HiBlock: Learning Hierarchical Block-Size Policy for Cross-Domain Generalisation in Diffusion Large Language Models

Anonymous authors
Paper under double-blind review

Abstract

Recent advances in diffusion large language models (dLLMs) enable efficient block-based generation in single-domain settings, where blocks have fixed sizes and tokens within each block are denoised jointly. However, directly extending fixed-size block decoding to multi-domain scenarios is challenging because different domains naturally favour different block size strategies and no single size performs best across all domains. Recent adaptive block-size methods attempt to address this by adding heuristic rules that extend block boundaries to certain predefined tokens or newline characters. However, these methods rely on external rules and prompt-specific cues, thereby not enabling dLLMs to explicitly learn adaptive block-size selection ability for cross-domain generalisation. To address this, we propose **HiBlock**, a **Hi**erarchical **Block**-size policy learning framework via Group Relative Policy Optimization (GRPO). HiBlock comprises a high-level policy that selects a block-size action and a low-level policy that generates the response under the selected block size, transforming block size from a predefined hyperparameter into a learnable action. During post-training, HiBlock explores diverse block-size actions across GRPO rollouts, while during inference, the learnt policy selects a block size for each input task before response generation. Experiments on 13 benchmarks show its effectiveness in dLLM cross-domain generalisation. Our code is available at anonymous.4open.science/r/HiBlock-2026.

1 Introduction

Recent advances in diffusion large language models (dLLMs) have enabled efficient and effective block-based reasoning generation through block-wise semi-autoregressive decoding (Nie et al., 2025; Ye et al., 2025; Arriola et al., 2025; Bie et al., 2025; 2026; Zhu et al., 2026; Wu et al., 2026b;a). In this parallel decoding paradigm, blocks are decoded sequentially whilst tokens within each block can be denoised in parallel. Most current dLLMs use a predefined fixed block size throughout training and inference for all domain tasks. Through empirical observation in Figure 1 on LLaDA-8B-Instruct, we reveal that no single block size achieves the best reasoning performance across all domains. This is because tasks from different domains exhibit different reasoning structures and naturally favour distinct block size strategies. Smaller blocks condition more frequently on completed tokens for fine-grained reasoning, whereas larger blocks jointly denoise more tokens and require fewer blocks for faster decoding. Therefore, one predefined block size cannot satisfy the different requirements of all domains.

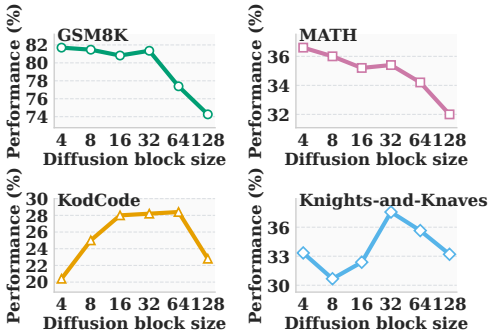


Figure 1: **Motivation of HiBlock.** The performance of dLLMs varies substantially across block sizes, and no single block size achieves the best performance across all domains.

Most existing dLLM methods adopt a single fixed block size throughout post-training and inference, preventing the dLLM block-based parallel decoding granularity from adapting to dif-

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

ferent prompts or domains (Zhao et al., 2025; Tang et al., 2026a; Rojas et al., 2026; Zhong et al., 2026; Wang et al., 2026a; Ou et al., 2026; Wang et al., 2026c; Ni et al., 2026). Recent studies introduce dynamic blocks to improve decoding flexibility within a single domain. However, their block-size decisions largely depend on externally specified tokens, such as adding extra block-ending tokens (Jiang et al., 2026a), or hand-crafted cues and predefined rules, such as locating the next newline character to cut the block (Lu et al., 2026). As in Table 1, these methods generally adapt block boundaries within a response via predefined rules or controlled tokens. Yet they typically do not enable dLLMs to jointly learn the block-size policy and the generation policy.

Table 1: **Comparison of existing block size methods. Dyn.** denotes dynamic block sizes, **Learn.** denotes learnable sample-level block-size policies, **Ext.** denotes reliance on extra rules or tokens, **CD gen.** denotes support for cross-domain generalisation. "–" denotes not applicable.

Category	Dyn.	Learn.	Ext.	CD gen.
Fixed-block RL methods	×	×	–	×
Dynamic block methods	✓	×	✓	×
HiBlock	✓	✓	×	✓

In light of the above discussion, we propose **HiBlock**, a **Hi**erarchical **Block**-Size Policy Learning framework that explicitly learns block-size decisions via a Group Relative Policy Optimisation (GRPO) style objective. HiBlock formulates block-based decoding as a hierarchical policy implemented by the same dLLM. The high-level block-size policy selects a block-size action according to the input, while the low-level block-based decoding policy generates the response under the selected structure. During post-training, HiBlock explores block-size actions within the GRPO rollouts of each prompt and applies the group-relative advantage to both the selected action and its response, jointly optimising block-size selection and response generation. Through multi-domain post-training, HiBlock learns the rich task-dependent block-size preference information on diverse domain tasks. During inference, the learnt high-level policy selects a block-size action for each input task, after which the low-level policy generates the response under the corresponding block size. Extensive experiments across 6 dLLM backbones and 13 benchmarks demonstrate its improved cross-domain generalisation. Our main contributions are:

- Our empirical results reveal a **cross-domain block-size conflict** in dLLM multi-domain RL post-training, where different domains favour different block-size decoding strategies in dLLMs.
- We propose **HiBlock**, a novel hierarchical block-size policy learning framework for dLLMs that reformulates block size from a predefined hyperparameter into a learnable policy action.
- HiBlock can be readily integrated into and improve existing GRPO-based methods for dLLMs.
- Experiments across 6 dLLMs and 13 benchmarks show improved cross-domain generalisation.

2 Preliminaries

We first introduce the notation required to formalise **adaptive block-size decoding** in multi-domain dLLM post-training. Let $\{\mathcal{D}_d\}_{d=1}^{D}$ denote a collection of domain distributions. A prompt $x \sim \mathcal{D}_d$ belongs to domain d, and $\mathbf{o} = (o^1, \ldots, o^L)$ denotes its response body of length L. The raw block-size candidate set is denoted by $\mathcal{C}_0 = \{c_1, \ldots, c_S\}$, where S is the number of candidate block sizes.

2.1 Diffusion Large Language Models

Diffusion training. Let θ denote the trainable parameters of the dLLM and let p_θ denote its denoising distribution. Let $\mathcal{D}_{\text{text}}$ denote the clean-text distribution and let $\mathbf{x}_0 = (x_0^1, \ldots, x_0^L) \sim \mathcal{D}_{\text{text}}$ denote a clean sequence of length L. At diffusion time $t \in [0, 1]$, let $q_t(\cdot \mid \mathbf{x}_0)$ denote the forward corruption distribution and let $\mathbf{x}_t \sim q_t(\cdot \mid \mathbf{x}_0)$ denote the corrupted sequence. Define the masked-position set as $\mathcal{M}_t = \{i : x_t^i = \texttt{mask}\}$. The denoising objective is defined as:

$$\mathcal{L}_{\text{diff}}(\theta) = -\mathbb{E}_{\mathbf{x}_0, t, \mathbf{x}_t}\left[\frac{1}{t}\sum\nolimits_{i \in \mathcal{M}_t} \log p_\theta(x_0^i \mid \mathbf{x}_t)\right]. \tag{1}$$

Here, $t \sim \mathcal{U}[0, 1]$, and the factor $1/t$ balances reconstruction losses across diffusion times.

Block-based decoding in dLLMs. Block-based dLLMs generate a response through a sequence of fixed-size decoding blocks. The blocks are completed sequentially, while the masked positions within each active block are denoised jointly as:

Definition 1: Fixed Block-Size Decoding

Given a response of length L and a block size c satisfying $c \mid L$, block-based generation partitions the response into $K_c = L/c$ blocks:

$$\mathbf{b}_k^{(c)} = \{(k-1)c+1, \ldots, kc\}, k = 1, \ldots, K_c. \tag{2}$$

The blocks are completed in increasing order of k, while all masked positions within the active block $\mathbf{b}_k^{(c)}$ are denoised jointly. Completing the response therefore requires K_c sequential blocks, with each block containing exactly c token positions.

Intuitively, increasing c reduces the number of blocks but weakens token-level conditioning within each block. Decreasing c provides finer-grained conditioning but requires more sequential transitions. Block-size selection can therefore affect both decoding efficiency and response quality.

2.2 Reinforcement Learning for dLLMs

Existing reinforcement learning (RL) for dLLM methods (Zhao et al., 2025; Tang et al., 2026a; Rojas et al., 2026; Zhong et al., 2026; Wang et al., 2026a; Ou et al., 2026; Wang et al., 2026c; Ni et al., 2026) typically optimise dLLM response generation under a single predefined block size, leaving block-size selection outside the learnable action space. Let $\pi_\theta^{\text{resp}}(\mathbf{o} \mid x, c)$ denote the block-based decoding policy parameterised by θ under the predefined block size c. Its induced rollout distribution is $P_\theta^{(c)}(\mathbf{o} \mid x) = \pi_\theta^{\text{resp}}(\mathbf{o} \mid x, c)$. Let G denote the number of rollouts generated for each prompt, and let θ_{old} denote the frozen policy parameters used for rollout generation. For each prompt x, a response group $\mathbf{o}_{1:G} = (\mathbf{o}_1, \ldots, \mathbf{o}_G)$ is sampled independently from the old policy:

$$\mathbf{o}_g \sim P_{\theta_{\text{old}}}^{(c)}(\cdot \mid x), \qquad g = 1, \ldots, G. \tag{3}$$

GRPO optimises the sampled response tokens according to their group-relative advantages. Let o_g^i denote the i-th token of response $\mathbf{o}_g$, and let $\widetilde{\mathbf{o}}_{g,t}$ denote its corrupted diffusion state at diffusion time step t under the block partition induced by c. The token-level policy ratio is:

$$\rho_{g,i}(\theta) = \frac{p_\theta(o_g^i \mid x, \widetilde{\mathbf{o}}_{g,t})}{p_{\theta_{\text{old}}}(o_g^i \mid x, \widetilde{\mathbf{o}}_{g,t})}. \tag{4}$$

The corresponding clipped surrogate is $\mathcal{S}_\epsilon(\rho, A) = \min\{\rho A, \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A\}$, where A denotes an advantage value and ϵ is the clipping range. Let $\widehat{A}_g$ denote the group-relative advantage of rollout g, and let $\mathbb{E}_c[\cdot]$ denote the expectation over prompts, response groups, and the diffusion corruption states used to construct $\widetilde{\mathbf{o}}_{g,t}$ under block size c. The GRPO objective under a fixed block-size constraint is:

$$\mathcal{J}_{\text{GRPO}}(\theta; c) = \mathbb{E}_c\left[\frac{1}{GL}\sum_{g=1}^{G}\sum_{i=1}^{L}\mathcal{S}_\epsilon\left(\rho_{g,i}(\theta), \widehat{A}_g\right) - \beta D_{\text{KL}}\right]. \tag{5}$$

Here, D_{KL} denotes the KL divergence between the current and reference policies, and $\beta \geq 0$ controls its contribution. The dependence on c indicates that the objective optimises response generation under a predetermined block-based decoding structure rather than learning the block-size decision:

Definition 2: Block-Based Decoding Policy under a Predefined Block Size

Given a prompt x and a predefined block size c, existing dLLM RL methods learn only the block-based decoding policy under a single fixed c:

$$\mathbf{o} \sim \pi_\theta^{\text{resp}}(\cdot \mid x, c), \qquad c \notin \mathcal{A}_\pi. \tag{6}$$

Here, $\mathcal{A}_\pi$ denotes the learnable action space. Since the block size c lies outside this space, the policy neither learns to select it nor flexibly adapts it across different inputs.

