

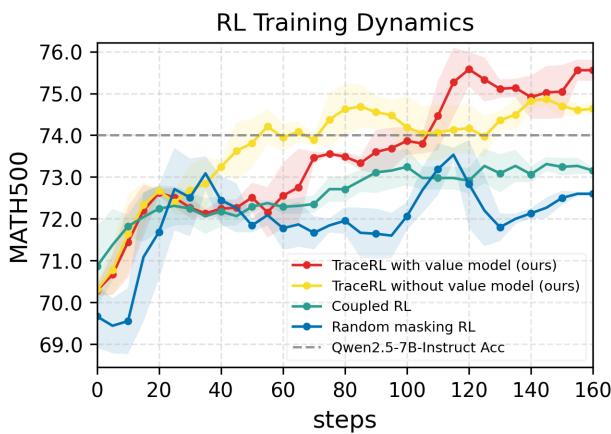
Revolutionizing Reinforcement Learning Framework for Diffusion Large Language Models

Yinjie Wang^{1,2*}, Ling Yang^{1*} , Bowen Li¹, Ye Tian¹, Ke Shen and Mengdi Wang¹

*Equal contributions, Corresponding author, ¹Princeton University, ²University of Chicago

Code: <https://github.com/Gen-Verse/dLLM-RL> Models:  TraDo-4B/8B

We propose **TraceRL**, a trajectory-aware reinforcement learning framework for diffusion language models (DLMs) that incorporates preferred inference trajectory into post-training, and is applicable across different architectures. Equipped with a diffusion-based value model that enhances training stability, we demonstrate improved reasoning performance on complex math and coding tasks. Besides, it can also be applied to adapt block-specific models to larger blocks, which improves sampling flexibility. Employing TraceRL, we derive a series of state-of-the-art diffusion language models, namely **TraDo**. Although smaller than 7B-scale AR models, TraDo-4B-Instruct still consistently outperforms them across complex math reasoning tasks. TraDo-8B-Instruct achieves relative accuracy improvements of 6.1% over Qwen2.5-7B-Instruct and 51.3% over Llama3.1-8B-Instruct on mathematical reasoning benchmarks. Through curriculum learning, we also derive the first long-CoT DLM, outperforming Qwen2.5-7B-Instruct on MATH500 with an 18.1% relative accuracy gain. To facilitate reproducible research and practical applications, we release a comprehensive open-source framework for building, training, and deploying diffusion LLMs across diverse architectures. The framework integrates accelerated KV-cache techniques and inference engines for both inference and reinforcement learning, and includes implementations of various supervised fine-tuning and RL methods for mathematics, coding, and general tasks.



Model	MATH500	AIME2024	LCB-V2
TrDo-8B-Thinking	87.4	35.5	34.6
TrDo-8B-Instruct	78.5	13.3	25.9
TrDo-4B-Instruct	75.6	10.3	18.7
Qwen2.5-7B-Instruct	74.0	8.2	26.9
Llama3.1-8B-Instruct	51.9	6.7	20.0
Dream-7B-Instruct	38.7	0.0	10.7
LLaDA-8B-Instruct	38.3	1.7	5.9

Figure 1 | Left: RL training dynamics with different methods, where our TraceRL achieves the best optimization. Right: Benchmark results on complex math reasoning tasks and LiveCodeBench-V2, all evaluated with KV-cache. TraDo series models outperform strong AR models even at smaller scales.

1. Introduction

To apply diffusion models (Ho et al., 2020; Song and Ermon, 2019) on language tasks, approaches like projecting discrete tokens into a continuous latent space (Dieleman et al., 2022; Graves et al., 2023; Gulrajani and Hashimoto, 2023), and employ state transition matrices to directly derive the diffusion process (Austin et al., 2021; Ou et al., 2024; Sahoo et al., 2024; Shi et al., 2024), have been explored. Masked diffusion models (MDMs) have emerged as promising and scalable architecture for diffusion language model (DLM) recently (Nie et al., 2025; Ye et al., 2025). DLMs enable massive inference acceleration by parallel generation (Google DeepMind, 2025; Labs et al., 2025; Song et al., 2025), and improve consistency through bidirectional attention (Ye et al., 2024; Zhang et al., 2023).

Diffusion language models with full attention have been explored in coding tasks (Gong et al., 2025; Google DeepMind, 2025; Labs et al., 2025; Xie et al., 2025), fixed-format reasoning tasks (Ye et al., 2024, 2025). Beyond the full attention mechanism, block-attention diffusion models (Arriola et al., 2025) have been scaled to large language models that excel at complex reasoning tasks (Cheng et al., 2025). However, a unified and effective reinforcement learning (RL) framework across different model architectures remains underexplored.

Existing post-training frameworks for full-attention DLMs estimate the score of the whole sequence by adding random masks, which causes a mismatch with the optimal inference process (Gong et al., 2025; Nie et al., 2025; Yang et al., 2025a; Ye et al., 2025), since the sequential and logical nature of language is not purely random (Arriola et al., 2025; Gong et al., 2025). Block diffusion employs a semi-autoregressive supervised fine-tuning method (Arriola et al., 2025), while its reinforcement learning remains unexplored. In this work, we demonstrate the importance of aligning the inference trajectory with the training objective, and propose a trajectory-aware reinforcement learning method, TraceRL, along with a diffusion-based value model. Our approach can be applied to both full-attention and block-attention models, achieving fast optimization.

Solely trained with TraceRL, we obtain 4B- and 8B-sized state-of-the-art diffusion language instruction models on reasoning tasks, both surpassing strong autoregressive models on math reasoning benchmarks. Furthermore, we develop the first long-CoT diffusion language model, TraDo-8B-Thinking, through a combination of TraceRL and long-CoT SFT.

