

NI Sampling++: Optimizing Token Order for Fast Discrete Diffusion Sampling with Reinforcement Learning

Enshu Liu
Tsinghua University
jsttlgdkycy@gmail.com

Theodore Zhengde Zhao
Microsoft Research
theodorezhao@microsoft.com

Xuefei Ning*
Tsinghua University
foxdoraame@gmail.com

Zinan Lin*†
Microsoft Research
zinanlin@microsoft.com

Yu Wang*
Tsinghua University
yu-wang@mail.tsinghua.edu.cn

Abstract

Discrete diffusion language models (dLLMs) have emerged as a compelling alternative to autoregressive (AR) models, offering flexible generation orders and inherent support for parallel decoding. While early sampling methods of dLLMs rely on heuristic strategies to select tokens to unmask, they remain inefficient as they can only sample a few tokens per step to avoid performance degradation. Recently, NI Sampling addresses this by framing the process as a token order optimization problem, employing a lightweight neural indicator, which is trained in a supervised manner, to determine which tokens to update at each step. However, the supervised learning paradigm of NI Sampling constrains the model to converge toward a deterministic, single target. This approach lacks sufficient exploration of alternative potential solutions, which may lead to suboptimal sampling policies. To bridge this gap, we reformulate token order optimization as a Markov Decision Process (MDP) and optimize the neural indicator using reinforcement learning. Experimental results demonstrate that our method achieves up to $9.7\times$ step compression compared to full-step sampling and is up to $1.96\times$ faster than NI Sampling, while yielding a consistently dominant step-accuracy trade-off.

1 Introduction

Large language models (LLMs) have received widespread attention for their ability to handle diverse tasks in a general and generative manner. Conventional LLMs are mostly autoregressive (AR) models (Van den Oord et al., 2016; Touvron et al., 2023; Achiam et al., 2023), necessitating a left-to-right, one-token-per-step generation process, which significantly hampers sampling flexibility and speed. To tackle this problem, discrete diffusion large language models (dLLMs) (Nie et al., 2024; Hoogeboom et al., 2021; He et al., 2022; Reid et al., 2022; Sun et al., 2022; Nie et al., 2025; Zhu et al., 2025; Bie et al., 2025; Ye et al., 2025; Wu et al., 2025) emerges as a new paradigm for language generation, which have recently gained considerable traction due to their flexibility in generation order and their inherent support to sample multiple tokens per step. Similar to their continuous counterparts, discrete diffusion models iteratively recover the target sequential distribution from a simple prior discrete distribution. Among all types of discrete diffusion paradigms, masked diffusion models (MDMs) represent the most widely adopted discrete diffusion framework due to their superior performance, where the prior distribution consists exclusively of mask tokens. At inference time, MDMs initiate the sampling process from a fully masked sequence and progressively decode the mask tokens until the entire sequence is recovered.

*Corresponding Authors

†Project Advisor

In the sampling process of MDMs, the sampling order is a critical factor to consider. Specifically, it involves determining which tokens should be selected for unmasking at each step. Numerous studies have proposed various criteria for token selection (Chang et al., 2022; Lezama et al., 2022a;b; Wu et al., 2025; Wei et al., 2025; Fu et al., 2025; Mohamed et al., 2025; Kim et al., 2025b; Lee et al., 2025). Max-confidence sampling (Nie et al., 2025; Ye et al., 2025; Kim et al., 2025a) is the most straightforward and safe way for sampling, which selects only one token with the highest confidence per step to avoid correspondence loss of parallel sampling. However, this strategy is inherently inefficient as it limits generation to a single token per step. To mitigate this, confidence threshold sampling (Wu et al., 2025) is introduced, which employs a predefined threshold to decode all tokens whose confidence exceeds the limit. By selecting an appropriate threshold, this method significantly accelerates sampling without compromising performance. More sophisticated criteria have been developed subsequently to further optimize sampling.

However, these methods are based predominantly on empirical heuristics, which limits their acceleration potential. To overcome this, NI Sampling (Liu et al., 2026) first introduced a learnable token ordering mechanism. Specifically, it employs a lightweight neural indicator, which is a small network with negligible inference overhead compared to the pre-trained dLLM, to decide whether each token should be unmasked at each step. To train this indicator, NI Sampling designs a reference token order strategy as the supervisory signal, called trajectory-preserving order, to label every masked position at each step. The indicator is then optimized in a supervised manner based on these labels. By following this specific order, the sampled tokens are guaranteed to maintain the original sampling trajectory, assuming perfect training.

Although NI Sampling achieves a substantial speedup over empirical methods, its performance still falls behind its theoretical upper bound due to training errors. For example, while ground truth trajectory-preserving order achieves $11.4\times$ speedup on GSM8K, NI Sampling can only accelerate the sampling by $4.9\times$ (Liu et al., 2026). We attribute this to the inherent drawback of the supervised learning approach used to train the indicator, which lacks flexibility and only serves as a proxy of the global optimization target. Specifically, while the ultimate objective is to decrease sampling steps and improve performance, strictly imitating the trajectory-preserving order is not the only, nor necessarily the optimal, path to this goal. To further enhance the effectiveness of optimized token order, we propose **NI Sampling++**, which reformulates the training of the neural indicator as a Markov Decision Process (MDP) and optimize it using reinforcement learning (RL). Our main contributions are summarized as follows:

- We propose NI Sampling++, a RL framework designed for token order optimization. We reformulate the unmasking process as a Markov Decision Process (MDP), where the selection of masked positions at each step is modeled as independent Bernoulli trials, with probabilities equal to the binary classification scores given by the neural indicator. We then leverage Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to optimize the indicator’s decision-making policy.
- We introduce an efficiency-aware reward to train the neural indicator, explicitly balancing sampling speed and generation quality. To further stabilize and enhance the RL process, we develop several techniques, including NI Sampling-based initialization, mid-start rollout, and a scaled indicator architecture.
- We conduct extensive experiments on widely adopted mathematics and code generation benchmarks, using LLaDA-8B-Instruct (Nie et al., 2025) and LLaDA-1.5 (Zhu et al., 2025) model. Compared to full-step sampling, **NI Sampling++** achieves up to $9.8\times$ acceleration with only minor accuracy loss. Furthermore, **NI Sampling++** delivers up to $1.96\times$ speedup over NI Sampling and significantly improve the Pareto frontier across the step-accuracy trade-off curve. **NI Sampling++** does not need finetuning any part of the underlying DLM.

2 Preliminary

2.1 Masked Diffusion Models

Masked diffusion models (MDMs) (Sahoo et al., 2024) have emerged as a prominent framework within the landscape of dLLMs, largely owing to their superior performance. Specifically, MDMs are characterized by a continuous-time probability transition process over a random variable $\mathcal{X}$ with a finite support $\{1, \dots, R\}$. In the forward process, MDMs gradually perturb the data distribution p_0 to a simple prior distribution p_T , which is a Dirac distribution with all probability mass centered at a specialized mask token $[M]$. The transition probability for each token is formulated as:

$$q(x_t|x_0) = \prod_{i=1}^n q(x_t^i|x_0^i) = \prod_{i=1}^n \text{Cat}(x_t^i; (1 - \alpha_t)\delta_{x_0^i} + \alpha_t\delta_{[MASK]}). \quad (1)$$

During this process, a neural network is trained to reconstruct the original data given a masked sequence x_t at an arbitrary timestep t . The training objective is given as:

$$\mathcal{L}(\theta) = -\mathbb{E}_{t, x_0 \sim \mathcal{D}, x_t \sim q(x_t|x_0)} (w_t \sum_{i=1}^n \mathbf{1}[x_t^i = [M]] \log p_\theta(x_0^i|x_t)), \quad (2)$$

where $\mathcal{D}$ is the training dataset. A well trained MDM estimates the conditional token probability $p_\theta(x_0|x_t)$ for all masked positions given the current state x_t .

2.2 Sampling Process of MDMs

Inference of a MDM reverses the forward process Eq. (1), iteratively recovering x_0 from a fully masked initial state x_T . The sampling procedure at each step can be viewed as a two-stage process: **(1) token sampling**: firstly, tokens are sampled at each masked position based on the conditional probability given by the MDM; and **(2) token selection**: a subset of these candidate tokens is selected for unmasking. Regarding the token sampling phase, we follow the established practice in contemporary dLLMs (Ye et al., 2025; Nie et al., 2025; Zhu et al., 2025; Wu et al., 2025) by employing greedy decoding (i.e., sampling the token with the maximum likelihood at each position) for our main experiments.