A predefined block size affects the rollout distribution $P_\theta^{(c)}(\mathbf{o} \mid x)$ and consequently influences the trajectory rewards and group-relative advantages during RL post-training. However, these learning signals optimise only response generation under a single fixed block size. Existing dLLM RL methods therefore learn **how to generate under a predefined block size strategy**, rather than **which block size should be selected for different tasks**.

2.3 CROSS-DOMAIN BLOCK-SIZE CONFLICT

In multi-domain RL, tasks from different domains may favour different block sizes. For domain d, we define its preferred block-size set as:

$$\mathcal{C}_d^\star = \arg\max_{c\in\mathcal{C}_0} \mathbb{E}_{\substack{x\sim\mathcal{D}_d,\\ \mathbf{o}\sim P_{\theta_{\text{old}}}^{(c)}(\cdot|x)}} \left[R_{\text{task}}^{(d)}(x,\mathbf{o}) \right]. \tag{7}$$

where $R_{\text{task}}^{(d)}(x,\mathbf{o})$ denotes the domain-specific task reward. Thus, $\mathcal{C}_d^\star$ contains the block sizes that maximise the expected task performance in domain d. We then formulate our problem definition as:

Problem: Cross-Domain Block-Size Conflict

The block-size conflict occurs when domains do not share a common preferred block size:

$$\bigcap_{d=1}^{D} \mathcal{C}_d^\star = \varnothing. \tag{8}$$

Therefore, no single block size can maximise the expected task performance across all domains.

To address this conflict, HiBlock introduces a **hierarchical block-size policy** comprising a high-level **block-size policy** that selects a block-size action for the input task and a low-level **block-based decoding policy** that generates the response.

3 HIBLOCK: LEARNING BLOCK-SIZE FOR CROSS-DOMAIN ADAPTATION

HiBlock aims to enhance dLLMs' cross-domain generalisation by learning to select different block sizes from multiple source domains with diverse reasoning structures. As illustrated in Figure 2, HiBlock formulates block-based decoding as a **hierarchical block-size policy** comprising a high-level **block-size policy** and a low-level **block-based response-generation policy**. The high-level policy selects a block-size action according to the input task, while the low-level policy generates the response under the selected block size. The detailed algorithm is provided in Algorithm 1.

3.1 MULTI-DOMAIN DLLM POST-TRAINING SETUP

To achieve multi-domain post-training, HiBlock first balances the training data across multiple source domains and formulates block-size selection and block-based response generation as two dependent policy levels for the same dLLMs.

Balanced multi-domain data pre-processing. For multi-domain post-training, HiBlock first samples multi-domain data in a balanced way to prevent large domain from dominating the learning process. Let N_d denote the number of training examples in domain d, and let $N_{\min} = \min_{1\le d\le D} N_d$. For each domain, we randomly sample $N_{\min}$ examples to obtain $\widetilde{\mathcal{D}}_d = \text{Sample}(\mathcal{D}_d, N_{\min})$. The sampled subsets are then combined and shuffled as:

$$\mathcal{D}_{\text{bal}} = \text{Shuffle}\left(\bigcup_{d=1}^{D} \widetilde{\mathcal{D}}_d \right). \tag{9}$$

This balanced training set ensures each domain contributes to multi-domain post-training in a balanced way for learning the block-size policy.

3.2 HIERARCHICAL BLOCK-SIZE POLICY FOR DLLMS.

HiBlock formulates the hierarchical block-size policy into two sequential modules for multi-domain post-training: high-level block-size action selection followed by low-level block-based response generation. Both modules are implemented by the same dLLM rather than by separate networks.

(i) High level: Block-size action selection. For an input prompt $x \sim \mathcal{D}_{\text{bal}}$, the high-level block-size policy selects a block-size action according to the input task. We retain only size in $\mathcal{C}_0$ that are positive and evenly divide the generation length L, i.e., $L \bmod c = 0$, yielding $\mathcal{C} = \{c \in \mathcal{C}_0 : c >$

$0,\ c \mid L\}$. Here, c denotes the numerical block size used for decoding. To represent this decision in the dLLM token space, each $c \in \mathcal{C}$ is encoded as a unique token sequence $\mathbf{z}_c = \mathrm{Enc}(c)$, where Enc denotes the tokeniser. These sequences form the block-size action set $\mathcal{Z}_\mathcal{C} = \{\mathbf{z}_c : c \in \mathcal{C}\}$. Thus, $\mathbf{z}_c$ serves as the learnable block-size action, while its corresponding value c determines the block size.

(ii) Low level: Block-based response generation. After the high-level policy selects a block size action $\mathbf{z} \in \mathcal{Z}_\mathcal{C}$, the low-level block-based response generation policy generates the response $\mathbf{o}$ conditioned on the input prompt and the selected action. Since each action $\mathbf{z}_c$ uniquely represents a numerical block size c, the selected high-level block size determines the block decoding structure used for low-level response generation.

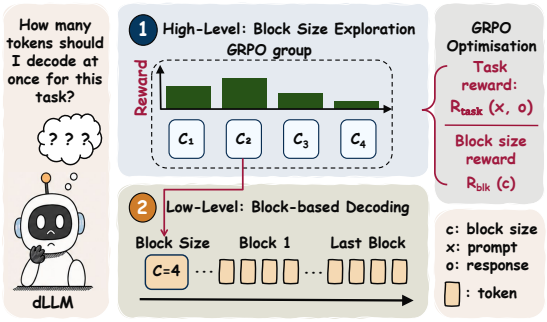


Figure 2: **Illustration of HiBlock.** Most existing dLLM methods do not explicitly learn input-conditioned block-size selection. HiBlock introduces a hierarchical policy that first selects a block-size action and then generates the response under the selected block size, with both policy levels jointly optimised using a GRPO-style objective.

Together, the hierarchical policy over the block-size action and response is defined as:

$$\pi_\theta(\mathbf{z}, \mathbf{o} \mid x) = \pi_\theta^{\mathrm{blk}}(\mathbf{z} \mid x)\pi_\theta^{\mathrm{resp}}(\mathbf{o} \mid x, \mathbf{z}), \tag{10}$$

where $\pi_\theta^{\mathrm{blk}}$ denotes the high-level block-size policy and $\pi_\theta^{\mathrm{resp}}$ denotes the low-level block-based response-generation policy. The high-level policy selects the block-size action that determines the decoding structure, while the low-level policy generates the response under that structure. Therefore, block-size selection is integrated into the reasoning trajectory within the same rollout, allowing each block-size action to be evaluated based on its resulting response generation.

Balanced block-size exploration. During multi-domain post-training, HiBlock explores different block sizes for each input task $x \sim \mathcal{D}_{\mathrm{bal}}$. Direct sampling from an untrained block-size policy may cover only a few candidates, limiting comparisons among different block sizes. HiBlock therefore balances exploration across G rollouts. For each rollout, a block-size action is randomly selected without replacement from the candidate set, ensuring that every candidate is evaluated once within each exploration cycle. The candidate set is reset after all candidates have been covered. For rollout g, let c_g denote the selected numerical block size and $\mathbf{z}_g$ the corresponding block-size action. The block-size and response trajectory is defined as:

$$\mathbf{y}_g = \mathbf{z}_g \,\|\, \mathbf{o}_g, \quad \text{with } \mathbf{o}_g \sim \pi_{\theta_{\mathrm{old}}}^{\mathrm{resp}}(\cdot \mid x, \mathbf{z}_g), \tag{11}$$

where $\|$ is concatenation and $\mathbf{o}_g$ is the block-based response generated under $\mathbf{z}_g$. These trajectories are then evaluated and optimised using a GRPO-style objective. During training, candidate block-size actions are assigned by balanced exploration rather than simply sampled from the high-level policy, while during inference, the learnt high-level policy selects the block-size action for decoding.

3.3 Hierarchical Policy Optimisation

HiBlock optimises the two-level hierarchical policy with a GRPO-style clipped objective. Specifically, a task reward and a block-size reward first evaluate each block-size and response trajectory. A group-relative advantage is then derived from these balanced trajectories and applied to both the high-level block-size policy and the low-level block-based response-generation policy.

