each tokens-per-step setting $N \in \{8, 4, 2\}$, while for coding tasks we fix N=2. We set the number of denoising steps for the answer re-prediction stage to S_{ans}=16, the KL coefficient β to 0.1 when S=128 and 0.04 otherwise, and the answer margin length m to match the dataset-specific average answer length: 2 for GSM8K, 4 for MATH-500 and Sudoku, and 10 for Kodcode. For mathematics and puzzle tasks, we train for 4,000 steps. For coding, we filter to prompts of length $\leq$ 200, train for 2,000 steps on Kodcode, and evaluate on HumanEval and MBPP. All reported results are taken from the final checkpoint. Implementation details are provided in Appendix F.

4.2 Results

Table 1 reports results on GSM8K, MATH-500, and Sudoku. For each tokens-per-step setting $N \in \{8, 4, 2\}$, we evaluate models trained with the same N under generation lengths $L \in \{128, 256, 512\}$, so that the number of denoising steps varies as $S = L/N$. Overall, AG-GRPO outperforms both the pretrained LLaDA-8B-Instruct baseline and *diffu*-GRPO across a wide range of settings. The gains

L	Model	GSM8K				MATH-500				Sudoku			
		N=8	N=4	N=2	Avg.	N=8	N=4	N=2	Avg.	N=8	N=4	N=2	Avg.
128	LLaDA-8B-Instruct	44.05	57.47	66.34	55.95	19.60	22.60	28.80	23.67	3.91	9.42	7.67	7.00
	diffu-GRPO (2025)	44.88	58.53	**68.31**	57.24	20.20	26.20	27.40	24.60	13.77	18.46	20.56	17.60
	AG-GRPO (ours)	**51.71**	**64.67**	68.16	**61.51**	**20.60**	**27.20**	**29.40**	**25.73**	**40.82**	**45.31**	**37.60**	**41.24**
256	LLaDA-8B-Instruct	44.43	65.28	76.19	61.97	19.80	28.80	32.04	26.88	3.47	7.28	5.96	5.57
	diffu-GRPO (2025)	51.48	69.29	76.57	65.78	23.40	29.20	33.60	28.73	14.79	13.13	16.26	14.73
	AG-GRPO (ours)	**55.27**	**70.51**	**76.72**	**67.50**	**25.40**	**32.60**	**35.60**	**31.20**	**37.70**	**49.32**	**45.60**	**44.21**
512	LLaDA-8B-Instruct	38.59	68.31	78.70	61.87	16.60	27.60	35.20	26.47	3.56	4.54	3.66	3.92
	diffu-GRPO (2025)	45.34	61.41	**79.45**	62.07	18.60	31.80	**38.00**	29.47	16.06	7.67	6.45	10.06
	AG-GRPO (ours)	**55.57**	**72.71**	79.08	**69.12**	**25.80**	**34.40**	36.80	**32.33**	**26.61**	**23.97**	**39.10**	**29.89**

Table 1: Results on reasoning benchmarks as a function of sequence length L. Each setting fixes tokens-per-step $N \in \{8, 4, 2\}$, and the number of denoising steps is determined by $S = L/N$. Accordingly, L=128 yields S=(16, 32, 64), L=256 yields S=(32, 64, 128), and L=512 yields S=(64, 128, 256) for N=(8, 4, 2), respectively. Avg. denotes the average over the three N settings. Higher is better and **bold** denotes the best score in each column.

Model	HumanEval		MBPP	
	L=128	L=256	L=128	L=256
LLaDA-8B-Instruct	**33.54**	35.98	36.19	35.41
diffu-GRPO (2025)	32.32	33.54	37.74	36.19
AG-GRPO (ours)	32.93	**36.59**	**38.52**	**36.58**

Table 2: Code generation on HumanEval and MBPP. Scores are pass@1 (%). We fix tokens-per-step (N) to 2. Higher is better and **bold** denotes the best score in each column.

are particularly pronounced on Sudoku, where the base model starts from low accuracy and the task imposes strict structural constraints, making verifiable rewards especially sparse. Under these conditions, AG-GRPO yields larger improvements in AF performance.

Table 2 shows that AG-GRPO matches or improves upon *diffu*-GRPO on code generation as well, indicating that the same training framework is also applicable to tasks with executable verifiers. Section 5 analyzes the training dynamics of AG-GRPO and the effect of its key design choices, and Appendix G provides qualitative case studies.

5 Analysis

5.1 Reward Curves

To study the training dynamics of AG-GRPO, we log rewards for AF and AG completions separately for each training batch. Figure 1 shows results on GSM8K trained with N=8.

Left. The left plot compares AF and AG rewards within AG-GRPO during training. Within AG-GRPO, AG completions improve rapidly early in training, while AF completions improve more gradually. Although AG rewards remain higher throughout training, AF rewards also rise steadily after the initial phase. This pattern is consistent with the view that the AG rollout condition provides stronger early training signals during optimization.

Right. The right plot compares AF rewards, which correspond to the test-time condition, between AG-GRPO and *diffu*-GRPO. For a fair comparison, we compute AF rewards using 3 AF completions per prompt, taking the first 3 AF completions from *diffu*-GRPO's G_{AF}=6, while AG-GRPO uses its 3 AF completions (G_{AF}=3). Under this matched AF comparison, AG-GRPO attains higher AF reward during training.