We summarize our main contributions as follows:

- We highlight the importance of aligning the training objective with the sampling trajectory. Building on this insight, we propose TraceRL, a fast and versatile reinforcement learning method applicable across diverse DLM architectures, together with a diffusion-based value model that reduces variance and improves training stability. We also explore TraceRL’s broader potential applications, including increasing the model’s block size and accelerating inference.
- Using TraceRL, we achieve significant improvements in reasoning ability across diverse tasks, resulting in the TraDo series. TraDo-4B-Instruct consistently outperforms strong AR models on various math reasoning datasets, and TraDo-8B-Instruct achieves relative accuracy gains of **6.1%** over Qwen2.5-7B-Instruct and **51.3%** over Llama3.1-8B-Instruct.
- Through curriculum learning combined with TraceRL and long-CoT SFT, we obtain the first long-CoT DLM, achieving relative accuracy gains of **18.1%** on MATH500 and **28.6%** on LiveCodeBench-V2 over Qwen2.5-7B-Instruct, demonstrating that diffusion language models can also perform complex reasoning.
- We release a complete and integrated framework for building, training, and deploying diffusion LLMs across diverse architectures. The framework incorporates implementations of various post-training methods and accelerated KV-cache techniques, enabling both reproducible research and practical applications.

2. Preliminaries

2.1. Masked Diffusion Language Models with Full Attention

Masked diffusion language models randomly replace non-masked tokens in the raw sample x_0 with mask tokens [MASK], resulting in x_t , where $t \in [0, 1]$. This process can be formulated as:

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

It has been demonstrated that training objective of DLM can be derived from data likelihood (Ou et al., 2024; Sahoo et al., 2024; Shi et al., 2024; Zheng et al., 2024). Specifically, the optimization objective is the evidence lower bound of $\log p_\theta(x)$:

$$\mathcal{J}_{full}(x_0, Q, \theta) = \int_0^1 \frac{1}{t|x_0|} \mathbb{E}_{q(x_t | x_0)} \left[\sum_{i: x_t^i = [\text{MASK}]} \log p_\theta(x_0^i | x_t, Q) \right] dt, \quad (2)$$

where Q stands for the prompt part, and $|x_0|$ stands for the number of tokens in x_0 .

2.2. Adapted From AR Models

Besides training from scratch with this objective (Nie et al., 2025; Yang et al., 2025a), adapting from an auto-regressive model has been shown to be a promising approach for preserving the AR model's capabilities (Ye et al., 2025), while only requiring adaptation training. This approach is more efficient than training from scratch and has led to a series of powerful DLMs (Gong et al., 2025; Xie et al., 2025; Ye et al., 2025). Its theoretical supervised fine-tuning objective is the same as Equation 2; the only difference is that a right-shift of the logits output is required, since its base model is an autoregressive LLM.

2.3. Block Diffusion

Block diffusion models (Arriola et al., 2025; Cheng et al., 2025) employ a block-diffusion attention mechanism that combines the training efficiency of autoregressive models with the sampling efficiency of diffusion models. Moreover, their block-wise generation makes KV-cache support naturally available, whereas full-attention DLMs experience performance degradation when using KV-cache (Arriola et al., 2025; Hu et al., 2025b; Liu et al., 2025; Ma et al., 2025; Wu et al., 2025; Yu et al., 2025b). A key limitation of block diffusion lies in its fixed block size. When the block size is small, its sampling speed potential falls short of that of full-attention DLMs, since fast sampling fundamentally depends on generating a large number of tokens at once (Song et al., 2025). This highlights the need to adapt block diffusion to larger block sizes in order to achieve sampling flexibility.

2.4. Decoding Strategy and Accelerated Inference

The reverse process (Equation 1) can be accelerated through an iterative generation procedure, in which multiple masked tokens are approximately recovered in a single step when transitioning from a noise level t to a lower level $s < t$. Specifically, given a masked sequence x_t , we select [MASK] tokens to be unmasked based on the confidence at each position i , defined as $\max_{x_s^i} p_\theta(x_s^i | x_t, Q)$. A common strategy is to unmask either a fixed number of tokens with the highest confidence at each step (Chang et al., 2022; Kim et al., 2025), referred to as **static sampling**, or all positions whose confidence exceeds a predefined threshold \mathcal{T} (Wu et al., 2025; Yu et al., 2025b), referred

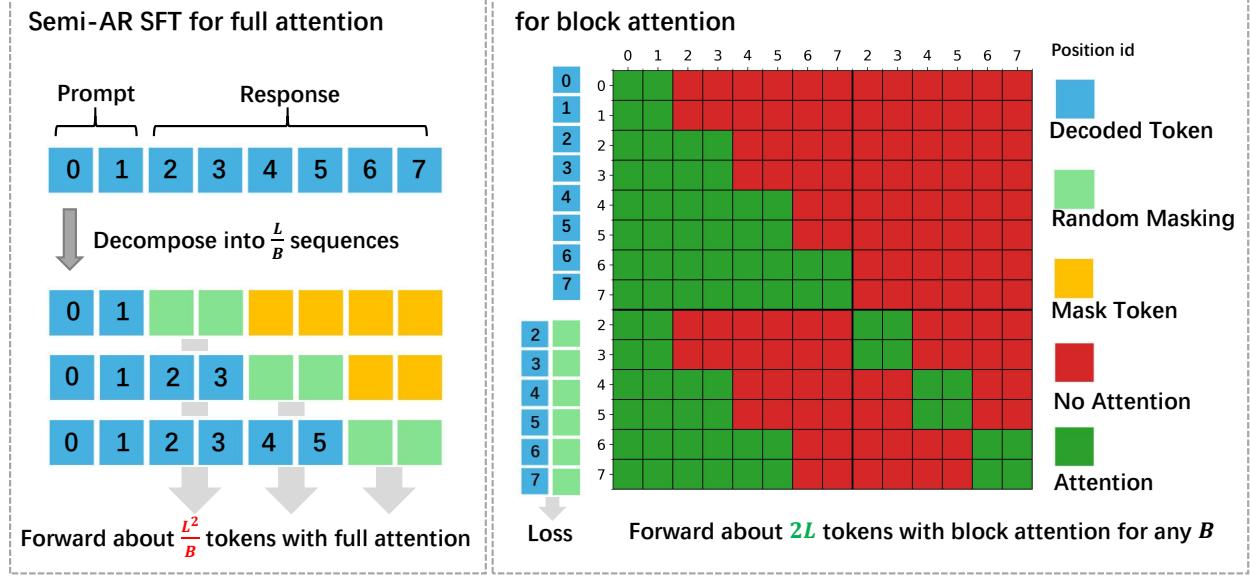


Figure 2 | Semi-AR SFT for full attention and block attention model Overview. This example uses a block size of $B = 2$ and a sequence length of $L = 6$. Block diffusion models naturally adapt to semi-AR SFT efficiently, whereas full attention models require slicing the data by B .

to as **dynamic sampling**, with the latter often being faster. By iteratively applying this procedure starting from a fully masked sequence, we eventually obtain a fully unmasked sequence. This parallel decoding strategy, combined with KV-cache techniques (Arriola et al., 2025; Hu et al., 2025b; Liu et al., 2025; Ma et al., 2025; Wu et al., 2025; Yu et al., 2025b), gives DLMs faster sampling speeds than LLMs (Google DeepMind, 2025; Labs et al., 2025; Song et al., 2025).