Dual-reward trajectory evaluation. HiBlock evaluates each response trajectory via a task reward and a block-size reward. The task reward $R_{\mathrm{task}}^{(d)}(x, \mathbf{o}_g)$ measures the quality of response $\mathbf{o}_g$ for input task $x \in \widetilde{\mathcal{D}}_d$ from domain d. The block-size reward introduces an efficiency preference for larger blocks and is defined as $R_{\mathrm{blk}}(c_g) = c_g / c_{\max}$, where $c_{\max} = \max_{c \in \mathcal{C}} c$. This prevents the block-size policy from consistently selecting small blocks, which would increase the number of sequential blocks and reduce decoding efficiency. The complete reward for rollout g is:

$$R_{d,g} = R_{\mathrm{task}}^{(d)}(x, \mathbf{o}_g) + \alpha R_{\mathrm{blk}}(c_g), \tag{12}$$

where $\alpha \geq 0$ controls the contribution of the block size reward. When candidate responses receive the same task reward, the block size reward acts as an efficiency tie breaker and favours larger blocks for decoding efficiency. For the G trajectories, let $\overline{R}_{d,x}$ and $s_{d,x}$ denote the mean and standard deviation of their rewards. The group-relative advantage is formulated as:

$$\widehat{A}_{d,g} = \frac{R_{d,g} - \overline{R}_{d,x}}{s_{d,x} + \delta}, \tag{13}$$

where $\overline{R}_{d,x} = \frac{1}{G}\sum_{g=1}^{G} R_{d,g}$ and $\delta > 0$ ensures numerical stability. $\widehat{A}_{d,g}$ measures the relative performance of the complete block-size and response trajectory within the rollout group. Intuitively, a positive advantage indicates that the block size strategy achieves a higher reward than the group average, suggesting that this block size is more suitable for the input task than others in the group.

Joint hierarchical policy optimisation. HiBlock jointly optimises both policy levels by applying $\widehat{A}_{d,g}$ to all positions in the block-size and response trajectory $\mathbf{y}_g$. Let $\mathcal{I}_g^{\text{blk}}$ and $\mathcal{I}_g^{\text{resp}}$ denote the block-size action and response positions, with the complete set of token positions defined as their union $\mathcal{I}_g = \mathcal{I}_g^{\text{blk}} \cup \mathcal{I}_g^{\text{resp}}$. For each $i \in \mathcal{I}_g$, $\rho_{g,i}(\theta)$ denotes the likelihood ratio between the current and old policies at the corresponding trajectory position. For block-size positions, since the actions are assigned by balanced exploration rather than sampled from the old high-level policy, we adopt this ratio as a proximal clipping term. For response positions, it is computed conditioned on the selected block action $\mathbf{z}_g$. The GRPO-style clipped surrogate $\mathcal{S}_\epsilon$ is then applied using $\widehat{A}_{d,g}$. Let $\mathbb{E}_d[\cdot]$ denote the expectation over input tasks from $\widetilde{\mathcal{D}}_d$, their rollout groups, and the diffusion corruption states used for likelihood estimation. The multi-domain hierarchical policy objective is defined as:

$$\mathcal{J}_{\text{HiBlock}}(\theta) = \frac{1}{D}\sum_{d=1}^{D} \mathbb{E}_d\left[\frac{1}{N_x}\sum_{g=1}^{G}\sum_{i\in\mathcal{I}_g} \mathcal{S}_\epsilon\Big(\rho_{g,i}(\theta), \widehat{A}_{d,g}\Big) - \beta D_{\text{KL}}\right], \tag{14}$$

where $N_x = \sum_{g=1}^{G} |\mathcal{I}_g|$ and $\beta \geq 0$ controls reference-policy KL regularisation, which can be removed by setting $\beta = 0$ following previous work (Tang et al., 2026a). Since $\mathcal{I}_g$ includes positions from both policy levels, the group-relative advantage jointly optimises the selected block-size action and its generated response. A positive advantage encourages higher likelihoods for the block-size action and its response, while a negative advantage discourages them. Accordingly, we refer to this formulation as a GRPO-style objective rather than standard on-policy GRPO. HiBlock therefore jointly learns block-size selection and block-based response generation within one dLLM.

3.4 Cross-Domain Generalisation through Adaptive Block-Size Selection

Learning from tasks with diverse reasoning structures and block-size preferences enables the high-level policy to associate input tasks with suitable block sizes. During inference on a held-out target domain, the learnt high-level policy selects block size with the highest normalised likelihood as:

$$\widehat{\mathbf{z}} = \arg\max_{\mathbf{z}\in\mathcal{Z}_\mathcal{C}} \frac{1}{|\mathbf{z}|}\log \pi_\theta^{\text{blk}}(\mathbf{z} \mid x), \tag{15}$$

where $|\mathbf{z}|$ denotes the number of tokens in the block-size action. The low-level policy then generates the response under the corresponding block size:

$$\widehat{\mathbf{o}} \sim \pi_\theta^{\text{resp}}(\cdot \mid x, \widehat{\mathbf{z}}), \tag{16}$$

where $\widehat{\mathbf{z}} \in \mathcal{Z}_\mathcal{C}$ determines the block size used to generate $\widehat{\mathbf{o}}$. By conditioning block-size selection on each input task, the model can adapt its decoding structure to different reasoning tasks across domains. Intuitively, HiBlock **transfers the block-size selection ability learnt across diverse source domains to different target domains.** This enables adaptive block-size decoding without predefined block-size labels, separately trained predictors, block-control tokens, or hand-crafted rules, distinguishing HiBlock from existing block-based methods (Jiang et al., 2026b;a; Lu et al., 2026).

3.5 Complexity Analysis

Let S denote the number of block sizes and G the number of rollouts per input. HiBlock adds only the evaluation of S candidate block size, while response generation remains the dominant training cost. During inference, these candidates are scored once before generation. Thus, HiBlock introduces only a small scoring cost without changing the asymptotic complexity of dLLM decoding.

Table 2: **Cross-domain generalisation of RL-post-trained dLLMs.** LLaDA-8B-Instruct (Nie et al., 2025) is post-trained on different source domains in Table 6, and evaluated on 13 target-domain benchmarks. StableDRL (Zhong et al., 2026) is the default RL algorithm for all experiments. For each target benchmark, **Vanilla** uses the same source domain training set but uses the fixed block size 32 without learnt block size selection. **C.** denotes Code Generation, and **Multi** denotes multi-domain RL. Shaded diagonal cells indicate in-domain RL, while shaded rows indicate multi-domain RL. Following previous dLLM work, the best checkpoint accuracy is reported under a consistent zero-shot setting and is highlighted by **best** and runner-up. Experiments with other dLLM backbones and different RL algorithms are provided in Sections 4.4 and 4.5.

Group	Source / Target	Mathematical Reasoning			Code Generation			Puzzle Solving		General Capabilities			Advanced Reasoning	
		Countdown	GSM8K	MATH500	HumanEval	MBPP	KodCode	Sudoku	KK	HellaSwag	MMLU	ARC-E	MMLU-Pro	ARC-C
Base	**LLaDA**	16.80	76.19	32.00	28.66	33.60	22.00	7.81	30.00	56.21	53.46	76.98	33.79	73.21
	+ SFT	15.23	75.59	32.20	28.05	30.40	19.40	8.20	32.00	55.65	55.58	77.65	34.03	69.80
Math	**Countdown**	**58.98**	70.74	28.40	26.83	31.00	22.60	7.81	24.57	53.38	54.24	78.16	30.91	73.55
	GSM8K	16.02	**81.35**	32.60	27.44	32.20	22.60	10.55	30.57	58.07	56.22	80.51	33.96	75.51
	MATH500	14.06	77.33	**35.40**	27.44	33.00	23.60	10.16	32.14	57.73	54.01	79.12	35.21	77.47
C.	**KodCode**	14.45	74.15	32.00	**31.10**	**34.80**	**28.20**	12.11	27.43	54.79	53.28	82.07	33.84	75.94
Puzzle	**Sudoku**	6.64	68.23	30.40	26.22	32.40	23.40	**25.39**	33.29	57.24	56.77	80.09	36.50	73.46
	KK	8.98	70.28	29.60	24.39	29.80	20.80	15.63	**37.57**	58.25	55.62	81.10	34.89	74.32
Multi	**Vanilla**	30.08	57.24	28.20	24.39	24.40	22.60	9.77	30.14	52.10	52.06	73.95	29.79	65.87
	AdaBlock-dLLM	32.81	66.49	30.40	23.78	23.80	23.40	11.33	31.43	51.70	53.12	75.08	30.24	67.15
	b1	35.55	72.48	33.00	26.83	28.80	25.20	12.25	33.86	55.36	57.34	79.25	30.08	68.86
	HiBlock	**59.76**	**81.58**	**34.40**	**32.32**	**33.60**	**29.20**	**25.80**	**42.29**	**60.63**	**58.89**	**85.52**	**37.20**	**80.80**

4 EXPERIMENTS

Setup. We follow most dLLM methods to evaluate HiBlock on 13 benchmarks covering code generation, mathematical reasoning, logical puzzles and general capabilities as shown in Table 6. Following existing work (Jiang et al., 2026b), we treat each benchmark as an individual domain for multi-domain post-training and cross-domain evaluation. The evaluation covers 6 popular dLLM backbones: LLaDA-8B-Instruct, LLaDA-1.5, Dream-Instruct-7B, SDAR-8B-Chat-b32, TraDo-8B-Instruct, and LLaDA2.0-16B. Unless stated otherwise, we adopt LLaDA-8B-Instruct (Nie et al., 2025) as the default backbone. Our multi-domain post-training and cross-domain evaluation setting follows existing work (Jiang et al., 2026b). We report zero-shot task performance with a default generation length of 256 and a block size of 32 for fixed-size baselines. Following previous methods (Zhao et al., 2025; Wang et al., 2026c; Tang et al., 2026a; Rojas et al., 2026; He et al., 2025; Zhong et al., 2026; Ni et al., 2026; Jiang et al., 2026b), we use pass@1 accuracy for benchmarks evaluated at the response level. Knights-and-Knaves uses character role assignment accuracy, while Sudoku instead uses cell-level accuracy over all initially empty cells following Zhao et al. (2025).

Baselines. We first compare HiBlock with existing dynamic block-size methods, including AdaBlock-dLLM (Lu et al., 2026) and *b1* (Jiang et al., 2026a), under a consistent multi-domain post-training and cross-domain evaluation protocol. As a plug-and-play framework, HiBlock can be integrated with existing GRPO-based RL methods for dLLMs to improve their performance. StableDRL (Zhong et al., 2026) is adopted as the default fixed-size RL post-training algorithm due to its strong empirical performance. We further integrate HiBlock with *d1* (Zhao et al., 2025), *wd1* (Tang et al., 2026a) and GDPO (Rojas et al., 2026) to evaluate its effectiveness with different RL algorithms. SFT follows *d1* by the denoising cross-entropy objective.

Implementation. Unless otherwise specified, all experiments are conducted on four AMD MI300X GPUs, each with 192GB of memory. StableDRL (Zhong et al., 2026) is used as the default RL algorithm because of its robust empirical performance. All methods follow the same training and evaluation settings unless their original designs require additional supervision or data construction. We report the zero-shot accuracy of the best checkpoint following a consistent evaluation protocol in most existing dLLM methods (Zhao et al., 2025; Tang et al., 2026a; Rojas et al., 2026). The detailed rewards and hyperparameters are provided in additional implementation details in Appendix C.