5.2 Reward Grouping

We analyze how the choice of reward grouping affects learning on AF completions. To isolate this effect, we compare variants that differ only in whether AF and AG share the same group-mean baseline when computing advantages. Figure 2 compares four reward-grouping strategies on Sudoku with N=8 and L=256.

(1) **Relative** (grouped mean over AF+AG) groups AF and AG together and yields the fastest and most stable convergence.

(2) **AG-Intra** (AG: intra, AF: grouped) uses an AG-only mean for AG advantages but keeps the grouped baseline for AF advantages, so AF can still benefit from higher-reward AG rollouts.

(3) **AF-Intra** (AF: intra, AG: grouped) normalizes AF only within its own group, weakening relative comparisons against AG and making AF re-

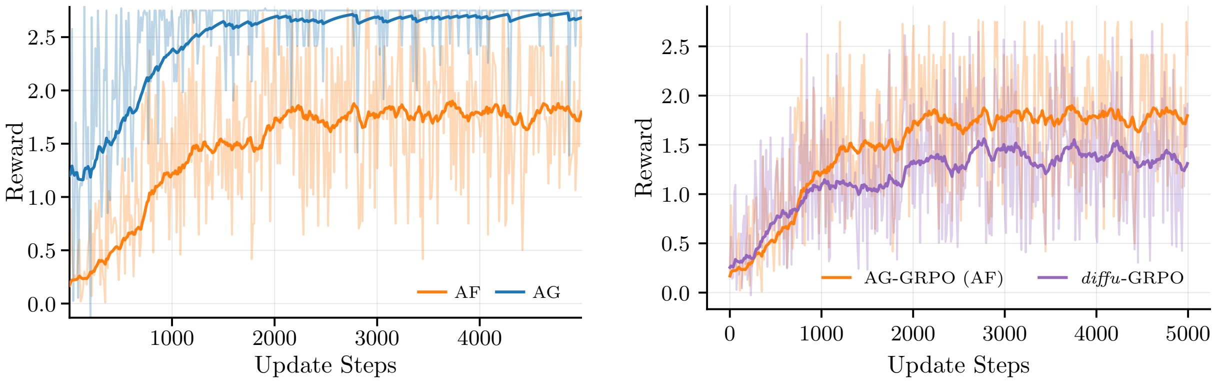

Figure 1: Reward curves of AG-GRPO on GSM8K trained with $N = 8$ and $L = 256$. Left: reward trajectories for AF and AG completions during AG-GRPO training. Right: comparison between AG-GRPO (AF) and *diffu*-GRPO. Curves are EMA-smoothed with α=0.05 for readability.

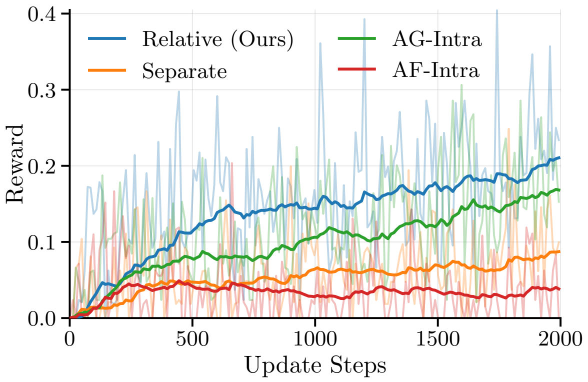

Figure 2: Effect of reward grouping on AF rewards on Sudoku with N=8 and L=256. Curves are EMA-smoothed with α=0.05 for readability.

S_{train}	S_{test}	**Model**	**MATH-500**	**Sudoku**
32	64	*diffu*-GRPO	30.8	15.28
		AG-GRPO (ours)	**31.6**	**38.53**
	96	*diffu*-GRPO	**32.0**	14.79
		AG-GRPO (ours)	31.4	**38.48**
	128	*diffu*-GRPO	33.2	15.58
		AG-GRPO (ours)	**34.4**	**38.38**
64	96	*diffu*-GRPO	31.8	8.45
		AG-GRPO (ours)	**33.0**	**51.27**
	128	*diffu*-GRPO	31.8	12.16
		AG-GRPO (ours)	**33.6**	**51.07**

Table 3: Cross-step evaluation on MATH-500 and Sudoku. S_{train} and S_{test} denote the denoising budgets used during training and evaluation, respectively.

ward convergence less stable.

(4) **Separate** (AF: intra, AG: intra) removes cross-condition relative comparisons entirely, limiting the extent to which AF can benefit from AG signals.

5.3 Denoising Steps

As described in Section 2.2, larger S corresponds to finer-grained restoration. Since larger denoising budgets also increase rollout cost, a practical question is whether AG-GRPO remains effective when trained with smaller S_{train} but evaluated with larger S_{test}.

Table 3 reports this cross-step evaluation on MATH-500 and Sudoku. Across both training settings ($S_{\text{train}} \in \{32, 64\}$), AG-GRPO generally preserves its advantage over *diffu*-GRPO under larger denoising budgets at test time. These results suggest that the observed gains are not explained solely by using finer denoising during training.

5.4 Reasoning Quality

A natural concern is that AG rollouts might achieve higher rewards through degenerate shortcuts rather than through reasoning that is more consistent with the ground-truth answer. To examine this, we compare the reasoning quality of AF and AG rollouts using an LLM judge.