For the token selection phase, dLLMs inherently support the simultaneous unmasking of an arbitrary number of tokens at any masked positions. However, parallelizing the sampling process across multiple tokens often results in a loss of inter-token correspondence (Liu et al., 2024; 2025a; Wu et al., 2025; Kang et al., 2025). Specifically, when sampling k tokens $x^{i_1}, \dots, x^{i_k}$ in parallel, the original joint distribution $p(x^{i_1}, \dots, x^{i_k})$ is approximated as the product of k marginal distributions $\prod_{j=1}^k p(x^{i_j})$, which introduces a distribution shift. This discrepancy will grow as more tokens are involved. Consequently, default decoding strategies constrain the unmasking subset per step to a single token (Nie et al., 2025; Zhu et al., 2025; Ye et al., 2025; Kim et al., 2025a). **Max-confidence sampling** (Nie et al., 2025; Chang et al., 2022) is the most commonly applied method, prioritizing the position exhibiting the highest prediction probability: $i^* = \arg \max_{i \in \{i|x_t^i = [MASK]\}} \max_{j \in \{1, 2, \dots, V\}} p_\theta(x_0^i = j|x_t)$, where x^i is the i -th token in the sequence.

To fully exploit the inherent parallelism of dLLMs, numerous strategies have been developed to maximize the number of tokens unmasked per step while preserving generation quality (Wu et al., 2025; Wei et al., 2025; Fu et al., 2025; Mohamed et al., 2025; Kim et al., 2025b; Lee et al., 2025). A representative approach is confidence threshold sampling (Wu et al., 2025), which decodes all tokens whose probabilities exceed a pre-defined threshold. Building upon this, subsequent works have devised more sophisticated heuristics to determine the eligibility of tokens for unmasking (Wei et al., 2025; Fu et al., 2025; Mohamed et al., 2025; Kim et al., 2025b; Lee et al., 2025). However, these methods remain fundamentally based on hand-crafted empirical rules. Such heuristics act only as sub-optimal proxies for the actual optimization target, thereby limiting the potential of efficient sampling.

2.3 Policy Optimization

Given a policy $\pi_\theta(a_t|s_t)$ that defines the probability of action a_t at state s_t , policy gradient methods optimize the parameters θ by maximizing the following objective:

$$\mathcal{L}_\theta = \mathbb{E}_{t, \tau \sim \pi_\theta} A_t \log \pi_\theta(a_t|s_t), \quad (3)$$

where τ denotes the whole rollout trajectory, and A_t is the advantage at timestep t . To ensure stable updates when the sampling policy diverges from the target policy, Proximal Policy Optimization (PPO) (Schulman et al., 2017) was introduced as a robust framework. Following the emergence of Large Language Models (LLMs), policy optimization has been extensively adopted in the post-training phase to align and enhance capabilities (Guo et al., 2025; Achiam et al., 2023; Jaech et al., 2024). Recently, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) has been proposed as a highly effective alternative to PPO for RL-based LLM training. Specifically, GRPO samples a group of trajectories for each input and computes the advantage of each trajectory as a normalized reward within the group:

$$A_i = \frac{R_i - \text{mean}(\mathbf{R})}{\text{std}(\mathbf{R})}, \quad (4)$$

where R_i is the reward for the i -th trajectory, $\mathbf{R} = (R_1, \dots, R_G)$ represents the set of rewards within the group.

3 Method

In this section, we first provide a formal definition of the token order optimization problem and review the existing baseline, NI Sampling (Liu et al., 2026), in Sec. 3.1. Then we present the core framework of NI Sampling++ in Sec. 3.2. We further elaborate several key techniques of our method in Sec. 3.3.

3.1 Token Order Optimization

Problem Definition. As noted in Sec. 2, the strategy for selecting tokens to be unmasked at each step is a key component of the MDM sampling process, which significantly influences the overall generation performance. Let N denote the generation length. Since standard MDMs decode each position exactly once without remasking previously unmasked tokens, and ultimately ensure all positions are unmasked, the sequence of token indices selected across all sampling steps forms an ordered partition of the set $\mathcal{M} = \{1, \dots, N\}$: $\mathbf{O} = (O_1, \dots, O_n) = (\{i_{1,1}, \dots, i_{1,o_1}\}, \dots, \{i_{n,1}, \dots, i_{n,o_n}\})$, where $\mathbf{O}$ denotes a token sampling order, O_t denotes the indices of all o_t selected positions at step t , i.e., $i_{t,1}, \dots, i_{t,o_t}$. Given a pre-trained dLLM θ , a user-specified prompt c , our goal is to find the optimal order $\mathbf{O}^*$ that maximizes a given reward function $R(\cdot)$ (Liu et al., 2026):

$$\mathbf{O}^* = \arg \max_{\mathbf{O} \in \text{OPart}(\mathcal{M})} R(\mathbf{O}, c, \theta), \quad (5)$$

where $\text{OPart}(\mathcal{M})$ refers to the set of all possible ordered partition of $\mathcal{M}$.

NI Sampling. NI Sampling (Liu et al., 2026) is a representative method targeting at the token order optimization problem. Specifically, it trains a lightweight MLP S_ϕ parameterized by ϕ called neural indicator that makes a binary decision independently to determine whether a masked position should be unmasked at each sampling step. Specifically, at every step, the neural indicator gives a score s_ϕ^i at each masked position i with token candidate x^i :

$$s_\phi^i = S_\phi(x^i, h^i, \text{TOP}_K(\text{logits}^i)), \quad (6)$$

where h^i denotes the last layer hidden states at position i , and $\text{TOP}_K(\text{logits}^i)$ refers to the top- k predicted logits at position i . At step j , all masked positions whose indicator score is larger than a pre-defined threshold ϵ_ϕ will be unmasked:

$$O_t = \{i | s_\phi^i > \epsilon_\phi, i \notin \bigcup_{j=1}^{t-1} O_j\}, \quad (7)$$

Regarding the training stage, NI Sampling introduce *trajectory-preserving-order* to assign each masked token a binary label as the training signal. Details can be found in App. A

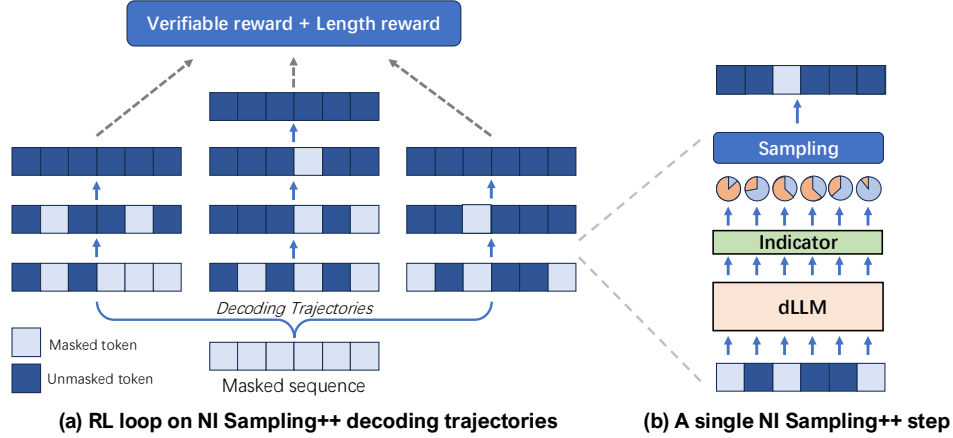


Figure 1: Overview of **NI Sampling++** training pipeline. (a) The reinforcement learning loop. For each input query, **NI Sampling++** generates multiple trajectories guided by the neural indicator. These trajectories are then evaluated via a reward function to facilitate GRPO-based policy optimization. (b) The unmasking policy within a single **NI Sampling++** step. The neural indicator assigns a score to each masked token, representing its unmasking probability. For each position, the decision to unmask is determined stochastically and independently.

Limitations of NI Sampling. Despite outperforming empirical heuristics, NI Sampling still significantly lag behind the verified upper bounds of token order optimization. For instance, on the GSM8K dataset, NI Sampling achieves a $4.7\times$ speedup, whereas the ideal *trajectory-preserving-order* reaches $11.4\times$, and the more aggressive *final-results-preserving-order* even attains $36.8\times$ (Liu et al., 2026).