3. Mismatch Between Post-Training Objective and Inference Trajectory

Pretraining with the fully random masking objective (Equation 2) enables parallel decoding. However, language inherently depends on previous context. When combined with the chosen decoding strategy and the widely adopted block-wise generation with KV-cache, this creates a mismatch between the post-training objective and the model’s inference behavior. In this section, we present a simple experiment to illustrate this discrepancy.

3.1. Semi-Autoregressive Fine-Tuning

Block diffusion models naturally employ semi-autoregressive fine-tuning with the following objective, which trains the language model’s ability to generate later tokens conditioned on earlier context using block attention, while preserving the sampling efficiency characteristic of DLMs.

$$\mathcal{J}_{\text{semi}}(x, Q, \theta) = \sum_{i=1}^{\lceil L/B \rceil} \mathcal{J}_{\text{full}}(x^{(i-1)B:\min(iB,L)}, [Q, x^{0:(i-1)B}], \theta).$$

For full-attention DLMs, applying the semi-autoregressive objective inevitably increases the number of forward passes by approximately $\lceil L/B \rceil$ (see Figure 2). Nevertheless, the optimization performance of $\mathcal{J}_{\text{semi}}(x, Q, \theta)$ is substantially better than that of $\mathcal{J}_{\text{full}}(x, Q, \theta)$ under $\lceil L/B \rceil$ independent repetitions, where the computational load is the same (see Table 1). This demonstrates the importance of aligning the post-training objective with the general left-to-right inference pattern.

Table 1 | We explore how effectively different methods tune the model to learn CoT reasoning in order to improve reasoning accuracy under non-CoT prompts. The 2000 datapoints are generated by Qwen2.5 models and filtered to retain high quality. l denotes the length of each collected step in the whole trace. “ $\times m$ ” indicates that we apply m independent random maskings to augment the data scale for a fair comparison. “Token forward” denotes the number of tokens processed by the model, representing the computational load or time. “Token trained” refers to the number of tokens actually used in the optimization objective. We report accuracy on MATH500 here. The block-attention model used here is SDAR-4B-Chat (default block size of 4), and the full-attention model is Dream-7B-Instruct.

Token Utilization Efficiency	Full Attention					Block Attention		
	trace	semi-ar ($B = 16$)		fully random		trace	semi-ar ($B = 4$)	
	$l = 16$	$\times 2$	$\times 1$	$\times 35$	$\times 1$	$l = 2$	$\times 2$	$\times 1$
accuracy (%)	54.4	53.4	52.6	45.1	39.6	71.3	70.4	70.0
# token forward (M)	39.2	78.4	39.2	39.2	1.1	4.5	4.5	2.2
# token trained (M)	1.1	1.1	0.6	20.0	0.6	1.1	1.1	0.6

3.2. Aligning Post-training with Preferred Inference Traces

To further investigate, we collect the pre-trained model’s own preferred inference traces from the data and use them for finetuning. Specifically, we apply static sampling (see definition in Section 2.4) to obtain the optimal trace for each data point, which is then used in our finetuning (see Section B.1 for details).

Table 1 shows that using the model’s own preferred inference traces achieves optimal performance over the baselines, even when the computational load is equal to or lower than that of the baselines, for both block-attention and full-attention structures.

3.3. Reinforcement Learning Naturally Leverages Inference Traces

A major limitation of this fine-tuning approach is that collecting inference traces requires substantial effort. In contrast, reinforcement learning naturally produces these traces during rollouts, making it a more practical and effective strategy for post-training. Accordingly, we propose TraceRL in the following section.

4. RL Training with Trajectory

Current RL methods for DLMs focus on full-attention models, rewarding or penalizing rollout responses based on the overall generated sequence (Gong et al., 2025; Yang et al., 2025a; Zhao et al., 2025) through the random masking objective $\mathcal{J}_{\text{full}}$ (Equation 2). We propose **TraceRL** (Figure 3, detailed pipeline in section B.6), which instead focuses on the intermediate traces generated by the DLM, and can be applied across different architectures. We also introduce a diffusion-based value model, which helps reduce variance and improve stability in training. We demonstrate that this leads to a more effective optimization.

For each generated response τ_i given the task Q , we can write it in the trajectory form $\tau_i \triangleq (\tau_i(1), \dots, \tau_i(|\tau_i|))$, where $|\tau_i|$ is the total number of decoding steps, and $\tau_i(t)$ is the set of tokens decoded during the t -th step. TraceRL rewards or penalizes the generation trajectory of policy π_θ , based on the verifiable reward r_i for the response τ_i . Process-level rewards can also be incorporated