4.1 OVERALL PERFORMANCE

In Table 2, we present the main cross-domain generalisation results of HiBlock. For the multi-domain setting, we reproduce *b1* and AdaBlock-dLLM under a consistent cross-domain zero-shot evaluation protocol. Our main findings are: **(i) HiBlock achieves the strongest overall perfor-**

mance across the evaluation benchmarks, showing that jointly learning block-size selection and response generation enables more consistent generalisation than using a single predefined block size. **(ii) Direct multi-domain RL with a fixed block size does not reliably improve performance and can even cause negative transfer**, confirming that simply combining data from different domains cannot resolve their conflicting block-size preferences. In contrast, HiBlock adapts the decoding structure to each input and therefore accommodates diverse reasoning generation structures for different domain tasks. **(iii) HiBlock generalises more effectively than both single-domain RL and heuristic dynamic block-size methods**. Single-domain RL tends to specialise in its source domain and transfers inconsistently, while heuristic dynamic block-size methods adjust block boundaries using predefined rules without learning task-dependent block-size decisions. Overall, these results demonstrate that the block-size selection ability learnt from multiple source domains can effectively transfer to different target-domain tasks, leading to stronger cross-domain generalisation.

Further detailed analyses of ablations, effectiveness of block size policy, additional dLLM backbones and RL algorithms are provided in Sections 4.2, 4.3, 4.4 and 4.5, respectively. Detailed hyperparameter sensitivity, block-size conflict visualisation by rewards, longer generation length, and case study are provided in Appendix D, E, F, and H, respectively.

4.2 Ablation Study

Table 3 examines the contribution of each HiBlock component. Vanilla Block Size Prompt uses the same candidate set without a learnt selector, and its resulting selections are nearly uniform across domains, indicating that prompting alone provides little task-dependent block-size preference, resulting in low throughput and inferior accuracy. Removing the high-level policy achieves higher throughput with a larger fixed average block size, but substantially reduces accuracy as the same block size is constrained across all domains. Notably, Vanilla Block Size Prompt and HiBlock share the same candidate block-size set, while only HiBlock learns block-size selection from trajectory rewards. Removing the block-size reward retains comparable accuracy but decreases the average block size and throughput. These results show that our learnt high-level block size policy enables effective task-dependent block-size selection, while the block-size reward improves the accuracy and throughput trade-off by encouraging larger selected block sizes and preserving dLLM block decoding efficiency.

Table 3: **Ablation study.** Accuracy (%), end-to-end inference throughput (tokens/s), and selected block size are averaged over Countdown, GSM8K, MATH500, and Sudoku. Throughput includes block size selection for HiBlock. **Vanilla Block Size Prompt** presents the same candidate block size set as HiBlock and prompts the dLLM to select one without learning block size policy. **w/o High-Level Policy** refers to the vanilla fixed-size baseline.

Method	Avg. Acc.	Throughput	Avg. Block Size
Vanilla Block Size Prompt	26.50	16.94	16.85
w/o High-Level Policy ($c = 32$)	31.32	**26.51**	32.00
w/o Block-Size Reward ($\alpha = 0$)	49.25	20.75	18.70
HiBlock	**50.39**	24.78	23.66

4.3 Effectiveness of the Block-Size Policy

Figure 3 compares the block-size distributions selected by Oracle, Vanilla, and HiBlock. We evaluate $\{4, 8, 16, 32, 64, 128\}$ because these block sizes are commonly adopted in existing dLLMs and can evenly divide the default generation length. For Oracle, we evaluate each test example using all candidates and record the proportion for which each size performs best, selecting the largest in the case of a tie to remain consistent with the block size efficiency preference in Eq. 12. Vanilla directly prompts LLaDA-8B-Instruct to select one candidate before response generation. Its selections are nearly uniform across domains, indicating that prompt-only selection provides little task-dependent block-size preference. In contrast, **HiBlock's block size selection more closely follows the oracle block size distribution**, showcasing that its learnt policy effectively captures task-dependent block-size preferences for dLLMs.

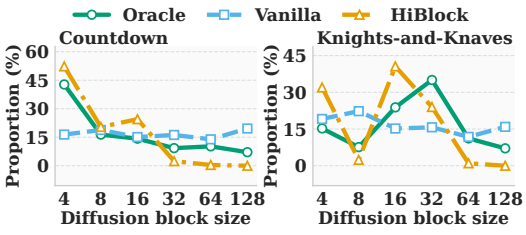


Figure 3: **Effectiveness of block-size selection in HiBlock.** Oracle denotes the proportion of test examples for which each candidate size achieves the best performance. Vanilla uses prompt-based selection from the same block size set, while HiBlock uses the learnt block size policy.

4.4 On Generalisation to More dLLMs

In this section, we evaluate whether HiBlock generalises to different dLLM backbones under the same multi-domain post-training and cross-domain evaluation settings. As in Table 4, integrating HiBlock consistently outperforms the vanilla multi-domain RL performance across model families and scales, indicating that the learnt block-size selection ability is not tied to a specific backbone. This showcases that **HiBlock can generalise across different dLLMs.**

Table 4: **On generalisation to different dLLMs.** Each dLLM is post-trained via default StableDRL and evaluated under the same zero-shot setting. The unshaded rows denote vanilla fixed-size baselines, while the shaded rows denote integration with HiBlock. Test accuracy is highlighted by **best** and runner-up.

Model	Countdown	GSM8K	MATH500	Sudoku
LLaDA-8B-Instruct	30.08	57.24	28.20	9.77
+ **HiBlock**	**59.76**	81.58	34.40	25.80
LLaDA-1.5	25.39	66.41	24.40	12.27
+ **HiBlock**	57.42	80.29	35.20	24.86
Dream-Instruct-7B	19.92	62.09	30.20	10.04
+ **HiBlock**	32.81	81.12	34.80	**28.90**
SDAR-8B-Chat-b32	24.22	70.36	25.40	10.61
+ **HiBlock**	46.09	83.02	36.80	24.37
TraDo-8B-Instruct	26.95	58.83	23.60	14.33
+ **HiBlock**	30.47	65.73	34.60	21.94
LLaDA2.0-16B	39.06	70.81	31.40	19.60
+ **HiBlock**	59.38	**84.91**	**40.60**	28.73

4.5 On Generalisation to More RL

We further examine whether HiBlock can be integrated with different dLLM RL algorithms under their default single-domain post-training and evaluation settings in contrast to the multi-domain training setting in Table 2. As shown in Table 5, plugging HiBlock into *d1*, *wd1*, GDPO and StableDRL consistently improves their performance across all four representative benchmarks, which are the most commonly evaluated datasets in previous methods (Zhao et al., 2025; Tang et al., 2026a). This demonstrate that **HiBlock is not tied to a specific RL algorithm** and can consistently enhance different dLLM RL methods by adaptive block-size decoding.

Table 5: **On generalisation to different RL algorithms.** Each model is post-trained and evaluated on the same domain. HiBlock can be readily plugged into existing RL algorithms as it is orthogonal to those RL algorithms. Test accuracy is highlighted by **best** and runner-up.

Method	Countdown	GSM8K	MATH500	Sudoku
LLaDA	16.80	76.19	32.00	7.81
+ *d1*	25.39	77.03	33.40	15.06
+ *d1* + **HiBlock**	30.47	79.08	34.80	20.31
+ GDPO	17.97	76.57	32.20	9.94
+ GDPO + **HiBlock**	24.22	78.24	34.00	16.82
+ *wd1*	39.45	78.85	34.20	23.14
+ *wd1* + **HiBlock**	44.53	81.12	35.60	25.73
+ StableDRL	58.98	81.35	35.40	25.39
+ StableDRL + **HiBlock**	**59.38**	**82.11**	**35.80**	**26.09**

5 Related Work

This work is related to three main research fields. **(1) Diffusion Large Language Models.** dLLMs generate text by iteratively denoising masked or corrupted tokens (Sohl-Dickstein et al., 2015; Campbell et al., 2022; Meng et al., 2022; Austin et al., 2021; Lou et al., 2024; Shi et al., 2024; Sahoo et al., 2024; Nie et al., 2024; Zhu et al., 2026; Ye et al., 2025; Nie et al., 2025; Bie et al., 2025; 2026). Many dLLMs further adopt block-based semi-autoregressive decoding (Arriola et al., 2025). **(2) Reinforcement Learning for dLLM Reasoning.** Recent dLLM RL studies improve reasoning through token-level or sequence-level policy optimisation (Zhao et al., 2025; Wang et al., 2026c; Tang et al., 2026a; Ni et al., 2026; Rojas et al., 2026; He et al., 2025; Ou et al., 2026; Wang et al., 2026a; Zhong et al., 2026; Zhu et al., 2025; Yang et al., 2026; Jiang et al., 2026b). Most optimise response generation under a predefined block size and focus on single-domain post-training. Block-R1 is a dataset method that requires additional sample-level block-size labels and tailored training data, whereas HiBlock learns block-size selection directly from the original data. Therefore, Block-R1 is excluded to maintain a consistent data setting for all methods. **(3) Adaptive Block-Based Decoding.** Existing approaches dynamically adjust block-based decoding mainly at inference time (Lu et al., 2026; Shu et al., 2026; Ma et al., 2025b; Huang & Tang, 2025; Lee et al., 2025; Jiang et al., 2026a; Wang et al., 2026b; Han et al., 2026; Zou et al., 2026; Mou et al., 2026). In contrast, HiBlock learns a hierarchical block-size policy during multi-domain RL post-training and transfers the learnt selection ability to different target domain tasks. We provided additional details and discussions in Appendix B.

6 Conclusion

In this paper, we propose **HiBlock**, a novel block-size policy learning framework for multi-domain post-training in dLLMs. HiBlock learns block-size selection by joint optimisation of block-size actions and response generation during multi-domain RL post-training for adaptive block-size decoding across different domains. HiBlock can be readily integrated with existing dLLM RL algorithms. Extensive experiments on 13 benchmarks, 6 dLLMs and 4 RL algorithms validate its effectiveness, providing a promising direction towards more effective multi-domain post-training of dLLMs.