The limited performance of NI Sampling stems from a fundamental issue. The number of possible decoding orders (in Eq. (5)) is $\approx \frac{\sqrt{2\pi N}}{2\ln 2} \left(\frac{N}{e\ln 2}\right)^N$, which grows super-exponentially with N . Exploring the full space is intractable. Instead, NI Sampling considers only a single solution guided by the *trajectory-preserving order*, which is not necessarily optimal. Based on this insight, we propose **NI Sampling++**, which leverages RL to efficiently explore this large decision space and identify effective decoding policies.

3.2 MDP Reformulation and Policy Optimization

Our key insight is that the sampling process, guided by a parameterized neural indicator, can be naturally formulated as a Markov Decision Process (MDP). We firstly replace transition from the deterministic threshold-based sampling in Eq. (7) to a stochastic formulation, where a Bernoulli trial is independently performed at each masked position:

$$O_t = \{i | s_\phi^i > b_i, i \notin \bigcup_{j=1}^{t-1} O_j\}, \quad (8)$$

where $b_i \sim \mathcal{U}(0, 1)$ and is sampled independently for different position i . The score s_ϕ^i given by the neural indicator serves as the probability to unmask token x^i . Since every token is independent, the probability of any specific unmasking set O_t can be factorized as follows:

$$p(O_t) = \prod_{i \in O_t} s_\phi^i \prod_{i \notin \bigcup_{j=1}^t O_j} (1 - s_\phi^i). \quad (9)$$

Here, indices $i \in O_t$ denote the set of masked positions that are selected to be unmask at step j , indices that satisfy $i \notin \bigcup_{j=1}^t O_j$ represent the positions that remain masked. Under the assumption of independence, the probability of sampling a specific set O_t is formulated as the product of the individual probabilities for these two disjoint groups.

Based on above discussions, we can describe the sampling process as a MDP:

$$s_t = \mathbf{x}_t, \quad a_t = O_t, \quad \tau = \mathbf{O} = (O_1, \dots, O_n), \quad \pi_\phi(a_t|s_t) = p(O_t). \quad (10)$$

The policy gradient methods are then used to train the neural indicator. To ensure training stability, we employ an on-policy optimization scheme with a single-gradient update per rollout. Furthermore, we incorporate GRPO (Shao et al., 2024) to enhance training, which samples a group of trajectories for each input to compute their relative advantages. Notably, we modify the standard GRPO by simplifying the advantage to the deviation from the group mean, skipping the variance normalization step (Liu et al., 2025b):

$$\hat{A}_i = R_i - \text{mean}(\mathbf{R}). \quad (11)$$

Empirically, we find that this unnormalized formulation leads to superior final performance (Chen et al., 2025). The complete training objective to be maximized is given as follows:

$$\begin{aligned} \mathcal{L}_\phi &= \frac{1}{G} \sum_{i=1}^G \hat{A}_i \frac{1}{|l_{\tau_i}|} \sum_t \log \pi_\phi(a_t|s_t) \\ &= \frac{1}{G} \sum_{i=1}^G \hat{A}_i \frac{1}{|l_{\tau_i}|} \sum_t \left(\sum_{k \in O_{t,i}} \log s_\phi^k + \sum_{k \notin \bigcup_{j=1}^t O_{j,i}} \log(1 - s_\phi^k) \right), \end{aligned} \quad (12)$$

where G is the group size, i refers to the indices of trajectories in the group, j refers to previous sampling steps, k refers to the indices of tokens, and $|l_{\tau_i}|$ is a normalizing factor that equals to the total number of indicator scores that are involved in loss computation: $|l_{\tau_i}| = \sum_t (N - |\bigcup_{j=1}^{t-1} O_{j,i}|)$.

Reward function. The reward function should incentivize both solving the problem correctly and reducing the number of sampling steps. Therefore, it should integrate a verifiable reward r_v with an efficiency reward r_e . For the former, we evaluate the generated answers, assigning a reward of $+1$ for correct results and -1 for incorrect ones. For the latter, we design a truncated formulation: $r_e = -w * \max(|\mathbf{O}| - n_0)$, where $|\mathbf{O}|$ is the number of steps of the rollout trajectory, and n_0 and w are the predefined threshold and penalty weight per step, respectively. The intuition is to prevent the policy from collapsing into an extremely low-step regime, which typically leads to a significant drop in accuracy. Under such conditions, the inherent nature of GRPO further exacerbates this issue: if the verifiable rewards for all trajectories in a group are consistently -1 , the relative advantage will be driven solely by the efficiency reward. Consequently, the model would be trained to further reduce sampling steps. Eliminating efficiency reward by incorporating the step threshold can mitigate this issue.

3.3 Techniques for RL Training

In this section, we propose several techniques to improve **NI Sampling++**.

NI Sampling-based initialization. Training RL agents within expansive discrete action spaces often suffers from high variance and slow convergence when initialized from scratch. For RL in LLMs, it is standard practice to initialize the policy with a supervised fine-tuned (SFT) checkpoint (Ouyang et al., 2022). Following these settings, we initialize our neural indicator with the pre-trained weights from NI Sampling (Liu et al., 2026).

Mid-start rollout. Conventional RL methods typically sample trajectories from scratch and optimize the entire action sequence. To enhance training stability, we introduce a “mid-start rollout” strategy. Specifically, we first generate a set of high-quality reference trajectories for each problem in the dataset using a confidence thresholding approach. During training, we randomly sample an intermediate timestep on the trajectory as the starting point, from which the neural indicator completes the remaining generation. Optimization is then performed exclusively on this latter portion of the trajectory. Empirical results in Fig. 4b show that mid-start rollout outperforms sampling from scratch.

Scaled architecture. NI Sampling employs a MLP as the indicator architecture. Such indicator processes each token independently at every step, which limits its ability to capture inter-token dependencies beyond the information contained in the last layer hidden states.

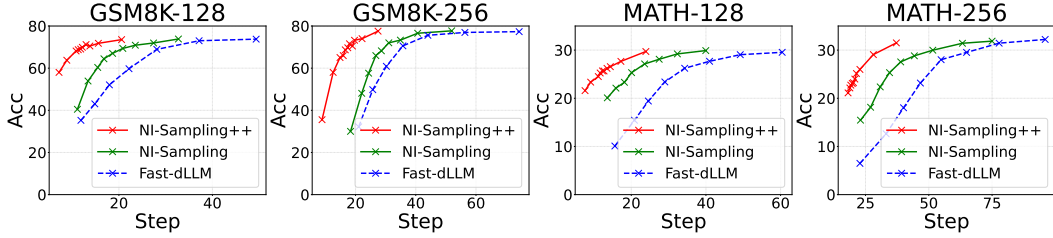


Figure 2: Trade-off curves between accuracy and steps with LLaDA-8B-Instruct model on math datasets.

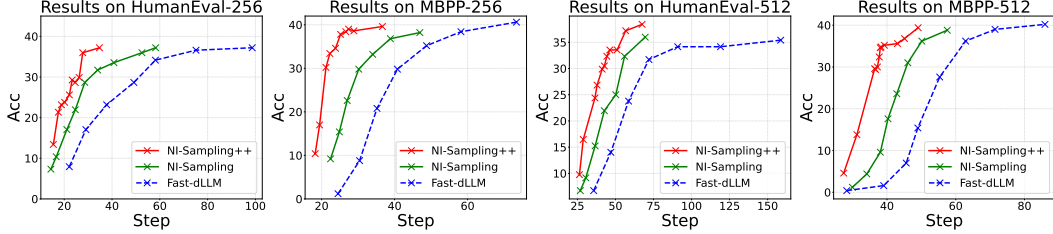


Figure 3: Trade-off curves between accuracy and steps with LLaDA-8B-Instruct model on code generation datasets.

Furthermore, MLP structure usually exhibits poor scalability compared to transformer-based models. To address these limitations, we replace the MLP with a two-layer transformer. Each layer shares the same architecture with that in the pre-trained diffusion LLM. During inference, we construct a input sequence consisting of the sampled tokens, last-layer hidden states, and top-K logits across all generation positions. Notably, the input sequence also incorporates positions that are already unmasked to provide stable context. This sequence is then processed by the transformer indicator to produce a sequence of unmasking scores $(s_\phi^1, \dots, s_\phi^N)$. By setting the indicator precision to BF16, we validate that the scaled neural indicator introduces minimal inference cost, with results shown in [Sec. B.2](#).