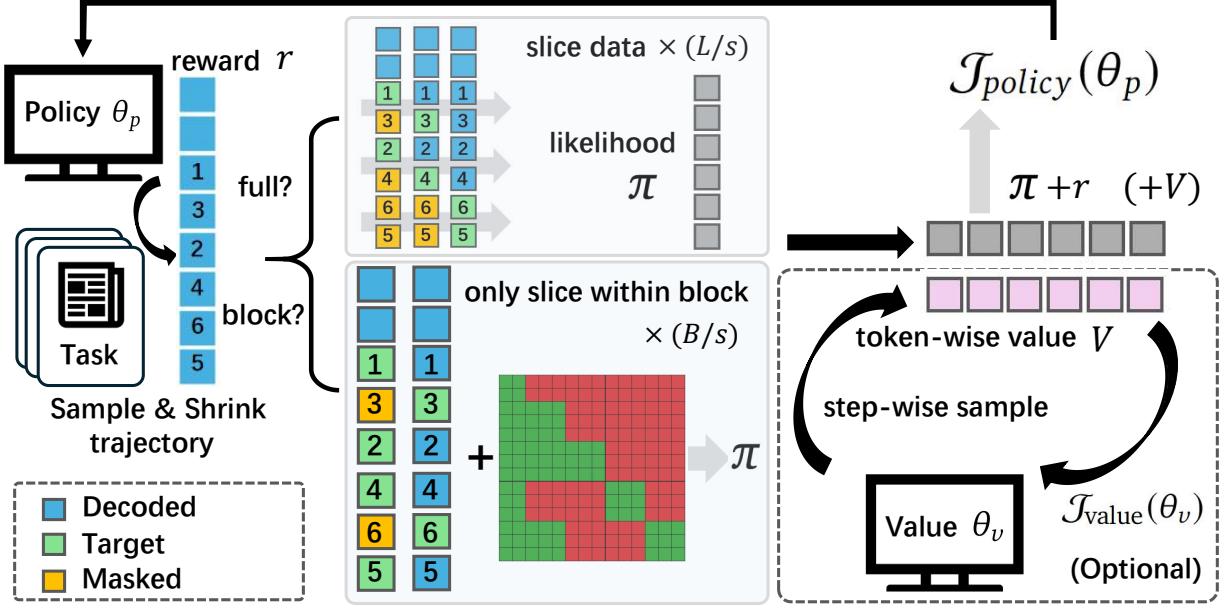


Figure 3 | TraceRL Overview (detailed pipeline in section B.6). This is an example for $s = 2$, $L = 6$ and $B = 3$. We aggregate every s neighboring steps to perform trajectory-aware reinforcement learning. The integers in these squares represent the sequence of the policy’s inference process.

through the use of a value model.

4.1. Accelerated Training with Shrinkage Parameter

For full-attention models, decomposing the sequences based on each sampling step leads to a large number of forward passes during training. So we introduce a shrinkage parameter s , which aggregates every s neighboring steps to improve training efficiency. Formally, we shrink the trajectory τ_i into $\tau_i^s \triangleq (\tau_i^s(1), \dots, \tau_i^s(|\tau_i^s|))$, where $\tau_i^s(k) \triangleq \cup_{j=s(k-1)+1}^{\min(sk, |\tau_i|)} \tau_i(j)$ and $|\tau_i^s| = \lceil |\tau_i|/s \rceil$, then optimize the following objective:

$$\mathcal{J}_{\text{policy}}(\theta_p) = \mathbb{E}_{\substack{Q \sim D_{\text{task}} \\ \{\tau_i\}_{i=1}^G \sim \pi_{\text{old}}(\cdot|Q)}} \left[\sum_{i=1}^G \sum_{t=1}^{|\tau_i^s|} \sum_{o_k \in \tau_{i,t}^s} C_\epsilon \left(\frac{\pi_{\theta_p}(o_k | \tau_i^s(1 : (t-1))), A_i)}{\pi_{\text{old}}(o_k | \tau_i^s(1 : (t-1))), A_i)} / |\tau_i^s(t)| \right] - \beta \mathbb{KL}[\pi_{\theta} || \pi_{\text{old}}] \right], \quad (3)$$

where $C_\epsilon(r, A) \triangleq \min(rA, \text{clip}(r, 1-\epsilon, 1+\epsilon)A)$, $\tau_i^s(1 : t) \triangleq \cup_{j=1}^t \tau_i^s(j)$, π_{old} is the old policy, and A_i is the standardized reward.

By introducing the shrinkage parameter s , we reduce the training computation complexity by a factor of s . We can directly use $\mathcal{J}_{\text{policy}}(\theta_p)$ for efficient RLVR (reinforcement learning with verifiable rewards) training.

4.2. Value Model with Diffusion Modeling

Rather than assigning a single sequence-level advantage to all tokens in a completion, a value function enables prefix-conditioned, token-wise advantages (e.g., GAE-based estimates), providing a variance-reducing baseline that typically stabilizes policy optimization (Hu et al., 2025a; Schulman et al., 2017). We adopt a diffusion-based value model to estimate step-wise values. For each rolled-out

trajectory τ (possibly shrunken by a shrinkage factor s), we keep the notation τ for brevity. We evaluate a frozen value network V_{old} to obtain token-wise values along the trajectory: $V_j^{\text{old}} \triangleq (V_{\text{old}}(\tau))_j$, $j \in \tau$. Concretely, for each trace step t , we condition on the prefix $\tau(1:t-1)$ and predict the values for all tokens generated at that step, i.e., $\{V_j^{\text{old}} : j \in \tau(t)\}$. We then define the step-wise value at step t as $V_t^{\star, \text{old}} = \sum_{j \in \tau(t)} V_j^{\text{old}} / |\tau(t)|$. V^{old} is treated as a *stop-gradient* baseline when constructing returns and advantages; the learnable value network V_{θ_v} is only updated via its own regression objective later.

We now derive the training objective. Given token-wise rewards r_j , we first form the step-wise reward $r_t^* = \sum_{j \in \tau(t)} r_j / |\tau(t)|$. We compute step-wise returns backward as $R_t^* = r_t^* + \gamma R_{t+1}^*$ for $t \leq |\tau|$ with terminal $R_{|\tau|+1}^* = 0$. Using the frozen baseline, define $V_t^{\star, \text{old}} = \sum_{j \in \tau(t)} V_j^{\text{old}} / |\tau(t)|$, the TD residual $\delta_t^* = r_t^* - V_t^{\star, \text{old}} + \gamma V_{t+1}^{\star, \text{old}}$, and the step-wise GAE $A_t^* = \delta_t^* + \gamma \lambda A_{t+1}^*$. We then map step-level quantities back to tokens via $R_j = r_j + \gamma R_{t_j+1}^*$ and $A_j = (r_j - V_j^{\text{old}}) + \gamma V_{t_j+1}^{\star, \text{old}} + \gamma \lambda A_{t_j+1}^*$, where t_j is such that $j \in \tau(t_j)$. The value network is trained with a clipped regression loss:

$$\mathcal{J}_{\text{value}}(\theta_v) = \frac{1}{2} \mathbb{E}_\tau \left[\frac{1}{|\tau|} \sum_{j \in \tau} \max((V_{\theta_v}(\tau)_j - R_j)^2, (V_j^{\text{clip}} - R_j)^2) \right], \quad (4)$$

where $V_j^{\text{clip}} = V_j^{\text{old}} + \text{clip}(V_{\theta_v}(\tau)_j - V_j^{\text{old}}, -\epsilon, \epsilon)$. The token advantages A_j are used in $\mathcal{J}_{\text{policy}}(\theta_p)$ to update the policy. We provide the explicit forms of R_j and A_j in Proposition 1 (see Section A in Appendix).