AI USE STATEMENT

AI tools were used solely to assist with grammar checking. We checked the technical content, experimental results, analysis, references, and all claims presented in the paper, and we remain fully responsible for the final manuscript.

ETHICS STATEMENT

This study relies exclusively on publicly accessible pre-trained models, benchmark datasets, and research software used for their intended academic purposes. No new user data were collected, and the work does not involve human participants or human annotation. We therefore do not identify any additional ethical issues specific to the study.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *ICLR*, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. *arXiv preprint arXiv:2107.03006*, 2021.

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A General Theoretical Paradigm to Understand Learning from Human Preferences. *arXiv preprint arXiv:2310.12036*, 2023.

Ashutosh Baheti, Ximing Lu, Faeze Brahman, Ronan Le Bras, Maarten Sap, and Mark Riedl. Leftover Lunch: Advantage-Based Offline Reinforcement Learning for Language Models. *arXiv preprint arXiv:2305.14718*, 2024.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.0: Scaling Up Diffusion Language Models to 100B. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. LLaDA2.1: Speeding Up Text Diffusion via Token Editing. *arXiv preprint arXiv:2602.08676*, 2026.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A Continuous Time Framework for Discrete Denoising Models. *NeurIPS*, 2022.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

Shutong Ding, Ke Hu, Zhenhao Zhang, Kan Ren, Weinan Zhang, Jingyi Yu, Jingya Wang, and Ye Shi. Diffusion-Based Reinforcement Learning via Q-Weighted Variational Policy Optimization. *arXiv preprint arXiv:2405.16173*, 2024.

Yuhao Du, Zhuo Li, Pengyu Cheng, Zhihong Chen, Yuejiao Xie, Xiang Wan, and Anningzhe Gao. Simplify RLHF as Reward-Weighted SFT: A Variational Method. *arXiv preprint arXiv:2502.11026*, 2025.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. KTO: Model Alignment as Prospect Theoretic Optimization. *arXiv preprint arXiv:2402.01306*, 2024.

Yijia Fan, Jing Yang, Mingyu Liu, Kaitong Cai, Jian Wang, Keze Wang, and Jusheng Zhang. Reinforcement Learning for Diffusion LLMs via Energy-Based Gibbs Alignment. In *ACL*, 2026.

Ligong Han, Hao Wang, Han Gao, Kai Xu, and Akash Srivastava. S2D2: Fast Decoding for Diffusion LLMs via Training-Free Self-Speculation. *arXiv preprint arXiv:2603.25702*, 2026.

Haoyu He, Katrin Renz, Yong Cao, and Andreas Geiger. Mdpo: Overcoming the training-inference divide of masked diffusion language models. *arXiv preprint arXiv:2508.13148*, 2025.

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic Preference Optimization without Reference Model. *arXiv preprint arXiv:2403.07691*, 2024.

Chihan Huang and Hao Tang. Ctrldiff: Boosting large diffusion language models with dynamic block prediction and controllable generation. *arXiv preprint arXiv:2505.14455*, 2025.

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.

Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-Fast Language Models Based on Diffusion. *arXiv preprint arXiv:2506.17298*, 2025.

Yan Jiang, Ruihong Qiu, and Zi Huang. Break the Block: Dynamic-size Reasoning Blocks for Diffusion Large Language Models via Monotonic Entropy Descent with Reinforcement Learning. *arXiv preprint arXiv:2605.02263*, 2026a.

Yan Jiang, Ruihong Qiu, and Zi Huang. Block-R1: Rethinking the Role of Block Size in Multi-domain Reinforcement Learning for Diffusion Large Language Models. *arXiv preprint arXiv:2605.11726*, 2026b.

Jihoon Lee, Hoyeon Moon, Kevin Zhai, Arun Kumar Chithanar, Anit Kumar Sahu, Soummya Kar, Chul Lee, Souradip Chakraborty, and Amrit Singh Bedi. Test-time scaling in diffusion llms via hidden semi-autoregressive experts. *arXiv preprint arXiv:2510.05040*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution. In *ICML*, 2024.

Cheng Lu, Huayu Chen, Jianfei Chen, Hang Su, Chongxuan Li, and Jun Zhu. Contrastive Energy Prediction for Exact Energy-Guided Diffusion Sampling in Offline Reinforcement Learning. In *ICML*, 2023.

Guanxi Lu, Hao Mark Chen, Yuto Karashima, Zhican Wang, Daichi Fujiki, and Hongxiang Fan. AdaBlock-dLLM: Semantic-Aware Diffusion LLM Inference via Adaptive Block Size. In *ICLR*, 2026.

Haitong Ma, Tianyi Chen, Kai Wang, Na Li, and Bo Dai. Efficient Online Reinforcement Learning for Diffusion Policy. In *ICML*, 2025a.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, Zhenzhong Lan, Junbo Zhao, Jianguo Li, and Da Zheng. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025b.

Chenlin Meng, Kristy Choi, Jiaming Song, and Stefano Ermon. Concrete Score Matching: Generalized Score Matching for Discrete Data. *NeurIPS*, 2022.

Xingyu Mou, Zijin Huang, Tianze Zhang, Yuxin Ma, Lanning Wei, Zengfeng Huang, Da Zheng, and Lun Du. Beyond Block Boundaries: Multi-Block Editing for Diffusion Large Language Models. *arXiv preprint arXiv:2607.22663*, 2026.

Youssef Mroueh. Reinforcement Learning with Verifiable Rewards: GRPO's Effective Loss, Dynamics, and Success Amplification. *arXiv preprint arXiv:2503.06639*, 2025.

Zanlin Ni, Shenzhi Wang, Yang Yue, Tianyu Yu, Weilin Zhao, Yeguo Hua, Tianyi Chen, Jun Song, Cheng Yu, Bo Zheng, and Gao Huang. The Flexibility Trap: Rethinking the Value of Arbitrary Order in Diffusion Language Models. In *ICML*, 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.

Jingyang Ou, Jiaqi Han, Minkai Xu, Shaoxuan Xu, Jianwen Xie, Stefano Ermon, Yi Wu, and Chongxuan Li. Principled RL for Diffusion LLMs Emerges from a Sequence-Level Perspective. In *ICLR*, 2026.

Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *arXiv preprint arXiv:1910.00177*, 2019.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *NeurIPS*, 2023.

Kevin Rojas, Jiahe Lin, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, Molei Tao, and Wei Deng. Improving reasoning for diffusion language models via group diffusion policy optimization. In *ICLR*, 2026.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. *arXiv preprint arXiv:2406.07524*, 2024.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *ICML*, 2015.

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, Y Wu, et al. Deepseekmath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. Simplified and Generalized Masked Diffusion for Discrete Data. In *NeurIPS*, 2024.

Yingte Shu, Yuchuan Tian, Chao Xu, Yunhe Wang, and Hanting Chen. Deferred commitment decoding for diffusion language models. *arXiv preprint arXiv:2601.02076*, 2026.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning Using Nonequilibrium Thermodynamics. In *ICML*, 2015.

Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. Wd1: Weighted policy optimization for reasoning in diffusion language models. In *ICLR*, 2026a.

Yao Tang, Li Dong, Yaru Hao, Qingxiu Dong, Furu Wei, and Jiatao Gu. Multiplex thinking: Reasoning via token-wise branch-and-merge. *arXiv preprint arXiv:2601.08808*, 2026b.

Chaoqi Wang, Yibo Jiang, Chenghao Yang, Han Liu, and Yuxin Chen. Beyond Reverse KL: Generalizing Direct Preference Optimization with Diverse Divergence Constraints. *arXiv preprint arXiv:2309.16240*, 2023.

Chenyu Wang, Masatoshi Uehara, Yichun He, Amy Wang, Tommaso Biancalani, Avantika Lal, Tommi Jaakkola, Sergey Levine, Hanchen Wang, and Aviv Regev. Fine-Tuning Discrete Diffusion Models via Reward Optimization with Applications to DNA and Protein Design. *arXiv preprint arXiv:2410.13643*, 2024.

Chenyu Wang, Paria Rashidinejad, DiJia Su, Song Jiang, Sid Wang, Siyan Zhao, Cai Zhou, Shannon Zejiang Shen, Feiyu Chen, Tommi Jaakkola, Yuandong Tian, and Bo Liu. SPG: Sandwiched Policy Gradient for Masked Diffusion Language Models. In *ICLR*, 2026a.

Danny Wang, Ruihong Qiu, and Zi Huang. When to Commit? Towards Variable-Size Self-Contained Blocks for Discrete Diffusion Language Models. *arXiv preprint arXiv:2604.23994*, 2026b.

Guanghan Wang, Gilad Turok, Yair Schiff, Marianne Arriola, and Volodymyr Kuleshov. d2: Improving Reasoning in Diffusion Language Models via Trajectory Likelihood Estimation. In *ICML*, 2026c.

Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. In *ICLR*, 2026d.

Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning. *arXiv preprint arXiv:2208.06193*, 2022.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dLLM v2: Efficient Block-Diffusion LLM. In *ICLR*, 2026a.

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. In *ICLR*, 2026b.

Wei Xiong, Jiarui Yao, Yuhui Xu, Bo Pang, Lei Wang, Doyen Sahoo, Junnan Li, Nan Jiang, Tong Zhang, Caiming Xiong, and Hanze Dong. A Minimalist Approach to LLM Reasoning: From Rejection Sampling to Reinforce. *arXiv preprint arXiv:2504.11343*, 2025.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A Diverse, Challenging, and Verifiable Synthetic Dataset for Coding. In *Findings of ACL*, 2025.

Jingyi Yang, Yuxian Jiang, Xuhao Hu, Shuang Cheng, Biqing Qi, and Jing Shao. Dare: Diffusion large language models alignment and reinforcement executor. *arXiv preprint arXiv:2604.04215*, 2026.

Ling Yang, Ye Tian, Bowen Li, Xinchen Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. Mmada: Multimodal Large Diffusion Language Models. *arXiv preprint arXiv:2505.15809*, 2025.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint arXiv:2410.14157*, 2024.

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.

Oussama Zekri and Nicolas Boullé. Fine-Tuning Discrete Diffusion Models with Policy Gradient Methods. *arXiv preprint arXiv:2502.01384*, 2025.

Shiyuan Zhang, Weitong Zhang, and Quanquan Gu. Energy-Weighted Flow Matching for Offline Reinforcement Learning. *arXiv preprint arXiv:2503.04975*, 2025.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling Reasoning in Diffusion Large Language Models via Reinforcement Learning. In *NeurIPS*, 2025.