4 Experiments

4.1 Experimental Setup

Base models and benchmarks. We primarily conduct our experiments with LLaDA-8B-Instruct and LLaDA-1.5 models. Following the experimental settings of [Wu et al. \(2025\)](#), we choose the test set of GSM8K (5-shot) ([Cobbe et al., 2021](#)), MATH (4-shot) ([Hendrycks et al., 2021](#)), HumanEval (0-shot) ([Chen et al., 2021](#)), and MBPP (0-shot) ([Austin et al., 2021](#)) as evaluation datasets, which reflect the model’s reasoning capability on mathematical and code generation tasks. Furthermore, to evaluate the generalization stability across different generation lengths, we incorporate varying generation lengths of 128, 256, and 512 tokens into our experiments.

Baselines and Evaluation. We choose NI Sampling as our main baseline. We also compare **NI Sampling++** with confidence threshold sampling ([Wu et al., 2025](#)) as a representative heuristic-based method. For our main results in [Tab. 1](#), we employ 0.9 as the threshold for confidence threshold sampling. For NI Sampling and **NI Sampling++**, we unmask all tokens whose indicator scores exceed a predefined threshold. To ensure fair efficiency comparison, we adjust the indicator threshold for both NI Sampling and **NI Sampling++** to make sure they achieve similar task performance with full-step method. We then report the acceleration ratios of these methods compared with the full-step method. Note that the additional latency of neural indicator has already been considered. Furthermore, by adjusting this threshold, we are able to derive a smooth step-accuracy trade-off curves, which are presented alongside two baseline methods in [Figs. 2](#) and [3](#), providing a comprehensive view of the model’s efficiency-performance landscape.

4.2 Main Results

We list the main results of **NI Sampling++** in [Tab. 1](#) and demonstrate the step-accuracy trade-off curve in [Figs. 2](#) and [3](#). Key takeaways are summarized below.

Table 1: Results of **NI Sampling++**, NI Sampling and other baseline methods

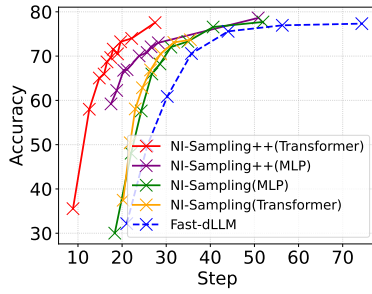
Dataset	Method	LLaDA-8B-Instruct			LLaDA 1.5		
		Acc	Steps	Token/s	Acc	Steps	Token/s
GSM8K-128	Full	73.92	128	8.95	76.65	128	8.95
	Threshold	73.77	49.35	23.22 (2.6 $\times$)	75.82	47.44	24.16 (2.7 $\times$)
	NI Sampling	73.69	34.36	32.59 (3.6 $\times$)	76.19	29.89	37.46 (4.2 $\times$)
	NI Sampling++	73.54	20.57	54.23 (6.1 $\times$)	76.19	19.98	55.83 (6.2 $\times$)
GSM8K-256	Full	77.63	256	8.38	81.05	256	8.38
	Threshold	77.33	74.34	28.84 (3.4 $\times$)	81.58	72.92	29.40 (3.5 $\times$)
	NI Sampling	77.18	50.97	40.88 (4.9 $\times$)	80.67	53.59	38.88 (4.6 $\times$)
	NI Sampling++	77.56	27.47	75.86 (9.1 $\times$)	80.29	27.54	75.67 (9.0 $\times$)
MATH-128	Full	29.64	128	11.12	31.03	128	11.12
	Threshold	29.69	60.43	23.56 (2.1 $\times$)	30.93	58.71	24.25 (2.2 $\times$)
	NI Sampling	30.07	39.01	35.37 (3.2 $\times$)	31.41	39.65	34.80 (3.1 $\times$)
	NI Sampling++	29.71	23.81	57.85 (5.2 $\times$)	30.80	23.18	59.42 (5.3 $\times$)
MATH-256	Full	31.89	256	10.41	33.18	256	25.0
	Threshold	31.68	95.98	27.75 (2.7 $\times$)	33.18	95.41	27.92 (2.7 $\times$)
	NI Sampling	31.67	61.92	41.52 (4.0 $\times$)	32.98	69.67	36.90 (3.5 $\times$)
	NI Sampling++	31.50	37.18	69.21 (6.7 $\times$)	33.01	36.71	70.10 (6.7 $\times$)
HumanEval-256	Full	37.80	256	20.12	43.90	256	20.12
	Threshold	37.20	98.66	52.21 (2.6 $\times$)	42.68	100.7	51.15 (2.5 $\times$)
	NI Sampling	37.20	54.75	89.62 (4.5 $\times$)	42.68	64.40	76.19 (3.8 $\times$)
	NI Sampling++	37.20	34.73	138.97 (6.9 $\times$)	42.68	49.93	96.67 (4.8 $\times$)
HumanEval-512	Full	35.37	512	14.93	40.85	512	14.93
	Threshold	35.37	158.6	48.18 (3.2 $\times$)	39.02	158.4	48.24 (3.2 $\times$)
	NI Sampling	36.59	88.45	81.83 (5.5 $\times$)	39.63	105.3	68.73 (4.6 $\times$)
	NI Sampling++	37.20	56.73	125.79 (8.4 $\times$)	40.24	63.01	113.25 (7.6 $\times$)
MBPP-256	Full	41.20	256	22.42	41.80	256	22.42
	Threshold	40.60	73.39	78.21 (3.5 $\times$)	40.60	75.67	75.85 (3.4 $\times$)
	NI Sampling	38.20	46.91	119.15 (5.3 $\times$)	40.20	48.65	114.89 (5.1 $\times$)
	NI Sampling++	39.00	27.39	194.84 (8.7 $\times$)	38.80	28.92	184.53 (8.2 $\times$)
MBPP-512	Full	38.4	512	15.17	41.2	512	15.17
	Threshold	40.20	85.59	90.77 (6.0 $\times$)	38.20	95.32	81.51 (5.4 $\times$)
	NI Sampling	38.80	57.52	132.66 (8.7 $\times$)	37.60	62.94	121.23 (8.0 $\times$)
	NI Sampling++	39.20	49.06	147.65 (9.7 $\times$)	36.00	39.02	185.65 (12.2 $\times$)

NI Sampling++ achieves significant speedups while maintaining full-step performance. As shown in Tab. 1, **NI Sampling++** attains up to a 9.7 $\times$ acceleration over full-step generation, while preserving comparable task performance. Compared to the strong baseline NI Sampling, **NI Sampling++** also delivers substantially better inference efficiency. For example, on GSM8K-256, **NI Sampling++** is over 1.85 $\times$ faster than NI Sampling while maintaining similar performance.

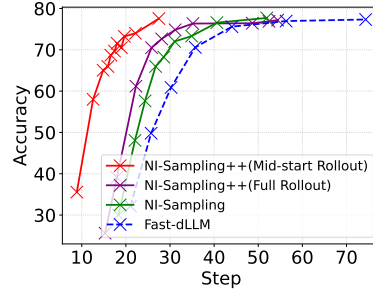
In some cases, **NI Sampling++** even substantially outperforms full-step sampling in task performance. For example, on HumanEval-512 with LLaDA-8B-Instruct, performance improves from 35.37 to 37.20 while achieving an 8.4 $\times$ speedup. This gain can be attributed to the benefits of RL training: unlike NI Sampling, where performance is constrained by the trajectory-preserving order, and full-step sampling, which corresponds to a single solution in the order space, RL enables the neural indicator to explore a broader space of token orders and potentially discover better solutions.

NI Sampling++ enables a flexible and effective trade-off between inference steps and accuracy. As shown in Figs. 2 and 3, by simply adjusting the neural indicator threshold, **NI Sampling++** can operate across a wide range of inference steps. This allows users to flexibly select an operating point based on their computational budget and desired performance. Notably, **NI Sampling++** consistently achieves a better trade-off between inference steps and accuracy than both NI Sampling and threshold sampling.