We conduct this analysis on GSM8K under S=128 (N=2, L=256), a fine-grained denoising setting. Among the reasoning benchmarks in Table 1, GSM8K is the highest-performing benchmark under this setting, and finer-grained restoration appears to make it easier to recover the correct answer. Accordingly, AF and AG completions exhibit the smallest gap in the total number of correct final answers across sampled training completions in this setting. This makes the evaluation a stringent test of whether AG improves reasoning quality beyond what is visible from final-answer accuracy

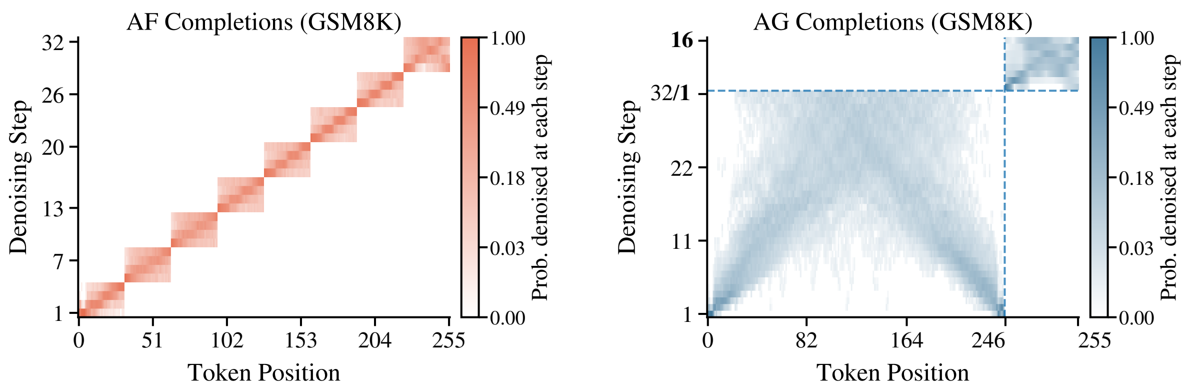

Figure 3: Denoising trajectory heatmaps on GSM8K. Each column shows the distribution over the denoising step at which the token at that position is first restored. Left: AF completion. Right: AG completion. **Bold** y-axis ticks indicate the answer-prediction phase in AG.

Answer	Judge verdict	Completion	
		AF	AG
Correct	YES	1376	1671
	NO	509	215
Wrong	YES	66	72
	NO	53	46

Table 4: Contingency table of answer correctness and judge verdict on GSM8K (S=128). The judge verdict indicates whether the generated reasoning supports the ground-truth answer.

alone.

We use `Llama-3.3-70B-Instruct-Turbo` as the judge model (AI@Meta, 2024). For each rollout, the judge determines whether the generated reasoning supports the ground-truth answer (YES) or contains a core flaw that would not reasonably lead to it (NO). The judge prompt is provided in Appendix H.

Table 4 reports the resulting contingency table. Although AF and AG exhibit similar aggregate answer correctness, AG yields substantially more Correct/YES cases, while Wrong/NO cases remain rare in both settings. These results suggest that answer anchoring improves the quality of the generated reasoning, not merely the final answer.

5.5 Denoising Trajectories

Figure 3 visualizes denoising trajectories for AF and AG completions. Using rollouts collected during the first 1,000 training steps on GSM8K with N=8, we compute, for each token position, the average distribution over the denoising step at which that token is first restored.

Left. For AF completions, we observe a staircase pattern aligned with block boundaries. Within each block, earlier token positions also tend to be restored earlier, indicating a left-to-right bias both across and within blocks.

Right. For AG completions, during *answer-guided reasoning* restoration, restoration probability is higher both immediately after the prompt and near the boundary adjacent to the anchored answer suffix. This pattern is consistent with the design in Section 3.1, where the suffix anchor remains fixed while all masked reasoning positions are restored jointly under a single-block setting. Additional denoising trajectories are provided in Appendix E.1.

6 Conclusion

One limitation of standard RLVR is that ground-truth answers are typically used only for reward verification, rather than to guide rollout generation. We address this limitation with AG-GRPO for masked dLLMs, which combines answer-guided and answer-free rollouts under a shared group-relative objective. In the answer-guided condition, the model conditions reasoning on an anchored ground-truth answer suffix and then re-predicts the answer, allowing answer-guided signals to improve the answer-free policy used at test time.

Empirically, AG-GRPO improves masked dLLMs across mathematics, puzzle-solving, and code-generation tasks, especially on tasks with strict constraints and low base accuracy. Additional analyses support its effect on training dynamics, reasoning quality, and denoising behavior.

Limitations

First, generating AG completions introduces additional computation because it requires two stages: answer-guided reasoning generation and answer re-prediction conditioned on the generated reasoning. However, this overhead remains modest because answer re-prediction reconstructs a relatively short answer suffix and uses a small number of denoising steps. Detailed overhead analysis is provided in Appendix E.3.

Second, the current method is not directly applicable to standard autoregressive LLMs or to tasks that rely purely on preference-based rewards without verifiable answers. In addition, due to the limited availability of publicly released discrete dLLMs suitable for large-scale experimentation, we could not extensively validate the method across a diverse set of backbones and model scales. As more compatible models and implementations become available, future work should evaluate generalization across broader settings.