Jianyuan Zhong, Kaibo Wang, Ding Ding, Zijin Feng, Haoli Bai, Yang Xiang, Jiacheng Sun, and Qiang Xu. Stabilizing Reinforcement Learning for Diffusion Language Models. *arXiv preprint arXiv:2603.06743*, 2026.

Banghua Zhu, Hiteshi Sharma, Felipe Vieira Frujeri, Shi Dong, Chenguang Zhu, Michael I Jordan, and Jiantao Jiao. Fine-Tuning Language Models with Advantage-Induced Policy Alignment. *arXiv preprint arXiv:2306.02231*, 2023.

Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-Reduced Preference Optimization for Large Language Diffusion Models. In *ACL*, 2026.

Ying Zhu, Jiaxin Wan, Xiaoran Liu, Siyang He, Qiqi Wang, Xu Guo, Tianyi Liang, Zengfeng Huang, Ziwei He, and Xipeng Qiu. Dirl: An efficient post-training framework for diffusion language models. *arXiv preprint arXiv:2512.22234*, 2025.

Shun Zou, Yong Wang, Zehui Chen, Lin Chen, Chongyang Tao, Feng Zhao, and Xiangxiang Chu. Breaking Block Boundaries: Anchor-based History-stable Decoding for Diffusion Large Language Models. In *ACL*, 2026.

A Appendix Overview

This appendix provides additional details and analyses of HiBlock:

- Appendix B presents detailed related work on dLLMs, reinforcement learning, and adaptive block-based generation.
- Appendix C provides implementation and reproducibility details, including dataset statistics, evaluation settings, and the complete HiBlock algorithm in Algorithm 1.
- Appendix D analyses the sensitivity to the block-size reward coefficient α.
- Appendix E visualises the cross-domain block-size conflict.
- Appendix F analyses the scalability of HiBlock to longer generation length.
- Appendix G reports a multi-seed robustness analysis under the same RL setting.
- Appendix H presents a case study of task-dependent block-size selection.
- Appendix I presents a detailed limitation analysis.

B Detailed Related Work

B.1 Diffusion Large Language Models

Diffusion Large Language Models (dLLMs) generate text by repeatedly recovering masked or corrupted tokens rather than predicting every token in a strictly left-to-right order (Sohl-Dickstein et al., 2015; Campbell et al., 2022; Meng et al., 2022; Austin et al., 2021; Lou et al., 2024; Shi et al., 2024; Sahoo et al., 2024; Nie et al., 2024; Zhu et al., 2026; Ye et al., 2025; Nie et al., 2025). This formulation allows multiple masked positions to be denoised jointly while maintaining dependencies through iterative refinement. Moreover, Dream (Ye et al., 2025), Mercury (Inception Labs et al., 2025), and other LLaDA-style models (Nie et al., 2025) also demonstrate the potential of diffusion-based text generation. Recent advanced models, including LLaDA2.0 (Bie et al., 2025) and LLaDA2.1 (Bie et al., 2026), further extend the model scale and improve dLLM reasoning.

Many dLLMs adopt block-based semi-autoregressive decoding, where the response is divided into consecutive blocks, the blocks are completed sequentially, and masked positions within each active block are denoised jointly (Arriola et al., 2025). The block size determines both the number of positions processed within a block and the number of sequential block transitions required for generation. Smaller blocks provide finer-grained conditioning on previously completed tokens, whereas larger blocks reduce the number of block transitions. Since reasoning tasks may differ in their dependency structures and required intermediate verification, a single predefined block size may not be suitable for every task (Xiong et al., 2025; Ye et al., 2024; Tang et al., 2026b). This motivates learning block-size selection as part of the dLLM policy rather than treating it solely as an externally specified decoding configuration.

B.2 Reinforcement Learning

Reinforcement learning (RL) has become a widely used post-training approach for improving language-model reasoning. Unlike supervised fine-tuning method, which trains models to imitate reference responses, RL directly optimises generated outputs using task rewards such as answer correctness, executable-code success, constraint satisfaction, and format validity. Representative policy optimisation methods include TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017), and Group Relative Policy Optimization (GRPO) (Shao et al., 2024).

GRPO removes the need for a separately trained value model by sampling a group of responses for the same prompt and estimating advantages from their relative rewards. This group-based formulation is particularly suitable for reasoning tasks because multiple valid solution paths may produce the same correct answer. Related analyses connect reasoning-oriented RL to reward-weighted regression, selective fine-tuning, and advantage-weighted learning (Mroueh, 2025; Peng et al., 2019; Zhu et al., 2023; Du et al., 2025; Baheti et al., 2024). Rejection-sampling approaches such as RAFT (Xiong et al., 2025) similarly increase the likelihood of high-reward generations. Preference-based objectives, including DPO and its extensions (Rafailov et al., 2023; Azar et al., 2023; Etha-

yarajh et al., 2024; Wang et al., 2023; Hong et al., 2024), provide another way to learn from comparative feedback without explicit value-function estimation.

B.3 Reinforcement Learning for Block-Based dLLM Reasoning

Recent advancements in RL methods adapt policy optimisation to diffusion-based response generation in dLLMs. Earlier directions include multimodal diffusion reasoning such as MMaDA (Yang et al., 2025), diffusion policy optimisation (Ding et al., 2024; Wang et al., 2022; 2024; Ma et al., 2025a; Zhang et al., 2025; Lu et al., 2023), reward-guided diffusion optimisation (Zekri & Boullé, 2025; Huang et al., 2025), and PPO-style diffusion reasoning methods such as TraceRL (Wang et al., 2026d). More recent work primarily builds on group-relative optimisation, including *d1* (Zhao et al., 2025), *d2* (Wang et al., 2026c), *wd1* (Tang et al., 2026a), JustGRPO (Ni et al., 2026), GDPO (Rojas et al., 2026), MDPO (He et al., 2025), ESPO (Ou et al., 2026), SPG (Wang et al., 2026a), StableDRL (Zhong et al., 2026), DiRL (Zhu et al., 2025), and DARE (Yang et al., 2026). Diffusion-Gibbs Alignment provides an alternative distribution matching formulation for dLLM RL (Fan et al., 2026).

These methods differ mainly in how the likelihood of a diffusion response is estimated and how the group-relative advantage is applied. Token-level approaches, including *d1*, *d2*, *wd1*, and JustGRPO, optimise token-wise likelihood terms or token-level weighting derived from completion rewards (Zhao et al., 2025; Wang et al., 2026c; Tang et al., 2026a; Ni et al., 2026). Sequence-level approaches, including GDPO, MDPO, ESPO, SPG, StableDRL, DiRL, and DARE, instead construct objectives over complete responses or denoising trajectories (Rojas et al., 2026; He et al., 2025; Ou et al., 2026; Wang et al., 2026a; Zhong et al., 2026; Zhu et al., 2025; Yang et al., 2026). Despite these differences, most methods generate all training rollouts under one predefined block size. They therefore optimise how the dLLM responds under a fixed block-based decoding structure but do not learn which block size should be selected for an individual task. Block-R1 (Jiang et al., 2026b) addresses block-size differences through a different dataset-oriented approach. It evaluates teacher and student dLLMs under multiple candidate block sizes and assigns each retained training example a best-improved block-size label based on the teacher–student reward gap. This produces sample-level block-size supervision but requires additional rollout computation, reward evaluation, filtering, and label construction before RL post-training. Its performance also depends on the specially constructed training dataset and the assigned block-size labels.

In contrast, HiBlock follows a different direction. It does not require predefined block-size annotations or specially tailored training samples. Instead, the high-level block-size policy and low-level block-based response-generation policy are jointly learnt through group-relative optimisation using the original multi-domain training data. To compare the considered RL methods under the same training data and without additional block-size labels, Block-R1 is excluded from the RL baselines in our experiments.

B.4 Adaptive Block-Based Generation Methods

Several recent studies improve block-based dLLM generation by modifying the inference process. AdaBlock-dLLM (Lu et al., 2026) dynamically shortens a decoding block by detecting newline tokens under predefined confidence conditions. AHD enables early cross-block decoding based on token stability (Zou et al., 2026), Multi-Block Editing revisits decoded tokens using cross-block context (Mou et al., 2026), and S2D2 combines block diffusion drafting with autoregressive verification for training-free self-speculative decoding (Han et al., 2026). Deferred Commitment Decoding (Shu et al., 2026) uses a confidence-aware sliding window to postpone uncertain token decisions until more context becomes available. Other methods, including dInfer (Ma et al., 2025b), CtrlDiff (Huang & Tang, 2025), and hidden semi-autoregressive experts for test-time scaling (Lee et al., 2025), improve decoding efficiency, controllability, or test-time computation. VSB (Wang et al., 2026b) selects variable block boundaries by examining whether token predictions remain consistent when future context is removed. *b1* (Jiang et al., 2026a) further learns dynamic reasoning-block boundaries by aligning the block structure with the progression of the generated reasoning process.

However, their block-size decisions generally depend on **externally specified block-control tokens, such as block-ending tokens (Jiang et al., 2026a), or hand-crafted cues, such as locating the next newline character (Lu et al., 2026).** Consequently, most of these methods do not enable dLLMs to autonomously learn block-size selection according to the input domain and task. This may

also create a mismatch between post-training under a predefined block size and inference with adaptive block configurations. Moreover, these methods are not designed to transfer block-size selection across tasks from different domains, limiting their ability to support cross-domain generalisation.

In contrast, HiBlock learns block-size selection directly during multi-domain RL post-training. **Its high-level policy selects a model-native block-size action according to the input prompt, after which its low-level policy generates the response under the selected block-based decoding structure.** The learnt selection ability can therefore be transferred to different target domain tasks without predefined block-control tokens or fixed hand-crafted selection rules that are generally required in existing methods (Jiang et al., 2026a; Lu et al., 2026).

C Additional Implementation Details

Our implementation code with our multi-domain post-training and evaluation framework is fully provided at anonymous.4open.science/r/HiBlock-2026. In this section, we further provide more implementation details to facilitate reproducibility.

Table 6: **Dataset statistics for the HiBlock experiments.** Datasets without an official training split, such as HumanEval, MMLU, and MMLU-Pro, are used only for evaluation. All 13 benchmarks participate in evaluation. General capabilities cover knowledge-intensive, commonsense, and scientific-comprehension tasks, while advanced reasoning covers benchmarks requiring more complex knowledge integration and multi-step inference.