NI Sampling++ generalizes well across datasets, models, and generation lengths. As shown in Tab. 1 and Figs. 2 and 3, although **NI Sampling++** is trained on a single dataset (GSM8K for math and Kodcode for code), a single model (LLaDA-8B-Instruct), and a fixed generation length (256), it generalizes effectively to other datasets, models, and generation lengths. This suggests that the neural indicator learns broadly applicable patterns rather than overfitting to the training setup. It also highlights the practical value of **NI Sampling++** in real-world deployments, where a single model must often support diverse workloads.



(a) Architecture (Transformer vs. MLP)



(b) Rollout Strategy (Mid-start vs. Full)

Figure 4: Ablation studies on GSM8K-256. (a) Comparison of different indicator architectures. (b) Impact of the mid-start rollout strategy on training stability and performance.

4.3 Ablation Studies

In this section, we study two key design choices in the framework of **NI Sampling++**: indicator architecture and strategy for rollout.

Indicator architecture: transformer v.s. MLP. We train an MLP indicator, keeping all other experimental settings consistent with our main experiment. We show the accuracy-step trade-off curve in Fig. 4a. While **NI Sampling++** using an MLP indicator still outperforms both NI Sampling and the confidence threshold baseline, the transformer-based variant significantly surpasses all other configurations. Notably, this superiority is achieved even when initializing from a checkpoint (labeled as NI Sampling (Transformer) in Fig. 4a) comparable to that of MLP. This demonstrates that the transformer serves as a more suitable architecture for RL in token order optimization.

Strategy for rollout: mid-start rollout v.s. full rollout. To validate the effectiveness of our mid-start rollout strategy, we train an indicator using the conventional approach where the trajectory is generated from scratch. We denote this as full rollout. We reduce the learning rate from $1e-5$ to $5e-6$ for this variant, as we observed that higher learning rates frequently led to training collapse in the full rollout setting. All other configurations remain the same as our main experiments. As illustrated in Fig. 4b, the full rollout method yields only marginal improvements over the NI Sampling baseline, with gains primarily restricted to low-step regions. In contrast, our mid-start rollout consistently dominates the entire trade-off curve of all baselines, significantly outperforming the full rollout across all sampling budgets.

5 Related Works: RL for Diffusion Language Models

Reinforcement Learning (RL) has become a pivotal post-training paradigm for enhancing LLMs (Achiam et al., 2023; Guo et al., 2025). In the context of MDMs, LLaDA-1.5 (Zhu et al., 2025) proposes VRPO to address high variance issue when applying preference optimization to dLLMs. Huang et al. (2025) also adopts an additional network to decide the unmasking pattern and uses policy optimization to jointly optimize the dLLM and the unmasking decision network. Ni et al. (2026) identifies the flexibility trap when training dLLMs with RL and proposes to use fixed AR order when training. These methods adopt only verifiable rewards without optimizing the efficiency.

Several studies have also utilized RL to optimize the sampling efficiency of MDMs. A concurrent and closely related work is Jazbec et al. (2025), which similarly employs RL to optimize an unmasking planner. However, their planner relies solely on token confidence and binary mask sequences as inputs, which fails to incorporate context information, and lacks a proper initialization strategy. Therefore, they can only outperform baseline methods under extremely restrictive step budgets. Another concurrent work dUltra (Chen et al., 2025) also introduces an unmasking planner with efficiency-aware rewards. However, this approach jointly optimizes the dLLM, which incurs prohibitive memory overhead and limit flexibility to apply to other models. In contrast, our method can be simply

plugged in to pretrained dLLM without requiring finetuning, achieving substantial efficiency improvements with significantly lower training costs.

6 Conclusion

In this paper, we propose **NI Sampling++**, a RL framework for token order optimization in dLLMs. We reformulate the sampling process as a MDP and adopt GRPO to train the policy given by a neural indicator. Without training the dLLM, we significantly outperforms existing token order optimization method based on supervised training and other heuristic-based methods. This work further leverages the intrinsic parallelism of dLLMs, pushing their inference efficiency to a superior frontier.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.
- Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, et al. Llada2. 0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.
- Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11315–11325, 2022.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Shirui Chen, Jiantao Jiao, Lillian J Ratliff, and Banghua Zhu. dultra: Ultra-fast diffusion language models via reinforcement learning. *arXiv preprint arXiv:2512.21446*, 2025.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Hengyu Fu, Baihe Huang, Virginia Adams, Charles Wang, Venkat Srinivasan, and Jiantao Jiao. From bits to rounds: Parallel decoding with exploration for diffusion language models. *arXiv preprint arXiv:2511.21103*, 2025.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Zhengfu He, Tianxiang Sun, Kuanning Wang, Xuanjing Huang, and Xipeng Qiu. Diffusionbert: Improving generative masked language models with diffusion models. *arXiv preprint arXiv:2211.15029*, 2022.
- 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. *arXiv preprint arXiv:2103.03874*, 2021.
- Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in neural information processing systems*, 34:12454–12465, 2021.

- Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. Reinforcing the diffusion chain of lateral thought with diffusion language models. *arXiv preprint arXiv:2505.10446*, 2025.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.
- Metod Jazbec, Theo X Olausson, Louis Béthune, Pierre Ablin, Michael Kirchhof, João Monteiro, Victor Turrise, Jason Ramapuram, and Marco Cuturi. Learning unmasking policies for diffusion language models. *arXiv preprint arXiv:2512.09106*, 2025.
- Wonjun Kang, Kevin Galim, Seunghyuk Oh, Minjae Lee, Yuchen Zeng, Shuibai Zhang, Coleman Hooper, Yuezhou Hu, Hyung Il Koo, Nam Ik Cho, et al. Parallelbench: Understanding the trade-offs of parallel decoding in diffusion llms. *arXiv preprint arXiv:2510.04767*, 2025.
- Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025a.
- Seo Hyun Kim, Sunwoo Hong, Hojung Jung, Youngrok Park, and Se-Young Yun. Klass: Kl-guided fast inference in masked diffusion models. *arXiv preprint arXiv:2511.05664*, 2025b.
- Sanghyun Lee, Seungryong Kim, Jongho Park, and Dongmin Park. Lookahead unmasking elicits accurate decoding in diffusion language models. *arXiv preprint arXiv:2511.05563*, 2025.
- José Lezama, Huiwen Chang, Lu Jiang, and Irfan Essa. Improved masked image generation with token-critic. In *European Conference on Computer Vision*, pp. 70–86. Springer, 2022a.
- Jose Lezama, Tim Salimans, Lu Jiang, Huiwen Chang, Jonathan Ho, and Irfan Essa. Discrete predictor-corrector diffusion models for image synthesis. In *The Eleventh International Conference on Learning Representations*, 2022b.
- Enshu Liu, Xuefei Ning, Yu Wang, and Zinan Lin. Distilled decoding 1: One-step sampling of image auto-regressive models with flow matching. *arXiv preprint arXiv:2412.17153*, 2024.
- Enshu Liu, Qian Chen, Xuefei Ning, Shengen Yan, Guohao Dai, Zinan Lin, and Yu Wang. Distilled decoding 2: One-step sampling of image auto-regressive models with conditional score distillation. *arXiv preprint arXiv:2510.21003*, 2025a.
- Enshu Liu, Xuefei Ning, Yu Wang, and Zinan Lin. Ni sampling: Accelerating discrete diffusion sampling by token order optimization. In *The Fourteenth International Conference on Learning Representations*, 2026.
- 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. *arXiv preprint arXiv:2503.20783*, 2025b.
- Amr Mohamed, Yang Zhang, Michalis Vazirgiannis, and Guokan Shang. Fast-decoding diffusion language models via progress-aware confidence schedules. *arXiv preprint arXiv:2512.02892*, 2025.
- Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, et al. The flexibility trap: Why arbitrary order limits reasoning potential in diffusion language models. *arXiv preprint arXiv:2601.15165*, 2026.
- Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. Scaling up masked diffusion models on text. *arXiv preprint arXiv:2410.18514*, 2024.

- 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. *arXiv preprint arXiv:2502.09992*, 2025.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Machel Reid, Vincent J Hellendoorn, and Graham Neubig. Diffuser: Discrete diffusion via edit-based reconstruction. *arXiv preprint arXiv:2210.16886*, 2022.
- Subham Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. Score-based continuous-time discrete diffusion models. *arXiv preprint arXiv:2211.16750*, 2022.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.
- Qingyan Wei, Yaojie Zhang, Zhiyuan Liu, Dongrui Liu, and Linfeng Zhang. Accelerating diffusion large language models with slowfast: The three golden principles. *arXiv e-prints*, pp. arXiv–2506, 2025.
- Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.
- Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.