Acknowledgments

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

References

AI@Meta. 2024. Llama 3.3 model card. Official model card for meta-llama/Llama-3.3-70B-Instruct, accessed 2026-04-20.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rian van den Berg. 2021a. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv: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. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Arel Cordero. 2025. Arel's sudoku generator. `https://www.ocf.berkeley.edu/~arel/sudoku/main.html`. Accessed: 2025-04-08.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Jiawei Han, Hao Peng, and Lingpeng Kong. 2025a. Scaling diffusion language models via adaptation from autoregressive models. *Preprint*, arXiv:2410.17891.

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

Zhengfu He, Tianxiang Sun, Qiong Tang, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. 2023. DiffusionBERT: Improving generative masked language models with diffusion models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4521–4534, Toronto, Canada. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *Preprint*, arXiv:2103.03874.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35.

Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. 2022. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe.

2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Huanyu Liu, Jia Li, Chang Yu, Taozhi Chen, Yihong Dong, Lecheng Wang, XiaoLong Hu, and Ge Li. 2025a. Evocot: Overcoming the exploration bottleneck in reinforcement learning. *Preprint*, arXiv:2508.07809.

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

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

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

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France. PMLR.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *Preprint*, arXiv:2211.14275.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. 2025. Kodcode: A diverse, challenging, and verifiable synthetic dataset for coding. *Preprint*, arXiv:2503.02951.

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

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

Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. 2025. Llada-v: Large language diffusion models with visual instruction tuning. *Preprint*, arXiv:2505.16933.

Runpeng Yu, Xinyin Ma, and Xinchao Wang. 2025. Dimple: Discrete diffusion multimodal large language model with parallel decoding. *Preprint*, arXiv:2505.16990.

Oussama Zekri and Nicolas Boullé. 2025. Fine-tuning discrete diffusion models with policy gradient methods. *Preprint*, arXiv:2502.01384.

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

A Related Work

A.1 Diffusion-based Language Models

Diffusion language models (dLLMs) generate text via iterative denoising rather than autoregressive next-token prediction. Early studies followed two directions: operating in a continuous embedding space that diffuses and denoises token embeddings (Li et al., 2022), and discrete-space formulations that corrupt and restore tokens directly (Austin et al., 2021a). The discrete paradigm was further developed by leveraging pretrained masked language models as denoisers, as in Diffusion-BERT (He et al., 2023). More recently, the paradigm has scaled to large models: Dream (Ye et al., 2025) and DiffuLLaMA (Gong et al., 2025a) build on autoregressive backbones, while LLaDA (Nie et al., 2025) is trained as a diffusion LM from scratch. Multimodal extensions that couple text and vision—e.g., LLaDA-V (You et al., 2025), Dimple (Yu et al., 2025), and

MMaDA (Yang et al., 2025)—have also been actively explored.

A.2 Reinforcement Learning Algorithms

Reinforcement learning for alignment and reasoning in large language models traces back to policy-gradient methods such as TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017). RLHF (Ouyang et al., 2022) trains a reward model from human preferences and optimizes with PPO, substantially improving instruction following and response quality. GRPO (Shao et al., 2024) removes the critic and normalizes advantages through within-prompt, multi-sample relative comparisons. Preference-based alternatives such as DPO (Rafailov et al., 2023) simplify the RLHF pipeline. RLVR uses executable or checkable outcomes, such as unit-test pass rates for code (Le et al., 2022) and exact-answer matching for math (Cobbe et al., 2021), while process supervision (Lightman et al., 2023; Uesato et al., 2022) evaluates intermediate reasoning steps.

A.3 Reinforcement Learning for dLLMs

Applying RL to dLLMs to enhance reasoning is comparatively recent, with recent work focusing mainly on policy-gradient methods. The policy-gradient line was initiated by *diffu*-GRPO (Zhao et al., 2025), which introduced an efficient single-step log-likelihood estimator that enables GRPO for dLLMs. Subsequent work focuses on mitigating the high-variance nature of policy-gradient estimation while adapting to the structural characteristics of diffusion models. DiffuCoder (Gong et al., 2025b) proposes Coupled-GRPO with complementary mask sampling to reduce variance; SEPO (Zekri and Boullé, 2025) incorporates score-entropy terms and importance sampling to obtain lower-variance gradient estimators; and MMaDA (Yang et al., 2025) extends these ideas to the multimodal regime via UniGRPO.

B Masked dLLM Modeling

Masked Diffusion Language Model (Austin et al., 2021a; Li et al., 2022; He et al., 2023; Nie et al., 2025) generates text by progressively restoring masked tokens over time. Given an input prompt, the model defines the initial generative segment as x_t, and denoises it along a continuous time variable $t \in (0, 1]$. The variable t represents the progress of the restoration process and determines how many denoising steps are required to reconstruct the full sequence.

The model $f_\theta(\cdot \mid x_t)$ is trained to predict all masked tokens in x_t simultaneously. While this resembles masked language modeling, the masking ratio varies with t.

Forward Process (Noising). Suppose we are given a fully observed sequence $x_0 \in \mathcal{V}^L$ at t=0. As t increases, tokens are progressively replaced by the mask token according to a predefined masking probability α_t. Following LLaDA, α_t is a monotone (typically linear) masking schedule; for example, setting $\alpha_t = t$ makes the masking probability increase linearly with t and yields a fully masked state at t=1.