4.3. Sliced Training in Block Diffusion

Block diffusion employs a block-attention mechanism for efficient supervised fine-tuning, and we extend this advantage to reinforcement learning. For each derived and processed trace τ generated by block-wise inference, we represent it as $\tau = (b_1, \dots, b_{\lceil |\tau|/B \rceil})$, where each block is defined as $b_k = (\tau_{k,1}, \dots, \tau_{k,|b_k|})$ with $|b_k|$ steps. We denote the block size as B . The training objective can then be sliced from $\sum_{i=1}^{|\tau|} f(\tau(i))$ into a B' training slice, $\left(\sum_{k=1}^{\lceil |\tau|/B \rceil} f(\tau_{k,l}) \mathbf{1}_{\{l \leq |b_k|\}} \right)_{l=1}^{B'}$, where f is a task-specific function and $B' = \max_k \{|b_k|\} \leq \lceil B/s \rceil$. Each slice only needs to be forwarded once using block attention (Figure 3). This formulation maximizes the utility of the block-attention mechanism and enables highly parallel and efficient training, applicable to both policy and value model training, which is significantly more efficient than full-attention training.

5. Experiments

In this section, we demonstrate the superiority and broad applicability of TraceRL. We present evaluation results for our state-of-the-art 4B and 8B instruction models trained solely with TraceRL, as well as for the long-CoT model TraDo-8B-Thinking, trained by a combination of TraceRL and SFT. We also highlight interesting applications such as block size enlargement and the observation of acceleration effects.

5.1. Experimental Setups

5.1.1. Data

We use different data source for reinforcement learning. For the Math tasks, we choose the MATH training dataset (Hendrycks et al., 2021) and retain only level 3–5 tasks (Hu et al., 2025a), resulting in 8K hard tasks. For the coding RL setting, we use 6K verified problems from PrimeIntellect (Jaghouar et al., 2024), verified by DeepCoder (Luo et al., 2025).

Table 2 | The main benchmark results across different math and coding tasks. “Static” refers to static sampling, and “Dynamic” refers to dynamic sampling. The long-CoT model TraDo-8B-Instruct here is evaluated by dynamic sampling with threshold 0.9.

Model	MATH500	AIME2024	GSM8K	LiveCodeBench-v2	LiveBench					
Autoregressive Models										
Llama3.1-8B-Instruct	51.9	6.7	84.5	20.0	19.7					
Qwen2.5-7B-Instruct	74.0	8.2	89.9	26.9	31.1					
Diffusion Language Models										
LLaDA-8B-Instruct	Static 37.3	Dynamic 38.3	Static 0.5	Dynamic 1.7	Static 82.5	Dynamic 82.5	Static 5.9	Dynamic 5.5	Static 4.9	Dynamic 6.0
Dream-7B-Instruct	38.7	32.3	/	/	72.7	57.8	10.7	4.7	10.7	4.9
SDAR-4B-Chat	70.2	67.4	5.0	8.2	90.2	88.9	15.6	11.2	14.0	7.6
TraDo-4B-Instruct	75.6 <small>+5.4</small>	71.8 <small>+4.4</small>	8.3 <small>+3.3</small>	10.3 <small>+2.1</small>	91.2 <small>+1.0</small>	90.3 <small>+1.2</small>	18.7 <small>+3.1</small>	15.1 <small>+3.9</small>	12.9	10.4 <small>+2.8</small>
SDAR-8B-Chat	74.3	70.7	11.8	8.3	91.1	90.4	18.5	15.3	11.5	11.2
TraDo-8B-Instruct	78.5 <small>+4.2</small>	75.5 <small>+4.8</small>	13.3 <small>+1.5</small>	11.0 <small>+2.7</small>	92.3 <small>+1.2</small>	91.2 <small>+0.8</small>	25.9 <small>+7.4</small>	22.4 <small>+7.1</small>	22.7 <small>+11.2</small>	20.6 <small>+9.4</small>
TraDo-8B-Thinking	87.4 <small>+13.1</small>		35.5 <small>+23.7</small>		94.2 <small>+3.1</small>		34.6 <small>+16.1</small>		36.0 <small>+23.8</small>	

Our evaluation focuses on reasoning tasks in mathematics and coding. For mathematics, we use GSM8K (Cobbe et al., 2021), MATH500 (Hendrycks et al., 2021), and AIME (Mathematical Association of America, American Mathematics Competitions, 2024). For coding, we use LiveCodeBench-V2 (Jain et al., 2024) and LiveBench (White et al., 2024).

5.1.2. Models and Optimization

Our experiments include both full-attention and block-attention models. For full attention, we use Dream-7B-Instruct (Ye et al., 2025) as the base model, and for block attention, we use the SDAR (Cheng et al., 2025) series of models, trained with a block size of 4.

We now describe our reinforcement learning settings. For the block-diffusion model, during each sampling step we sample 128 tasks and 32 responses per task, with a dynamic sampling strategy ($\mathcal{T} = 0.9$) and temperature 1.0. For the full-attention model, during each RL sampling step we sample 56 tasks and generate 8 responses per task from the policy π_θ using KV-Cache, with a temperature of 1.0 and a static sampling strategy (one token per step) to improve rollout quality. During training, we set the learning rate of the policy to 1×10^{-6} , with $\epsilon = 0.2$ and $\beta = 0.01$. When using the value model, we set the learning rate for RL to 5×10^{-6} , and use $\gamma = \lambda = 1.0$ as default values. We provide additional optimization details in the appendix (Sections B.4 and B.5).

5.1.3. Evaluation

During evaluation, we report results for both static and dynamic sampling. For the block-attention model, we use temperature $t = 1.0$ for dynamic sampling and greedy decoding (top- $k = 1$ for static sampling), with a default block size of 4. For the full-attention model, we use temperature $t = 0.1$ with block size 4 for static sampling and block size 32 for dynamic sampling. See Appendix (Section B.3 for evaluation details and Section B.2 for prompt templates).