A More Details of NI Sampling

As discussed in [Sec. 3.1](#), NI Sampling ([Liu et al., 2026](#)) introduce *trajectory-preserving-order* to label all masked tokens at each step as supervisory signal to train the neural indicator. The core idea of *trajectory-preserving-order* is to perform step compression by merging adjacent steps in a given reference trajectory, under the constraint that the final generated token at each position strictly remain unchanged. Specifically, suppose the reference trajectory has

Table 2: Inference time (ms) of LLaDA-8B-Instruct model and the neural indicator.

Dataset	Model	Indicator
GSM8K-128	111.7	3.05
GSM8K-256	119.4	3.45
MATH-128	89.9	3.03
MATH-256	96.1	3.38
HumanEval-256	49.7	3.34
HumanEval-512	67.0	4.75
MBPP-256	44.6	3.37
MBPP-512	65.9	4.78

an order $O = (O_1, \dots, O_n) = (\{i_{1,1}, \dots, i_{1,o_1}\}, \dots, \{i_{n,1}, \dots, i_{n,o_n}\})$. At step k , we can safely merge all tokens in O_k and O_{k+1} under the following conditions:

$$\forall i \in O_{k+1}, \arg \max_{j \in \{1, \dots, V\}} p_\theta(x_0^i = j | x_k) = x_*^i, \quad (13)$$

where x_k is the partially masked sequence before tokens in O_k are sampled, x_*^i is the sampled token at position i after O_{k+1} is unmasked. This merging operation can be viewed as simultaneously executing steps k and $k+1$ at the original step k . Since the sampled tokens remain identical to those in the reference trajectory, this merging operation introduces no discrepancy in the subsequent unmasking process. It can also be conducted more than one times if multiple steps can be merged.

NI Sampling firstly generate reference trajectories using confidence threshold sampling. At each step k , NI Sampling labels all masked tokens with a binary value using *trajectory-preserving-order*, and then trains the neural indicator to classify each token.

B Additional Experiments

B.1 More Implementation Details

We developed two specialized neural indicators tailored for mathematical reasoning and code generation, respectively. The mathematical indicator was trained on the GSM8K training set (7,473 samples), while the code-specific indicator utilized the Kodcode training set (9,240 samples). We set the generation length as 256 for rollout. Both models are initialized from a shared indicator checkpoint pre-trained via NI Sampling. To generate reference trajectories, we employed confidence thresholding sampling with the threshold set at 0.8. During training, we employ a group size of 8 for GRPO and an update frequency of 256 (i.e., update the model for every 256 inputs, equivalent to 256*8 rollouts). For mathematical indicator, we use a learning rate of 1e-5, step threshold n_0 of 17, and penalty per step w (see Sec. 3.2 for the definition of n_0 and w) as 0.3. For code generation indicator, we use a learning rate of 2e-6, step threshold n_0 of 25, and penalty per step w (see Sec. 3.2 for the definition of n_0 and w) as 0.05. We train all neural indicators with LLaDA-8B-Instruct, and evaluate them with both LLaDA-8B-Instruct and LLaDA-1.5.

B.2 Cost of the Neural Indicator

We employ a stack of two Transformer layers as the indicator, where each layer shares an identical architecture with a single layer of the base dLLM. The parameter size of the indicator is 584M, containing 1/16 of the parameters in the pre-trained dLLM. We provide the inference latency and the GPU memory cost of both neural indicator and pre-trained dLLM in Tab. 2 and Tab. 3, respectively. Notably, despite the increased scale of the transformer-based indicator compared with NI Sampling, its inference overhead and memory consumption remain a small fraction.

Table 3: Peak GPU memory consumption of the LLaDA-8B-Instruct model with and without the neural indicator.

Dataset	Base Model Memory (GB)	Indicator Memory (GB)
GSM8K-128	21.4	23.0
GSM8K-256	21.8	23.3
MATH-128	18.8	20.5
MATH-256	19.8	21.0
HumanEval-256	16.2	17.6
HumanEval-512	16.8	18.2
MBPP-256	16.5	18.0
MBPP-512	16.9	18.8

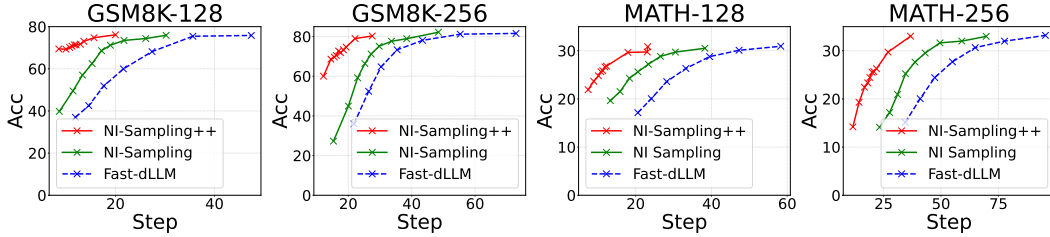


Figure 5: Trade-off curves between accuracy and steps with LLaDA-1.5 model on math datasets.

Table 4: Diversity evaluation results on HumanEval-256 dataset. "Slope" stands for the slope of the fitted line of pass@k with respect to $\log k$.

	NFE	Pass@1	Pass@2	Pass@4	Pass@8	Pass@16	Slope $\uparrow$
Full-step	256	36.99	44.69	51.43	57.81	63.76	6.67
Threshold	99.13	36.82	44.35	51.02	57.40	63.10	6.56
NI Sampling	66.72	36.70	44.38	51.34	58.03	64.33	6.89
NI Sampling++	49.31	36.09	43.97	50.97	57.60	64.63	7.07

B.3 Trade-off Curve with LLaDA-1.5 Model

We further provide the accuracy-step trade-off curves with LLaDA-1.5 models in Figs. 5 and 6. Similar conclusion holds for LLaDA-1.5 model, that **NI Sampling++** achieves better trade-off compared to all baseline methods.

B.4 Results with Stochastic Sampling

We further investigate the compatibility of **NI Sampling++** with stochastic sampling of the dLLM and evaluate the resulting generation diversity. We compute Pass@k on the HumanEval-256 dataset for various values of k and perform a linear regression between the accuracy and $\log k$. The resulting slope is utilized as a quantitative metric to evaluate diversity, with all results summarized in Tab. 4. As demonstrated, **NI Sampling++** consistently outperforms all baseline methods in both generation diversity and sampling efficiency.

B.5 Results with LLaDA2.0-mini Model

To evaluate the scaling ability of **NI Sampling++**, we apply it to LLaDA2.0-mini (Bie et al., 2025) model with a parameter size of 16B. Our results are demonstrated in Fig. 7 and Tab. 5.

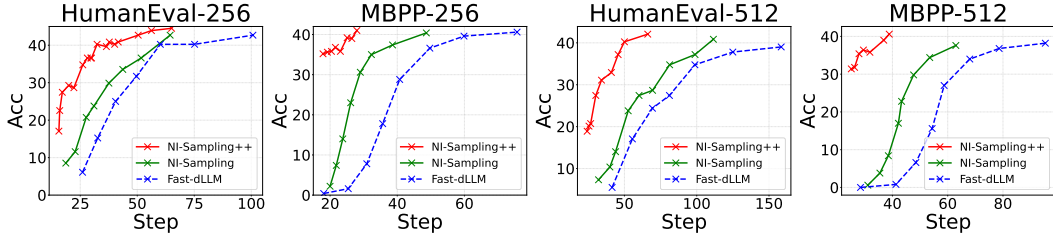


Figure 6: Trade-off curves between accuracy and steps with LLaDA-1.5t model on code generation datasets.

Table 5: Sampling performance on LLaDA2.0-mini.

Method	GSM8K-256		MATH500-256	
	Acc	Steps	Acc	Steps
Threshold	88.47	79.40	41.40	124.52
NI Sampling	88.40	49.26	39.80	83.76
NI Sampling++ (ours)	88.47	40.06	40.40	66.85

We can see that our NI Sampling++ can still outperform baseline methods NI Sampling and confidence threshold sampling.