The forward process can equivalently be described using a binary mask vector $m_t \in \{0, 1\}^L$, where each position independently samples whether it is masked:

$$m_t^i \sim \text{Bernoulli}(\alpha_t), \quad i = 1, \dots, L,$$

and the corrupted token is determined as

$$x_t^i = \begin{cases} \langle \text{mask} \rangle, & m_t^i = 1, \\ x_0^i, & m_t^i = 0. \end{cases}$$

Under the position-wise independent masking assumption, the conditional distribution factorizes as

$$q_{t|0}(x_t \mid x_0) = \prod_{i=1}^{L} q_{t|0}(x_t^i \mid x_0^i). \tag{26}$$

At each position i, the conditional distribution can be written as a two-point mixture:

$$\begin{aligned} & q_{t|0}(x_t^i \mid x_0^i) \\ & = (1 - \alpha_t)\, \mathbf{1}[x_t^i = x_0^i] + \alpha_t\, \mathbf{1}[x_t^i = \langle \text{mask} \rangle]. \end{aligned} \tag{27}$$

Reverse Process (Denoising). The reverse (denoising) process reconstructs the clean sequence x_0 from a partially masked state x_t. Given x_t, the learned mask predictor $f_\theta(\cdot \mid x_t)$ outputs, for each position i, a categorical distribution over the vocabulary $\mathcal{V}$, denoted $f_\theta^i(\cdot \mid x_t)$. During denoising, tokens at unmasked positions are kept fixed (i.e., copied from the input state), while only the masked positions are updated using the mask predictor's predictions to recover x_0^i. From a probabilistic perspective, $f_\theta^i(\cdot \mid x_t)$ can be viewed as an approximation to the ideal posterior $q_{0|t}(x_0^i \mid x_t)$.

In generation, we initialize from the fully masked state (t=1) and iteratively apply a fixed denoising (unmasking) schedule. At each step, we select a subset of currently masked positions and replace them with tokens predicted by f_θ, thereby progressively decreasing the mask ratio until a complete sequence is obtained.

C Datasets

We use publicly available datasets: GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021), Sudoku (Cordero, 2025), Kodcode (Xu et al., 2025), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021b) for training and evaluation.

GSM8K. A multi-step reasoning benchmark of grade-school to middle-school math word problems. Out of 8.5K problems, the official split provides *train* 7,473 and *test* 1,319. Problems typically require 2–8 steps of basic arithmetic, ratios, and unit conversions, and are scored against a single numeric (or short symbolic) answer. At evaluation time, we apply light string normalization and compute exact match. The original dataset includes reference rationales (chain-of-thought). We use the public split as-is.

MATH-500. An evaluation subset selected from MATH containing high–school competition-level problems spanning algebra, number theory, geometry, and combinatorics. Questions are free-form but require a final numeric or short symbolic answer. Scoring is based on exact answer match after simple normalization. Publicly provided splits contain 7,500 training and 500 evaluation problems.

Sudoku. A 4×4 Sudoku with 2×2 subgrids. Inputs are length-16 strings where `0` denotes a blank cell. Outputs are fully filled strings such that each row, column, and 2×2 subgrid contains digits 1–4 exactly once. We mark a prediction correct only if it satisfies all constraints and is consistent with the given entries.

Kodcode. A code-generation dataset where the model writes Python function bodies from natural-language descriptions and examples. Each instance includes a problem description, a function signature, and hidden unit tests. We perform RL in the coding domain on this dataset and evaluate generalization on other code benchmarks. We use the 9,000 preprocessed training instances provided by the official repository[1] for training only.

HumanEval. A collection of Python programming tasks specified by a function signature and a brief description. Evaluation executes public unit tests to determine correctness. We evaluate on all 164 problems.

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

D Reward Functions

We design dataset-specific reward functions tailored to each task's structure and output format, as detailed below.

GSM8K.

- **XML-Structure Reward:** A shaping reward that adds $+0.125$ for each of four XML boundary markers appearing exactly once with the expected newline placement (i.e., `<reasoning>\n`, `\n</reasoning>\n`, `\n<answer>\n`, `\n</answer>`). The sum is capped at 0.5 and further subtracts a small length-based penalty for any trailing text after `</answer>`.
- **Format Rewards (Soft / Strict):** Regex-based format checks; award $+0.5$ if the completion matches the corresponding soft/strict template, otherwise 0.
- **Integer-Type Reward:** Award $+0.5$ if the extracted `<answer>` content is digit-only (via `isdigit()`), otherwise 0.
- **Correctness Reward:** Award $+2.0$ if the extracted `<answer>` string exactly matches the gold answer; otherwise 0.

MATH-500.

- **Boxed-Answer / Tag Reward:** A formatting reward computed from the completion: award $+1.0$ if an `<answer>` block is present and its content contains `\boxed`; award $+0.75$ if only the `<answer>` block is present; award $+0.5$ if only `\boxed` appears; otherwise award $+0.25$.