5.2. Instruction Models Trained with TraceRL

We obtain TraDo-4B-Instruct and TraDo-8B-Instruct by applying TraceRL to math and coding tasks, starting from the SDAR base model. We evaluate our models across math and coding tasks, as well as five reasoning datasets, and compare them against both strong diffusion language models and autoregressive models (Dubey et al., 2024; Yang et al., 2024a). Our instruction models achieve

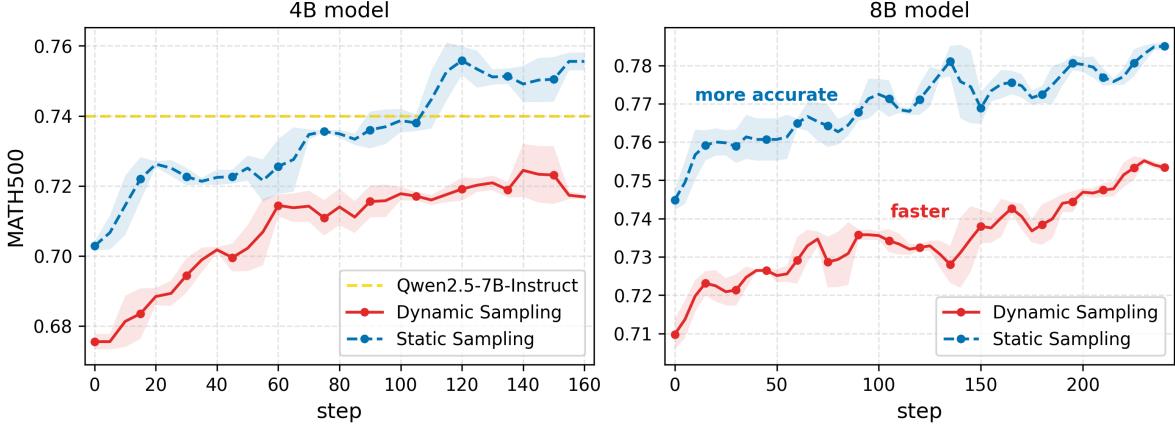


Figure 4 | The TraceRL training curves for the 4B and 8B models on the math task. The red curve denotes the dynamic sampling accuracy, which achieves faster sampling speed, while the blue curve denotes the static sampling accuracy, which achieves higher accuracy. The 4B model is trained with a value model, whereas the 8B model is trained directly using $\mathcal{J}_{\text{policy}}$.

state-of-the-art performance on reasoning tasks among current diffusion models, demonstrating the effectiveness of TraceRL. Both dynamic (faster) and static (more accurate) sampling abilities have been improved significantly. Notably, TraDo-4B-Instruct outperforms strong autoregressive baselines such as Qwen2.5-7B-Instruct across all math tasks.

5.3. Long-CoT Diffusion Language Model

The long-CoT diffusion language model TraDo-8B-Thinking is derived from TraDo-8B-Instruct, trained through a combination of long-CoT SFT and TraceRL. As the first long-CoT diffusion language model, TraDo-8B-Thinking demonstrates strong reasoning capabilities across benchmarks, notably achieving 85.8% accuracy on MATH500, showing that diffusion language models can also excel at complex reasoning tasks.

5.4. Training Dynamics of TraceRL

We record the training dynamics on math tasks for our 4B and 8B instruction models (see Figure 4). Although we adopt dynamic sampling during the RL training process, both dynamic and static accuracy improve steadily, and the trend suggests further potential for scaling. This RL training significantly enhances the models’ math reasoning ability: on MATH500, TraDo-4B-Instruct improves by 5.4% (static) and 4.2% (dynamic), surpassing Qwen2.5-7B-Instruct after optimization, while TraDo-8B-Instruct improves by 4.2% (static) and 4.8% (dynamic) (see Figure 4).

5.5. Stronger Optimization with TraceRL

We compare TraceRL with existing RL methods, focusing first on block diffusion models. Although current RL methods are primarily developed for full attention models, we adapt them directly to the block setting. For the random masking approach (Yang et al., 2025a), we restrict sampling within each block, making it resemble a semi-autoregressive method. For coupled RL (Gong et al., 2025), we introduce a complementary objective within each block, which provides more stable and effective training. We evaluate these methods on math tasks, with the results shown in Figure 5. The experiments demonstrate that TraceRL achieves the best optimization performance, both with and

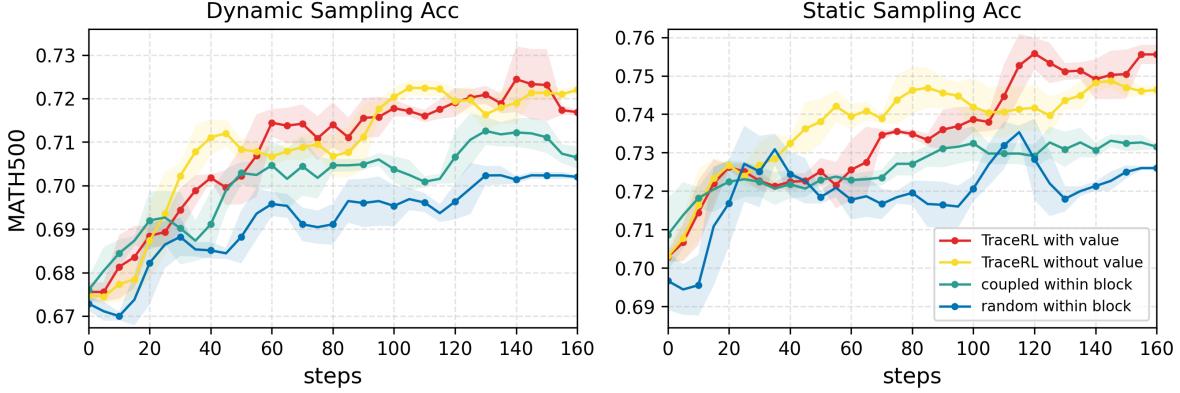


Figure 5 | RL method ablations on block diffusion models and math RL tasks. The red and yellow curves represent TraceRL with and without a value model, respectively. The blue curve corresponds to training with a random masking objective within block, similar to the semi-autoregressive training approach. The green curve represents training with an additional complementary mask within block.