Category	Dataset	Train Size	Test Size
Code Generation	MBPP	374	500
	HumanEval	N/A	164
	KodCode	9,285	500
Mathematical Reasoning	GSM8K	7,473	1,319
	MATH500	7,500	500
	Countdown	240,632	256
Logical Constraint Reasoning	Knights-and-Knaves (KK)	6,200	700
	Sudoku	1,000,000	256
General Capabilities	HellaSwag	39,905	10,003
	MMLU	N/A	14,042
	ARC-E	2,251	2,376
Advanced Reasoning	MMLU-Pro	N/A	12,032
	ARC-C	1,119	1,172

C.1 Datasets and Cross-Domain Protocol

Table 6 presents the statistics of the 13 benchmarks used in our experiments. Throughout this paper, we follow existing work (Jiang et al., 2026b) to regard a domain as a benchmark-level task distribution. Countdown, GSM8K, MATH500, KodCode, Sudoku, and Knights-and-Knaves form the six source domains for multi-domain post-training. Let $\mathcal{D}_{\text{src}}$ denote this source-domain set. We define the fixed per-domain sampling size as $N_{\text{bal}} = \min_{d \in \mathcal{D}_{\text{src}}} N_d = 6{,}200$, corresponding to the smallest training set among the six source domains, following the same implementation in existing multi-domain RL for dLLM benchmark (Jiang et al., 2026b). HumanEval, MMLU, and MMLU-Pro are used only for evaluation because they do not provide suitable training splits for post-training. MBPP is also excluded from post-training because its training set is substantially smaller than those of the six source domains. For a target belonging to the six source domains, we follow a leave-one-domain-out protocol. The target domain is removed, and the remaining five source domains are used for post-training. For all other target benchmarks, post-training uses all six source domains. Therefore, the corresponding target-domain data are never included in post-training.

C.2 Training Configuration

Unless otherwise stated, all experiments are conducted on four AMD Instinct MI300X GPUs, each with 192GB of memory. LLaDA-8B-Instruct (Nie et al., 2025) is used as the default dLLM backbone, and StableDRL (Zhong et al., 2026) serves as the primary RL algorithm due to its empirically

strong performance. We further integrate HiBlock with *d1* (Zhao et al., 2025) and *wd1* (Tang et al., 2026a) to examine its compatibility with different dLLM RL objectives. The SFT baseline adopts the denoising cross-entropy implementation of *d1* (Zhao et al., 2025).

We follow existing work (Zhao et al., 2025; Tang et al., 2026a) to apply Low-Rank Adaptation with rank 128, scaling factor 64, and dropout 0.05. AdamW (Loshchilov & Hutter, 2017) is used with a learning rate of 3×10^{-6}, $\beta_1 = 0.9$, $\beta_2 = 0.99$, weight decay 0.1, and a gradient clipping threshold of 0.2 following existing dLLM work. The per-device training batch size is 12, and the gradient accumulation step is 1. Training uses bfloat16 precision, 4-bit quantisation, and Flash Attention 2 (Dao, 2024). For domains without an official training split, such as HumanEval, we use training data from a closely related domain with similar code task characteristics. Specifically, KodCode Xu et al. (2025) is used as the default code generation training set following GDPO Rojas et al. (2026). The detailed configurations are in Table 7.

C.3 HiBlock Configuration

We follow the most common practice in prior dLLM studies (Zhao et al., 2025; Tang et al., 2026a; Rojas et al., 2026; Jiang et al., 2026b) to adopt a default seed of 42 to ensure a fair comparison under a consistent setting for all methods and additionally report results over three random seeds in Appendix G. HiBlock adopts the candidate block-size set $\mathcal{C}_0 = \{4, 8, 16, 32, 64, 128\}$. This is because these candidates evenly divide the default maximum generation length of 256 and are commonly adopted in block-based dLLM methods (Jiang et al., 2026b). The rollout-group size is set to $G = 6$, allowing each candidate block size to be evaluated exactly once for every input task through sampling without replacement.

The block-size reward coefficient is set to $\alpha = 0.3$, the KL coefficient to $\beta = 0.0$, and the clipping coefficient to $\epsilon = 0.5$. The block-size action and its corresponding response are optimised within the same dLLM. Algorithm 1 provides the complete procedure for balanced block-size exploration and hierarchical policy optimisation.

C.4 Evaluation Protocol

Following existing work (Jiang et al., 2026b), we treat each benchmark as an individual domain for multi-domain post-training and cross-domain evaluation. Accordingly, our cross-domain evaluation focuses on transfer across benchmark level domains rather than holding out entire task families. All methods are evaluated under a consistent zero-shot setting. Unless otherwise specified, the maximum response length is 256, and generation uses 128 diffusion steps to ensure a consistent comparison with existing methods (Jiang et al., 2026a; Zhao et al., 2025; Tang et al., 2026a). The 128 diffusion steps refer to the total denoising budget for each generated response. Fixed-block baselines use block size 32, following common settings in existing dLLM RL methods (Zhao et al., 2025; Tang et al., 2026a; Zhong et al., 2026; Ni et al., 2026).

We report pass@1 accuracy for benchmarks with deterministic final answers. Sudoku uses cell level accuracy, measured as the ratio of correctly filled initially empty cells, while Knights-and-Knaves uses character role assignment accuracy, measured as the ratio of correctly assigned character roles following existing work (Jiang et al., 2026b).

During inference, HiBlock evaluates the normalised likelihood of every valid block-size action and selects the action with the highest likelihood before response generation. In the Oracle analysis in Section 4.3, every test example is evaluated under all six candidate block sizes. When multiple candidates achieve the same maximum task reward, the largest block size is selected to remain consistent with the block size efficiency preference.

C.5 Task Reward Functions

The domain-specific task rewards follow the evaluation requirements of their corresponding source domains in a consistent formulation with existing work (Jiang et al., 2026a; Zhao et al., 2025). During HiBlock post-training, each task reward is combined with the block-size reward according to Eq. 12. Note that our reward weight for all tasks remains the same with our baseline for a fair and consistent comparison (Jiang et al., 2026a; Zhao et al., 2025; Tang et al., 2026a).

Countdown. The final answer span is parsed to obtain an arithmetic expression. A reward of 1.0 is assigned when the expression uses exactly the provided numbers and evaluates to the target value. An extractable but invalid or incorrect expression receives 0.1, while an output without a valid expression receives 0.

GSM8K. The task reward consists of formatting, answer-type, and correctness components. The formatting component evaluates the required reasoning and answer structure. The answer-type component checks whether the extracted answer is a valid integer, while the correctness component compares it with the reference answer.

MATH500. The formatting component checks whether the response contains a valid answer span and a boxed final answer. Mathematical correctness is evaluated using an equivalence checker between the extracted prediction and the reference solution.

KodCode. The code-format component verifies whether the response contains a syntactically valid Python code block. Valid programs are executed against the provided tests, and the execution reward is determined by the proportion of passed test cases.

Sudoku. The generated response is parsed as a 4×4 grid. The reward is computed as the proportion of originally empty cells whose predicted values match the reference solution.

Knights-and-Knaves. The response is parsed into character–role assignments. The primary reward is the proportion of characters correctly identified as knights or knaves, together with a default formatting component from existing methods (Jiang et al., 2026b) for producing a valid structured answer and fair comparison.

C.6 Default Hyperparameter Settings

Table 7 summarises the default model, optimisation, and detailed configurations used in HiBlock. Our multi-domain RL settings and hyperparameter choices follow existing benchmarks for a consistent setting (Jiang et al., 2026b).

Table 7: **Default hyperparameter settings of HiBlock following prior dLLM RL work (Zhao et al., 2025; Tang et al., 2026a).** Our main experiments are implemented on four AMD Instinct MI300X GPUs, each with 192GB of memory.

Parameter	Default Value
Model and Precision	
Default backbone	LLaDA-8B-Instruct
Precision	bfloat16
4-bit quantisation	true
Flash Attention 2	true
LoRA rank	128
LoRA scaling factor	64
LoRA dropout	0.05
Optimisation	
Seed	42
Learning rate	3×10^{-6}
Adam β_1	0.9
Adam β_2	0.99
Weight decay	0.1
Gradient clipping threshold	0.2
Warm-up ratio	0.0001
KL coefficient β	0.0
Clipping coefficient ϵ	0.5
Training and Generation	
Per-device training batch size	12
Gradient accumulation steps	1
Maximum training steps	5,000
Checkpoint evaluation interval	100
Maximum prompt length	200
Maximum response length	256
Diffusion steps	128
Rollout-group size G	6
Candidate block sizes $\mathcal{C}_0$	$\{4, 8, 16, 32, 64, 128\}$
Block-size reward coefficient α	0.3
Prompt masking probability	0.15
Remasking strategy	low confidence