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

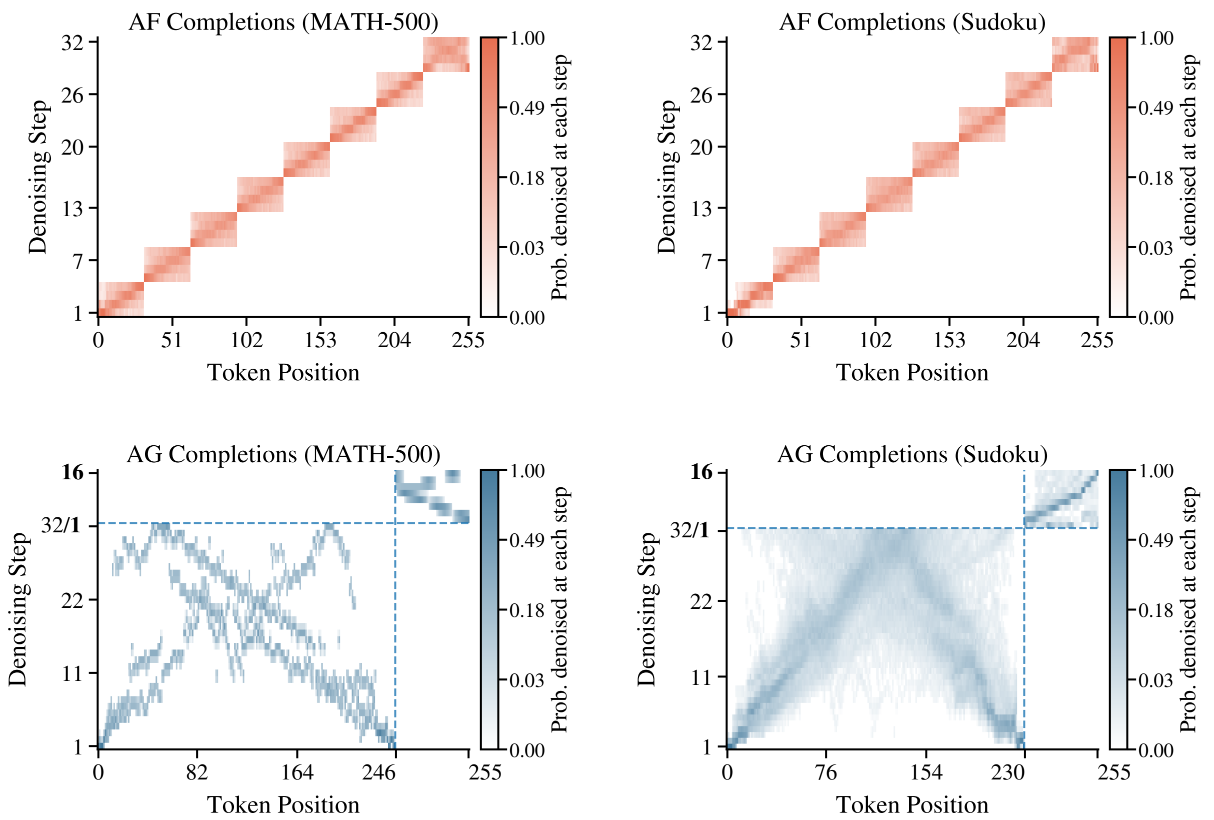

Figure 4: MATH-500 (left) and Sudoku (right) denoising trajectories under the same protocol as Figure 3. Top: AF. Bottom: AG. **Bold** y-tick labels indicate the answer-prediction phase in AG.

- **Mathematical Correctness Reward:** Extract the expression inside the last `\boxed{...}` in the completion (and in the reference solution), and award +2.0 if it is equivalent to the gold answer under `is_equiv` (normalization-based equivalence); otherwise 0.

Sudoku.

- **Blank-level Correctness Reward:** Compare the digit string extracted from `<answer>` with the ground-truth grid and award the fraction of previously blank cells that are filled correctly.

Kodcode.

- **Format Rewards (Soft / Strict):** Award +0.5 if the completion begins with a `<think>...</think>` block and is followed only by whitespace and then a `<answer>...</answer>` block (allowing arbitrary trailing text after `</answer>`). Award an additional +0.5 if the completion consists only of these two blocks, with optional whitespace between and inside the tags.

- **Success Reward (Unit-Test Pass):** If a fenced Python code block (` ```python ... ``` `) is present, execute the extracted code and award +1.0 if it passes all unit tests; otherwise 0.

E Extended Analysis

E.1 Denoising Trajectories on Additional Datasets

Figure 4 shows denoising trajectories on MATH-500 and Sudoku. On both datasets, we observe the same qualitative patterns as in Section 5.5: AF shows staircase-like block patterns with a within-block left-to-right restoration tendency, while AG shows higher restoration probability immediately after the prompt and near the anchored answer suffix. On MATH-500, AG restorations are also more concentrated on certain positions, possibly due to repetitive notational patterns in mathematical expressions and symbols.