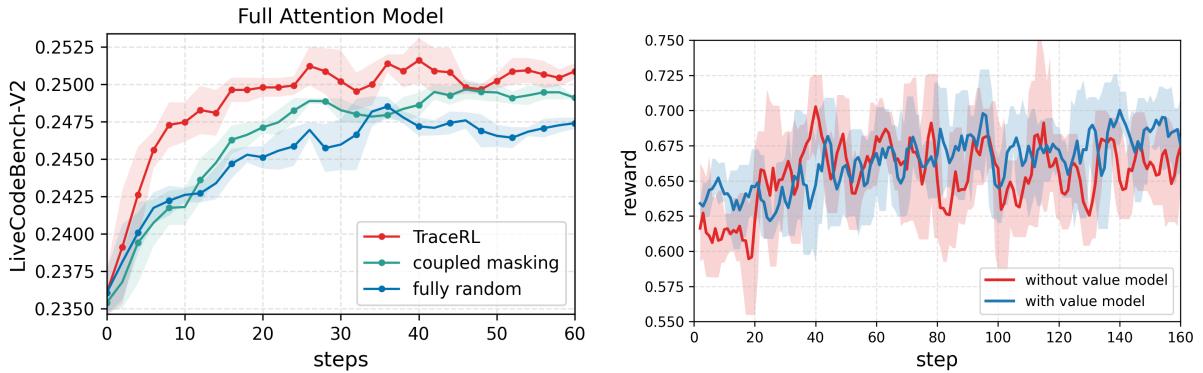


Figure 6 | (a) RL training ablations for the full attention model Dream-7B-Coder-Instruct, focusing on coding tasks. (b) The comparison between using and not using a value model shows that incorporating a value model leads to fewer fluctuations during training. The experiments are all conducted on the 4B model with math tasks.

without a value model. All RL methods use dynamic sampling during rollout, and TraceRL attains superior optimization under both dynamic and static evaluation settings. This demonstrates the importance of optimizing over the preferred trace, even within a small block.

5.6. Also Fit for Full Attention Models and Coding Tasks

To demonstrate the broad applicability of our method, we also experiment with a full attention model on coding RL tasks. We use Dream-7B-Coder-Instruct as the base model, fine-tuned on distillation data as a cold start before performing RL training (see details in Section B.5). To accelerate training, we set the shrinkage parameter to $s = 8$. Similar to Table 1, we augment the training data for the two baseline methods (Section B.5). We find that TraceRL converges faster and achieves the best performance compared with the baselines (see Figure 6 (a)). Notably, after training, we achieve 25.0% accuracy on LiveCodeBench-V2, setting a new state of the art among open-source full-attention DLMs.

Table 3 | Adapting block size $B = 4$ to 8 on reasoning tasks through TraceRL. The reported values are accuracies of these baselines under dynamic sampling with threshold 0.9.

Model	inference block size	MATH500	GSM8K	LiveCodeBench
Base model with $B = 4$	4	67.4	88.9	11.2
	8	60.2	83.0	9.8
Enlarge B to 8 by TraceRL	8	67.7	88.7	10.8

Table 4 | Acceleration ratio and Response Length Analysis. “Acceleration” is defined as the ratio of response length to the total sampling steps for a task, and we report the average ratio across the tested dataset. We use dynamic sampling with threshold 0.9 here.

Model	MATH500			LiveBench		
	accelerated	total step	avg len.	accelerated	total step	avg len.
SDAR-4B-Chat	2.28	240	548 tok.	1.56	119	181 tok.
TraDo-4B-Instruct	2.63	229	595 tok.	1.61	154	238 tok.
SDAR-8B-Chat	2.42	228	557 tok.	1.56	161	256 tok.
TraDo-8B-Instruct	2.50	240	625 tok.	1.60	151	233 tok.

5.7. Diffusion Value Model for Reducing Training Variance

Training a policy model coupled with a value model can provide a variance-reducing baseline in LLM RL training (Hu et al., 2025a; Schulman et al., 2017). We extend this idea to diffusion language models. In our diffusion-based value model, we assign the verifiable reward of the entire trajectory to the tokens in the final step, while leaving the reward for all preceding tokens as zero. We then use our proposed modeling to estimate values and advantages, jointly optimizing the policy and value models. This approach leads to reduced variance and fluctuations in the training process (see Figure 6(b)) for the 4B model on math tasks.

5.8. Scaling Block Size with TraceRL

A block size of 4 can limit inference flexibility, and some inference acceleration methods depend on larger block sizes (Hong et al., 2025; Song et al., 2025; Wang et al., 2025a). Therefore, we explore the potential of using TraceRL to increase the block size. We first perform rollout with a block size of $B = 4$, then apply TraceRL with $B = 8$. After 60 steps, we switch the rollout to $B = 8$ and continue optimizing with $B = 8$ for 40 steps, resulting in a model that is familiar with and better adapted to larger block sizes. As shown in Table 3, this approach proves effective across different tasks, even though it is trained only on math tasks.

5.9. Analysis on Acceleration Ratio and Response Length

From Table 4, we observe that TraceRL optimization accelerates dynamic sampling, achieving a 15.4% speedup on MATH500. A straightforward explanation is that the model becomes more confident when encountering problems in the domain it has been optimized for, which in turn allows each step of dynamic sampling more likely to unmask more tokens under the same threshold. We also find that the average response length on complex math reasoning tasks increases, providing further evidence of improved reasoning ability.

6. Open-source Framework

We implement an open-source framework for training and deploying large diffusion language models, released at <https://github.com/Gen-Verse/dLLM-RL>. Its main features are summarized as follows.

6.1. Support for Different Model Structures

Diffusion language models exhibit diverse architectures (see Section 2). Our framework is comprehensive and supports major classes of large diffusion models, including pretrained full-attention models (e.g., the LLaDA series (Nie et al., 2025; Zhu et al., 2025) and MMaDA (Yang et al., 2025a)), adapted full-attention models (e.g., the Dream series (Xie et al., 2025; Ye et al., 2025) and DiffuCoder (Gong et al., 2025)), and block diffusion models (e.g., SDAR (Cheng et al., 2025) and our TraDo series).