B.6 Comparison with Baselines

In this section, we compare our NI Sampling++ with two similar works, [Jazbec et al. \(2025\)](#) and [Chen et al. \(2025\)](#).

Regarding [Jazbec et al. \(2025\)](#), as discussed in [Sec. 5](#), their planner only takes token confidence and binary mask sequences as inputs, and lacks proper initialization. Since their official code (<https://github.com/apple/ml-rl-dllm>) does not provide trained checkpoints, we reproduced the training process following their workflow and evaluated it under our test implementation. Results are shown at [Fig. 8](#) and [Fig. 9](#). As can be seen, our performance far exceeds this method.

Regarding [Chen et al. \(2025\)](#), we first would like to point out that [Chen et al. \(2025\)](#) requires training an entire dLLM, which incurs a massive GPU memory overhead. We find that training an 8B LLaDA model with [Chen et al. \(2025\)](#) requires approximately 140G of GPU memory. In contrast, our method only requires GPU memory for dLLM inference and training the lightweight neural indicator, which takes around 27G, making it much more convenient to use than [Chen et al. \(2025\)](#). Nevertheless, [Chen et al. \(2025\)](#) also utilizes several design choices for RL. To conduct a thorough and fair comparison, we can compare these with our design choices, which include:

- **Advantage clip:** [Chen et al. \(2025\)](#) clips all advantages smaller than 0 to 0, whereas we do not apply this method.
- **Linear efficiency reward without threshold:** [Chen et al. \(2025\)](#)’s efficiency reward is fully linear, whereas we set a step threshold to clip this term, applying no penalty for steps less than the threshold and a linear penalty for steps greater than the threshold.
- **On-policy distillation:** [Chen et al. \(2025\)](#) uses a well-trained auto-regressive LLM to calculate the log-likelihood of the generated results as an additional reward term, whereas we do not apply this method.

We integrate these techniques into our RL training framework and present the experimental results below.

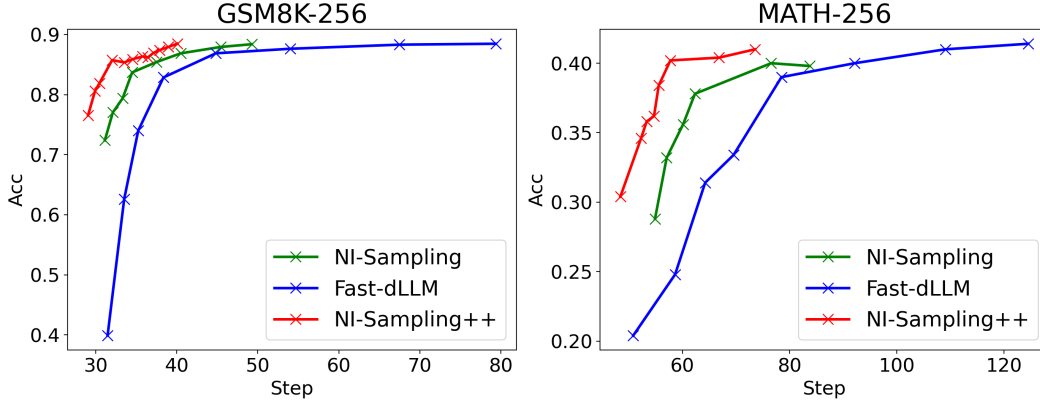


Figure 7: Trade-off curves between accuracy and steps with LLaDA2,0-mini model.

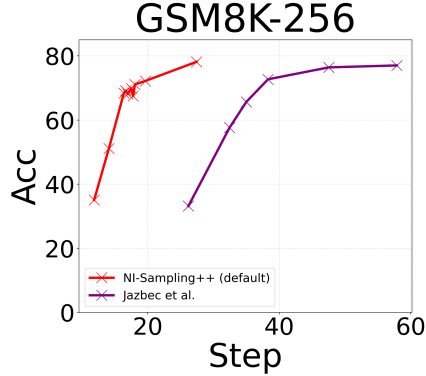


Figure 8: Trade-off curves of NI Sampling++ and Jazbec et al. (2025) on GSM8K dataset.

- **Advantage clip:** We conduct experiments on GSM8K-256 dataset, with step-acc trade-off curve shown in Fig. 10. As shown in the figure, using advantage clipping does not bring any performance improvement in our scenario.
- **Linear efficiency reward without threshold:** We set the step threshold to 0 and evaluated it on HumanEval and MBPP-Instruct. We found that if we use the efficiency reward weight of 0.05 from our setting, the indicator quickly collapses to sampling all tokens at once, which aligns with our intuition in Sec. 3.2. We therefore adjust the efficiency reward weight to 0.025. Results are shown in Fig. 11. As shown in the figure and discussed before, not using a step threshold leads to slightly better performance when the step budget is low, but it performs far worse than using a step threshold when the step budget is high. Generally, we believe that using a step threshold to truncate the linear efficiency reward is a better choice.
- **On-policy distillation:** Following Chen et al. (2025), we use Qwen-2.5-Coder-7B as the teacher AR model to perform on-policy distillation, with results shown in Fig. 12. As shown in the figure, using on-policy distillation does not yield better performance compared to our default setting. Considering the simplicity of the method and training efficiency, we do not employ this approach.

B.7 Degree of Disorder Analysis

To understand the underlying reason why our method achieves fewer steps, we additionally evaluate the degree of disorder across different sampling methods. Specifically, we record the index of each token at the time it is sampled, and then calculate the number of disordered pairs in the final resulting index list. For confidence threshold sampling, tokens sampled

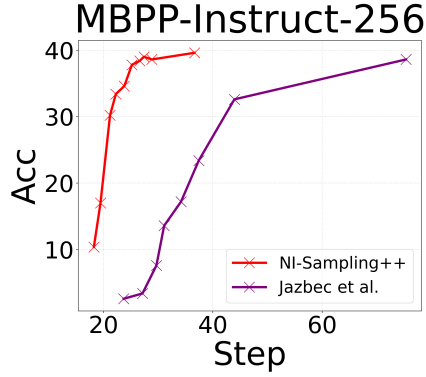


Figure 9: Trade-off curves of NI Sampling++ and Jazbec et al. (2025) on MBPP dataset.

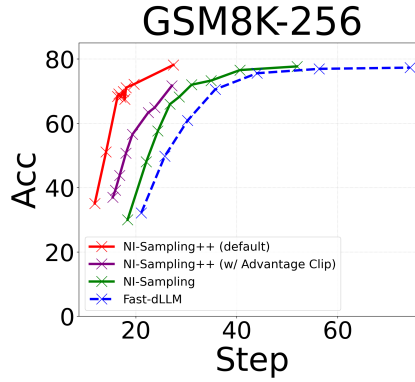


Figure 10: Trade-off curves of default setting and advantage clip.

Table 6: Comparison of sampling accuracy, sampling steps, and token-order disorder on GSM8K-256.

Method	Acc	Steps	Avg. Disordered Pairs
Full-step	77.63	256	746.24
Threshold	77.33	74.34	780.72
NI Sampling	77.18	50.97	1667.29
NI Sampling++ (ours)	77.56	27.47	1532.66

at the same time have their respective indices determined in descending order of their confidence. For **NI Sampling++** and NI Sampling, tokens sampled at the same time have their respective indices determined in descending order of their indicator scores. Results are shown at Tab. 6. As can be seen, the average number of disordered pairs for **NI Sampling++** and NI Sampling is significantly higher than that of full-step and confidence threshold sampling. This demonstrates that the reason why token order optimization is much more effective is that it samples many late-positioned tokens ahead of time, fully leveraging the capability of dLLMs to support arbitrary order.