E.2 Effect of the Number of AG Completions

We analyze how the number of answer-guided (AG) completions affects performance on MATH-500. Increasing G_{AG} can strengthen answer-guided training signals during training, but it also re-

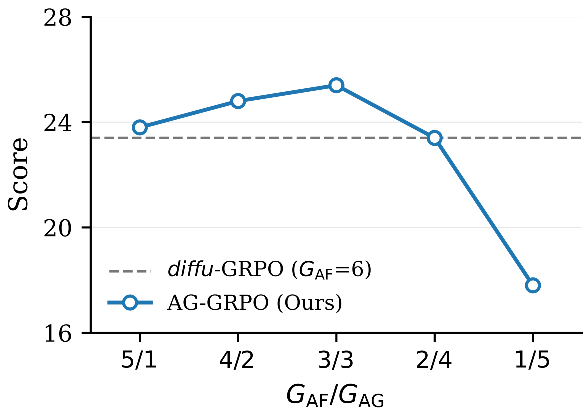

Figure 5: MATH-500 scores as a function of the AF/AG rollout split $(G_{\text{AF}}/G_{\text{AG}})$, with fixed total rollout budget $G_{\text{AF}} + G_{\text{AG}} = 6$ (N=8, L=256).

duces the number of answer-free (AF) samples that match the test-time distribution. To study this trade-off, we fix the total rollout budget per prompt to $G_{\text{AF}} + G_{\text{AG}} = 6$ and vary only the $(G_{\text{AF}}/G_{\text{AG}})$ split while keeping all other settings unchanged.

Figure 5 reports the resulting MATH-500 scores. The best performance is achieved when G_{AF} and G_{AG} are balanced. When G_{AG} is too small, the answer-guided signal is weak; when G_{AG} is too large, optimization on the AF (test-time) distribution becomes insufficient, leading to degraded performance.

E.3 Computational Overhead

The computational cost of Transformer models depends on both sequence length and the number of forward passes (Vaswani et al., 2017). Since prompt length and maximum generation length are fixed in our training, the main source of theoretical overhead is the number of forward passes. Although answer suffix length also affects the actual cost, we first approximate the overhead in terms of S.

Without AG rollouts, the total denoising cost is proportional to

$$G \cdot S. \tag{28}$$

For AG-GRPO, the total cost becomes

$$G_{\text{AF}} \cdot S + G_{\text{AG}} \cdot (S + S_{\text{ans}}), \tag{29}$$

which can be rewritten as

$$G \cdot S + G_{\text{AG}} \cdot S_{\text{ans}}. \tag{30}$$

Therefore, the theoretical overhead ratio is approximated by

$$\frac{G \cdot S + G_{\text{AG}} \cdot S_{\text{ans}}}{G \cdot S}, \tag{31}$$

Answer suffix length		
Dataset	Avg. length	Fraction of L
GSM8K	2.32	0.91%
MATH-500	4.21	1.64%
Sudoku	16.00	6.25%
Kodcode	61.15	23.89%
Measured wall-time		
Setting	*diffu*-GRPO	AG-GRPO
Sudoku (S=32)	9913	11030 (+11.27%)
Sudoku (S=64)	13752	14874 (+8.16%)
Sudoku (S=128)	21444	22565 (+5.23%)
Kodcode (S=128)	34323	37311 (+8.70%)

Table 5: Computational overhead of AG-GRPO

that is,

$$1 + \frac{G_{\text{AG}}}{G} \cdot \frac{S_{\text{ans}}}{S}. \tag{32}$$

Under our training setup (S_{ans}=16, G_{AG}=3, G=6), this gives +25% for S=32, +12.5% for S=64, and +6.25% for S=128. In practice, the overhead is further limited because stage 2 reconstructs only the short answer suffix. Table 5 therefore reports both answer suffix lengths and measured training wall-time. All wall-time measurements use 2× RTX 4090 GPUs and otherwise follow the corresponding main experimental settings in Section 4.1.

F Implementation Details

All experiments are conducted using the Hugging Face `transformers` library, loading the same pretrained model `LLaDA-8B-Instruct`. Training uses the AdamW optimizer (Loshchilov and Hutter, 2019) with the following key hyperparameters: learning rate 3×10^{-6}, weight decay 0.1, warmup ratio 0.0001, and Adam coefficients β_1=0.9, β_2=0.99. We applied LoRA (Hu et al., 2022) with rank r=128, scaling factor α=64, and dropout 0.05. LoRA adapters are inserted into `{q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj}`, and bias parameters are kept frozen. All experiments use multi-GPU training with 2–3 NVIDIA RTX 3090/RTX 4090 GPUs per run, totaling approximately 240 GPU-hours across all reported results.

G Case Study

Case Study 1: Training with GSM8K at early training steps (900)
Question: For 5 days, Chantel makes 2 friendship bracelets every day. She gives away 3 bracelets to her friends at school. Then for four days, she makes 3 friendship bracelets every day. Then she gives away 6 bracelets to her friends at soccer practice. How many bracelets does Chantel have in the end?
Answer: 13

AG-GRPO

AF Completion

<reasoning>
To determine how many friendship bracelets Chantel has in the end, we need to follow the steps of her activities:
1. **Calculate number of bracelets**: Chantel makes 2 days of for 5 days.

$$2 \text{ bracelets/day} \times 5 \text{ days} = 10 \text{ bracelets}$$

2. **Give bracelets away**: Chantel gives away 3 bracelets.

$$10 \text{ bracelets} - 3 \text{ bracelets} = 7 \text{ bracelets}$$

3. **Calculate more bracelets**: Chantel makes 3 bracelets/day for 4 days.

$$3 \text{ bracelets/day} \times 4 \text{ days} = 12 \text{ bracelets}$$

4. **Give bracelets away**: Chantel gives away 6 bracelets.