Algorithm 1: HiBlock Hierarchical Policy Optimisation

Input: Domain datasets $\{\mathcal{D}_d\}_{d=1}^{D}$, domain-specific task reward $R_{\text{task}}^{(d)}$, candidate block sizes $\mathcal{C}_0$, generation length L, rollout number G, dLLM policy π_θ, and coefficients α, β, ϵ, and δ.
Output: Optimised HiBlock policy $\pi_{\theta^\star}$.

```
// Stage 1: Balanced Multi-Domain Data Pre-processing
```
Construct $\mathcal{D}_{\text{bal}}$ by Eq. 9;
```
// Stage 2: Hierarchical Block-Size Policy Construction
```
Construct $\mathcal{C} = \{c \in \mathcal{C}_0 : c > 0,\ c \mid L\}$;
Construct $\mathcal{Z}_{\mathcal{C}} = \{\text{Enc}(c) : c \in \mathcal{C}\}$;
Form the hierarchical policy by Eq. 10;
Initialise $\theta_{\text{old}} \leftarrow \theta$;
for *each training iteration* **do**
  Sample a balanced mini-batch $\mathcal{B} \subseteq \mathcal{D}_{\text{bal}}$;
  for *each input task* $x \in \mathcal{B}$ *from domain* d **do**
    ```
    // Stage 3: Balanced Block-Size Exploration
    ```
    Initialise the remaining candidate set $\mathcal{C}_{\text{rem}} \leftarrow \mathcal{C}$;
    for $g = 1, \ldots, G$ **do**
      if $\mathcal{C}_{\text{rem}} = \varnothing$ **then**
        Reset $\mathcal{C}_{\text{rem}} \leftarrow \mathcal{C}$;
      Randomly select c_g from $\mathcal{C}_{\text{rem}}$;
      Set $\mathbf{z}_g \leftarrow \text{Enc}(c_g)$;
      Remove c_g from $\mathcal{C}_{\text{rem}}$;
      Generate $\mathbf{y}_g$ by Eq. 11;
      Compute $R_{d,g}$ by Eq. 12;
      Set $\mathcal{I}_g \leftarrow \mathcal{I}_g^{\text{blk}} \cup \mathcal{I}_g^{\text{resp}}$;
    ```
    // Stage 4: Joint Hierarchical Policy Optimisation
    ```
    Compute $\widehat{A}_{d,g}$ for all g by Eq. 13;
    Compute $N_x \leftarrow \sum_{g=1}^{G} |\mathcal{I}_g|$;
    Compute $\rho_{g,i}(\theta)$ for all g and $i \in \mathcal{I}_g$, conditioning response positions on $\mathbf{z}_g$;
  Update θ by maximising Eq. 14 over $\mathcal{B}$;
  Update $\theta_{\text{old}} \leftarrow \theta$;
Set $\theta^\star \leftarrow \theta$;
return $\pi_{\theta^\star}$;
```

Table 8: **Sensitivity analysis to the block-size reward coefficient $\alpha$.** We evaluate HiBlock with different values of $\alpha$ under the same multi-domain post-training and cross-domain zero-shot evaluation settings. **Fixed size baseline** denotes the Vanilla multi-domain StableDRL baseline with block size 32 from its official implementation. Test accuracy is highlighted by **best** and runner-up among the HiBlock variants.

| Method | Countdown | GSM8K | MATH500 | Sudoku | Avg. |
|---|---|---|---|---|---|
| Fixed size baseline | 30.08 | 57.24 | 28.20 | 9.77 | 31.32 |
| HiBlock ($\alpha = 0$) | 57.89 | 80.97 | 33.40 | 24.73 | 49.25 |
| HiBlock ($\alpha = 0.1$) | 56.67 | 81.35 | 34.00 | 26.17 | 49.55 |
| HiBlock ($\alpha = 0.3$) | **59.76** | **81.58** | 34.40 | 25.80 | **50.39** |
| HiBlock ($\alpha = 0.5$) | 55.28 | 81.43 | **34.60** | **26.32** | 49.41 |
| HiBlock ($\alpha = 1.0$) | 55.50 | 80.89 | 34.00 | 25.78 | 49.04 |

## D Hyperparameter Sensitivity Analysis

Table 8 examines the sensitivity of HiBlock to the block-size reward coefficient $\alpha$. **HiBlock remains stable across different values of $\alpha$** and consistently outperforms the fixed-size baseline on all four benchmarks. Setting $\alpha = 0$ causes only a moderate reduction, indicating that the improvements primarily arise from learning task-dependent block-size selection rather than solely from the block-size reward. Values between $0.1$ and $0.5$ achieve comparable performance, with $\alpha = 0.3$ obtaining the highest average accuracy. Increasing $\alpha$ to $1.0$ leads to a slight decrease because an excessively strong preference for larger blocks may reduce reasoning quality. Overall, the results show that **HiBlock is not sensitive to the choice of $\alpha$.**
```

E VISUALISATION OF BLOCK-SIZE CONFLICT

Figure 4 visualises the mean task reward achieved by LLaDA-8B-Instruct on the training set of each domain under different block sizes. The results show that rewards vary differently with block size across domains, and **no single block size is consistently preferred**. This reveals a **cross-domain block-size conflict**, where different domains favour different decoding strategies. For example, GSM8K favours smaller blocks, whereas Knights-and-Knaves performs better with relatively larger blocks. These rich block-size reward distributions motivate HiBlock to evaluate diverse block sizes within each GRPO rollout group and learn **task-dependent block-size selection** from trajectory rewards, rather than enforcing one fixed block size across all domains.

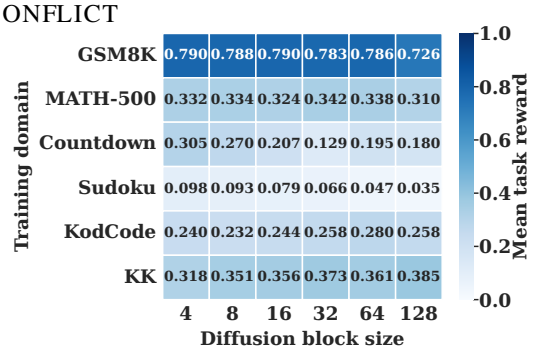


Figure 4: **Task rewards under different block sizes across training domains.** Each cell reports the mean task reward of LLaDA-8B-Instruct on the corresponding training set. Different domains exhibit distinct block-size preferences. Such informative reward distributions are often neglected by existing dLLM RL methods but can be exploited to learn block-size selection.

F SCALABILITY TO LONGER GENERATION LENGTH

We further examine whether HiBlock scales to a longer maximum generation length. Vanilla and HiBlock are post-trained by the default StableDRL algorithm on the same multi-domain data as in Table 2 using generation lengths of 256 and 512. As shown in Table 9, increasing the generation length generally improves both methods by providing more space for reasoning. HiBlock consistently outperforms the vanilla under both lengths, demonstrating that its hierarchical block-size policy remains effective for longer response generation.

Table 9: **Scaling to longer generation lengths.** Vanilla and HiBlock are post-trained on the same multi-domain data using maximum generation lengths of 256 and 512.

Method	GSM8K		MATH500		Sudoku	
	256	512	256	512	256	512
Vanilla	57.24	73.62	28.20	31.80	9.77	14.06
HiBlock	81.58	84.23	34.40	35.60	25.80	26.90

G MULTI-SEED ROBUSTNESS ANALYSIS

To examine the robustness of HiBlock across different random seeds, we repeat the main multi-domain RL experiments using three random seeds under the default LLaDA-8B-Instruct setting. Following the multi-domain evaluation in Table 2, we compare the baseline multi-domain RL with HiBlock on four representative benchmarks, including Countdown, GSM8K, MATH500, and Sudoku, following the dataset choice in most existing dLLM studies (Zhao et al., 2025; Jiang et al., 2026a). Both methods use the same multi-domain training data, cross-domain evaluation protocol and base RL algorithm (StableDRL), with Vanilla using the fixed block size 32 and HiBlock learning adaptive block-size selection. All training and evaluation configurations are kept unchanged across runs except for the random seed.

Table 10: **Multi-seed robustness analysis.** Results are reported as mean $\pm$ standard deviation over three random seeds generated using NumPy in Python. Vanilla and HiBlock use the same multi-domain training data and cross-domain evaluation setting as in Table 2.

Method	Countdown	GSM8K	MATH500	Sudoku
Vanilla	30.42 ± 4.62	55.82 ± 0.58	30.10 ± 1.60	11.24 ± 3.47
HiBlock	$\mathbf{57.39 \pm 2.31}$	$\mathbf{81.40 \pm 0.61}$	$\mathbf{34.05 \pm 0.58}$	$\mathbf{26.03 \pm 0.35}$

As shown in Table 10, HiBlock consistently achieves higher mean performance than baseline multi-domain RL on all four benchmarks. The relatively small standard deviations further indicate that the improvements of HiBlock are stable across different random seeds. Notably, to the best of our

Table 11: **Case study of adaptive block-size selection.** The fixed-size baseline and HiBlock answer the same mathematical reasoning problem from GSM8K. The fixed-size baseline uses block size 32, while HiBlock selects block size 4. Grey vertical bars indicate illustrative decoding-block boundaries. Red text highlights the disrupted reasoning transition, while green text highlights the preserved intermediate state. For readability, special tokens such as end-of-sentence markers and exact tokenizer-level segmentation are hidden.

Method	Block Size	Step-by-Step Generation with Block Boundaries	Result
Input: The pizzeria sells small pizzas for \$2 and large pizzas for \$8. They sold \$40 in pizzas. If they sold 8 small pizzas, how many large pizzas did they sell?			
Fixed-size	32	**Step 1:** Eight small pizzas cost $8 \times \$2 = \$16$. **Step 2:** Subtracting this amount from the total, the remaining revenue from | large pizzas is \$40. **Step 3:** Since each large pizza costs \$8, the number of large pizzas is $\$40 \div \$8 = 5$.	**Incorrect**
HiBlock	**4 (selected)**	**Step 1:** Eight small | pizzas cost $8 \times \$2$ | $= \$16$. **Step 2:** The remaining | revenue is $\$40 - \$16$ | $= \$24$. **Step 3:** Since each | large pizza costs \$8, the number | of large pizzas is $\$24 \div \$8 = 3$.	**Correct**

knowledge, many dLLM baseline studies (Zhao et al., 2025; Rojas et al., 2026; He et al., 2025; Jiang et al., 2026a; Zhong et al., 2026) do not generally report multi-seed results, so our three-seed evaluation therefore provides an additional robustness assessment beyond their standard evaluation settings. These results provide additional evidence that the cross-domain improvements observed in HiBlock are not specific to a single seed.

H Detailed Case Study

Table 11 compares the vanilla fixed-size decoding baseline and HiBlock on the same mathematical reasoning problem from GSM8K with LLaDA-8B-Instruct. With the fixed block size of 32, the decoding block spans a long reasoning segment and the subsequent generation fails to preserve the previously derived intermediate state. As a result, the model incorrectly reuses the original total revenue of \$40 rather than the remaining revenue after the small-pizza sales, which disrupts the reasoning chain and produces the incorrect answer of five.

In contrast, HiBlock selects block size 4 for the same input. Although the block boundaries do not necessarily align with complete sentences, the more frequent conditioning on completed tokens better preserves the intermediate values of \$16 and \$24 across successive blocks. The model consequently maintains a more coherent calculation and obtains the correct answer of three. This case further supports that task-dependent block-size selection can better preserve reasoning continuity than applying one fixed block size to every input task.

I Limitations

Following most existing block-based dLLM settings, HiBlock constrains block-size selection to a predefined candidate set whose values evenly divide the generation length. This design simplifies block partitioning and maintains compatibility with commonly used decoding implementations. However, it potentially limits the model from selecting irregular or extreme block sizes outside the candidate set, even when such sizes might be beneficial for particular tasks. The learnt block-size policy is therefore adaptive only within the predefined search space. Extending HiBlock to support arbitrary block sizes and more flexible block partitioning remains an important direction, yet is beyond the current scope.