C Theoretical Discussions

C.1 Comparison Between Bernoulli Sampling and Deterministic Sampling

We further clarify the connection between the stochastic Bernoulli policy used during RL training and the deterministic threshold rule used during inference. At sampling step t , let $\mathcal{M}_t$ denote the set of currently masked positions, with $m = |\mathcal{M}_t|$, and let $\mathbf{s}_\phi = (s_\phi^1, \dots, s_\phi^m)$

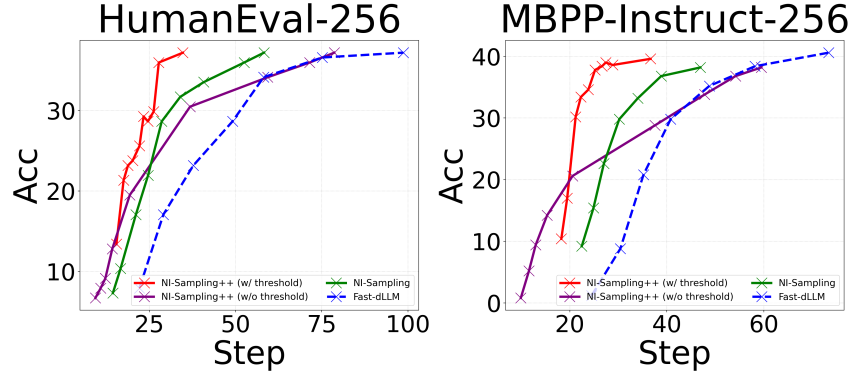


Figure 11: Trade-off curves of default setting and linear efficiency reward without threshold.

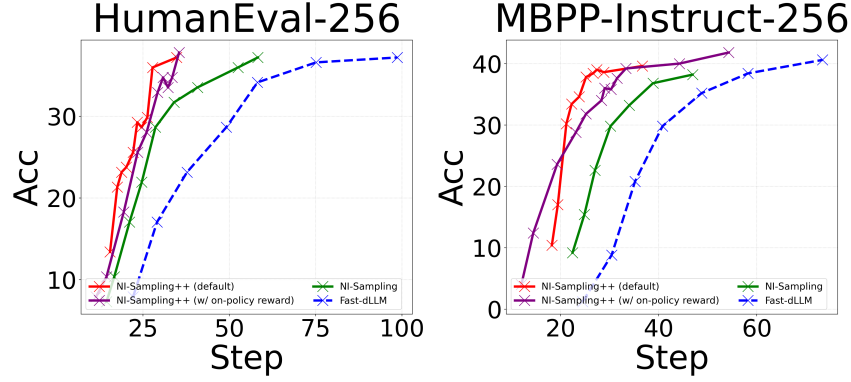


Figure 12: Trade-off curves of default setting and on policy distillation.

denote the indicator scores. NI Sampling deterministically selects the unmasking subset according to a predefined threshold ϵ_ϕ :

$$\mathcal{O}_{\epsilon_\phi} = \left\{ i \in \mathcal{M}_t \mid s_\phi^i > \epsilon_\phi \right\}. \quad (14)$$

For a fixed score vector $\mathbf{s}_\phi$, varying ϵ_ϕ can only generate a nested chain of at most $m + 1$ candidate subsets, whereas the full combinatorial action space contains 2^m possible subsets.

In contrast, our Bernoulli policy independently samples whether each masked position should be unmasked. The probability of selecting a particular subset $\mathcal{O}_t \subseteq \mathcal{M}_t$ is

$$p_\phi(\mathcal{O}_t) = \prod_{i \in \mathcal{O}_t} s_\phi^i \prod_{i \in \mathcal{M}_t \setminus \mathcal{O}_t} (1 - s_\phi^i). \quad (15)$$

As long as $s_\phi^i \in (0, 1)$ for every $i \in \mathcal{M}_t$, every possible subset has non-zero probability. Therefore, Bernoulli sampling provides full-support exploration over the combinatorial token-order space, while deterministic thresholding is restricted to the one-dimensional chain induced by the ranking of indicator scores. This distinction is important because the optimal unmasking subset does not necessarily correspond to a threshold cut of the current scores.

The independent Bernoulli formulation also makes policy-gradient optimization tractable. Specifically, the log-probability of an action decomposes into independent binary terms:

$$\log \pi_\phi(a_t \mid s_t) = \sum_{i \in \mathcal{O}_t} \log s_\phi^i + \sum_{i \in \mathcal{M}_t \setminus \mathcal{O}_t} \log (1 - s_\phi^i), \quad (16)$$

allowing its gradient with respect to ϕ to be directly obtained by backpropagation.

Although stochastic Bernoulli sampling is used during training, our framework remains compatible with deterministic thresholding at inference. For any threshold-selected action

$\mathcal{O}_{\epsilon_\phi}$, its probability under the Bernoulli training policy is

$$p_\phi(\mathcal{O}_{\epsilon_\phi}) = \prod_{i:s_\phi^i > \epsilon_\phi} s_\phi^i \prod_{i:s_\phi^i \leq \epsilon_\phi} (1 - s_\phi^i) > 0. \quad (17)$$

Hence, deterministic threshold inference always selects an in-support action of the training policy rather than an out-of-distribution action. Changing ϵ_ϕ can thus be viewed as a conservative calibration of the learned Bernoulli scores, trading decoding speed for generation quality.

Moreover, threshold inference can be interpreted as a MAP decision under a logit-shifted Bernoulli policy. Defining

$$\tilde{s}_\phi^i = \sigma \left(\text{logit}(s_\phi^i) - \text{logit}(\epsilon_\phi) \right), \quad (18)$$

we have

$$\tilde{s}_\phi^i > \frac{1}{2} \iff s_\phi^i > \epsilon_\phi. \quad (19)$$

Therefore, applying the threshold ϵ_ϕ is equivalent to selecting the MAP action after applying a logit shift that penalizes aggressive token selection.

C.2 Analysis of Modified GRPO

We further explain why we remove the group-wise standard-deviation normalization from the GRPO advantage. In our setting, the reward of trajectory τ_i is decomposed as

$$R_i = r_v(x_i) + r_e(\tau_i), \quad (20)$$

where $r_v(x_i) \in \{+1, -1\}$ is the verifier reward and

$$r_e(\tau_i) = -w \max(T_i - n_0, 0) \quad (21)$$

is the truncated efficiency reward. Here, T_i denotes the number of sampling steps, n_0 is the step threshold, and w controls the strength of the efficiency penalty.

Standard GRPO normalizes the group-relative advantage by the reward standard deviation:

$$A_i^{\text{std}} = \frac{R_i - \bar{R}}{\sigma_R}. \quad (22)$$

Although this normalization is useful when reward scales vary substantially, it introduces an undesirable effect under our reward design, where the verifier reward should dominate the efficiency reward.

Consider a group in which all trajectories obtain the same verifier outcome, such that

$$r_v(x_i) - \bar{r}_v = 0. \quad (23)$$

Let

$$L_i = \max(T_i - n_0, 0). \quad (24)$$

The variation of the reward within this group then comes entirely from the efficiency term:

$$r_e(\tau_i) = -w L_i. \quad (25)$$

Consequently, the normalized advantage becomes

$$A_i^{\text{std}} = \frac{-w(L_i - \bar{L})}{w\sigma_L} = -\frac{L_i - \bar{L}}{\sigma_L}. \quad (26)$$

Crucially, the efficiency weight w is completely cancelled by the normalization. Therefore, even when w is intentionally chosen to make the efficiency reward much weaker than the verifier reward, standard-deviation normalization converts small step-count differences into an $\mathcal{O}(1)$ policy-gradient signal. In particular, when all sampled trajectories are incorrect, this may aggressively optimize for shorter trajectories rather than encouraging exploration toward trajectories with higher correctness.

We instead use the mean-centered but unnormalized advantage

$$\hat{A}_i = R_i - \bar{R}. \quad (27)$$

Under the same tied-verifier setting, this reduces to

$$\hat{A}_i = -w(L_i - \bar{L}), \quad (28)$$

which preserves the intended relative magnitude of the efficiency reward controlled by w .

Mean-centering also remains a valid variance-reduction baseline for policy-gradient optimization. Let

$$S_i = \nabla_\phi \log \pi_\phi(\tau_i) \quad (29)$$

denote the score function of trajectory τ_i , and suppose that a group contains G independently sampled trajectories. Then

$$\mathbb{E}[(R_i - \bar{R})S_i] = \left(1 - \frac{1}{G}\right) \nabla_{\phi} J(\phi). \quad (30)$$

Thus, using the finite-group sample mean introduces only a positive multiplicative factor $1 - \frac{1}{G}$ and does not change the expected gradient ascent direction. In contrast, dividing by the group standard deviation adaptively rescales different groups and can disproportionately amplify gradients from low-variance groups. We therefore use the unnormalized group-relative advantage to preserve the intended balance between correctness and sampling efficiency.