$$12 \text{ bracelets} - 6 \text{ bracelets} = 6 \text{ bracelets}$$

</reasoning>
<answer>6</answer>

AG Completion

<reasoning>
First, let's calculate the number of friendship bracelets Chantel makes in the first 5 days:

$$2 \text{ bracelets/day} \times 5 \text{ days} = 10 \text{ bracelets}$$

Next, she gives away 3 bracelets to her friends at school:

$$10 - 3 = 7 \text{ bracelets}$$

Now, let's calculate the number of friendship bracelets she makes in the next 4 days:

$$3 \text{ bracelets/day} \times 4 \text{ days} = 12 \text{ bracelets}$$

Then, we add the bracelets made in the next 4 days to the 7 bracelets she had after giving 3 more away:

$$7 + 12 = 19 \text{ bracelets}$$

Finally, she gives away 6 bracelets to her friends at soccer practice:

$$19 - 6 = 13 \text{ bracelets}$$

Therefore, Chantel has 13 friendship bracelets in the end.
</reasoning>
<answer>13</answer>

diffu-GRPO

Completion

<reasoning>
Chantel makes 2 bracelets every day for 5 days, so she makes 2 x 5 = 10 bracelets. She gives away 3 bracelets at school, leaving has 10 - 3 = 7 bracelets. Then she makes 3 bracelets every day for 4 days, so she makes 3 x 4 = 12 bracelets. She gives away 6 bracelets at soccer practice, leaving has 7 - 6 = 1 bracelets.
</reasoning>
<answer>1</answer>

Figure 6: Early-stage GSM8K case study (training step 900; L=256, N=4). We show an answer-free (AF) completion and an answer-guided (AG) completion from AG-GRPO, together with a completion from *diffu*-GRPO. Green/red indicate correct/incorrect final answers.

Case Study 2: Training with GSM8K at later training steps (3600)
Question: Olivia earns $9 per hour. She worked 4 hours on Monday, 3 hours on Wednesday and 6 hours on Friday. How much money did Olivia make this week?
Answer: 117

AG-GRPO

AF Completion

<reasoning>
To determine how much money Olivia made this week, we need to calculate her earnings for each day and then sum them up.
1. On Monday, calculate Olivia's earnings:

$$4 \text{ hours} \times 9 \text{ dollars/hour} = 36 \text{ dollars}$$

2. On Wednesday, calculate Olivia's earnings:

$$3 \text{ hours} \times 9 \text{ dollars/hour} = 27 \text{ dollars}$$

3. On Friday, calculate Olivia's earnings:

$$6 \text{ hours} \times 9 \text{ dollars/hour} = 54 \text{ dollars}$$

4. Add up all the earnings:

$$36 \text{ dollars} + 27 \text{ dollars} + 54 \text{ dollars} = 117 \text{ dollars}$$

Olivia made $117 this week.
</reasoning>
<answer>117</answer>

AG Completion

<reasoning>
To determine how much money Olivia made this week, we need to calculate her earnings for each day she worked and then sum these amounts.
1. On Monday, Olivia worked 4 hours at a rate of $9 per hour:

$$4 \text{ hours} \times \$9 \text{ per hour} = \$36$$

2. On Wednesday, Olivia worked 3 hours at a rate of $9 per hour:

$$3 \text{ hours} \times \$9 \text{ per hour} = \$27$$

3. On Friday, Olivia worked 6 hours at a rate of $9 per hour:

$$6 \text{ hours} \times \$9 \text{ per hour} = \$54$$

To find the total, we sum the earnings from each day:

$$\$36 + \$27 + \$54 = \$117$$

Therefore, Olivia made $117 this week.
</reasoning>
<answer>117</answer>

diffu-GRPO

Completion

<reasoning>
Olivia worked 4 hours on Monday, 3 hours on Wednesday, and 6 hours on Friday, so she worked $4 + 3 + 6 = 13$ hours in total. At $9 per hour, she earned $13 \times 9 = 117$.
</reasoning>
<answer>97</answer>

Figure 7: Later-stage GSM8K case study (training step 3600; L=256, N=4). We show AF and AG completions from AG-GRPO and a completion from *diffu*-GRPO. Green/red indicate correct/incorrect final answers.

H Judge Prompt

Role. You are an evaluator for reasoning quality.
Input. You will be given:

- a QUESTION,
- the GROUND-TRUTH final answer,
- the MODEL REASONING (only the reasoning, not the final answer).

Task. Decide whether the MODEL REASONING plausibly supports deriving the GROUND-TRUTH answer.
Evaluation rules.

- Focus on whether the reasoning provides a coherent path to the ground truth.
- Output `NO` if the reasoning has a core or fatal logical/mathematical flaw, contradicts the problem, or strongly supports a different answer such that it would not reasonably lead to the ground truth even after minor fixes.
- Output `YES` if the reasoning is clearly aligned and would allow a competent solver to reach the ground truth, even if abbreviated.
- If the reasoning is too vague or underspecified to verify, or if it is unclear whether it supports the ground truth, output `UNCERTAIN`.
- Do not judge writing style. Do not reward verbosity.
- If MODEL REASONING is empty or missing, output `NO`.

Output format. Return a single-line JSON object with the following keys:

```
{
"verdict": "YES" | "NO" | "UNCERTAIN",
"rationale": "one or two short sentences explaining the decision"
}
```

Do not output anything else.