6.2. Accelerated Inference

To improve sampling efficiency while avoiding performance degradation, multiple KV-cache techniques have been proposed for full-attention diffusion language models (Arriola et al., 2025; Hu et al., 2025b; Liu et al., 2025; Ma et al., 2025; Wu et al., 2025; Yu et al., 2025b). We extend Fast-DLLM (Wu et al., 2025) by adding a tunable window-size further horizon for each forward pass (see Section B.3). For block diffusion models, we employ JetEngine (Cheng et al., 2025) to accelerate inference. These acceleration techniques are applied in both inference and RL sampling steps.

6.3. Diverse Post-training Method Implementations

For supervised fine-tuning, we support fully random masking and semi-AR methods for full-attention models, as well as efficient semi-AR fine-tuning for block-attention models. Our framework includes TraceRL, coupled RL (Gong et al., 2025), and random masking RL (Yang et al., 2025a), with accelerated inference steps. For block diffusion models, we also provide the option to use a diffusion-based value model. All training methods support multi-node training.

7. Related Work

7.1. Enhancing Reasoning Capabilities of Large Language Models

Approaches have been proposed to boost the reasoning abilities of large language models (LLMs), spanning Chain-of-Thought (CoT) prompting (Wei et al., 2022; Yang et al., 2024c)—which elicits step-by-step rationales—and self-improvement optimization algorithms (Hosseini et al., 2024; Yang et al., 2024d; Zelikman et al., 2024), in which the model iteratively refines its own answers through verification, reflection, or voting. Incorporating Long-CoT methods, such as self-checking and self-reflection patterns, significantly enhances the reasoning capabilities of LLMs. Reinforcement learning (RL) has been found to be able to incentivize the reasoning ability of LLMs (Guo et al., 2025; Hu et al., 2025a; Hugging Face, 2025; Team et al., 2025; Yang et al., 2025b) and to improve their ability to solve complex tasks (Jiang et al., 2025; Jin et al., 2025; Wang et al., 2025b; Yang et al., 2024b; Zou et al., 2025).

Despite the notable reasoning power of Long-CoT models, their inference speed remains prohibitively slow, highlighting the necessity and potential for faster inference methods (Sui et al., 2025). Diffusion language models, with their parallel decoding capabilities and bidirectional mechanisms, present a promising direction to overcome these limitations and achieve more efficient reasoning inference.

7.2. RL for Diffusion Language Model

Reinforcement learning methods for full-attention models have been studied in prior work (Gong et al., 2025; Yang et al., 2025a; Zhao et al., 2025). In MMAADA, Yang et al. (2025a) introduce random masking for each rollout sample and optimize a PPO objective to perform RL training. Gong et al. (2025) propose augmenting each rollout x_0 with not only a single noise sample x_t but also a complementary counterpart \hat{x}_t , thereby doubling the effective training data. They provide a theoretical analysis showing that this strategy reduces variance, and empirical results confirm faster and more stable optimization compared to random-masking RL. Nevertheless, these approaches overlook information contained in the sampling trajectory.

7.3. Diffusion Models for Language Modeling

Extending the iterative-refinement paradigm of diffusion language models (Ho et al., 2020; Song and Ermon, 2019) to discrete language data is not straightforward. One line of research projects tokens into a continuous latent space where standard Gaussian diffusion can operate (Dieleman et al., 2022; Graves et al., 2023; Gulrajani and Hashimoto, 2023). A parallel thread directly defines the forward noising process on the token simplex via state-transition matrices. Structured Diffusion LM (Austin et al., 2021) introduced a categorical forward kernel (e.g. uniform corruption or bit-flip), and trained a transformer to invert it. Subsequent work simplified or re-parameterised the transition matrix to reduce variance and improve stability (Ou et al., 2024; Sahoo et al., 2024; Shi et al., 2024).

Masked Diffusion Models (MDMs) have emerged as a scalable architecture for large-scale diffusion language models (Nie et al., 2025; Yang et al., 2025a; Ye et al., 2025). MDMs naturally leverage bidirectional attention, yielding stronger global consistency than left-to-right autoregressive transformers (Ye et al., 2024; Zhang et al., 2023), while also supporting parallel decoding, which enables substantial inference acceleration (Google DeepMind, 2025; Labs et al., 2025).

8. Conclusion

We present a new reinforcement learning method for diffusion language models with diverse architectures. Through extensive experiments and evaluations, we demonstrate the effectiveness of this method across different RL tasks, resulting in three state-of-the-art diffusion language models. We also highlight its benefits for accelerating inference and scaling block size, which suggest promising directions for further exploration.

In particular, integrating the accelerated inference strengths of diffusion models with their potential for strong reasoning ability represents an exciting research avenue. While current long-CoT LLMs achieve strong performance on complex tasks, they suffer from extremely long reasoning times. Such integration could unlock new opportunities for deploying complex reasoning tasks efficiently at scale. The proposed diffusion value model can incorporate process rewards, providing stronger supervision than a single verifiable reward (Lightman et al., 2023). Further exploration of process reward for TraceRL optimization remains an important direction for future work.

To accelerate progress in this area, facilitate reproducible research, and support practical applications, we release a comprehensive framework for building, training, and deploying diffusion language models with diverse architectures. This framework integrates accelerated inference techniques and incorporates them into its reinforcement learning pipelines. It supports implementations of various reinforcement learning and supervised fine-tuning methods.

References

- M. Arriola, A. Gokaslan, J. T. Chiu, Z. Yang, Z. Qi, J. Han, S. S. Sahoo, and V. Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.
- J. Austin, D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.
- H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11315–11325, 2022.
- S. Cheng, Y. Bian, D. Liu, Y. Jiang, Y. Liu, L. Zhang, W. Wang, Q. Guo, K. Chen, B. Qi*, and B. Zhou. Sdar: A synergistic diffusion–autoregression paradigm for scalable sequence generation, 2025. URL <https://github.com/JetAstra/SDAR>.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- S. Dieleman, L. Sartran, A. Roshannai, N. Savinov, Y. Ganin, P. H. Richemond, A. Doucet, R. Strudel, C. Dyer, C. Durkan, et al. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.
- A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024.
- S. Gong, R. Zhang, H. Zheng, J. Gu, N. Jaitly, L. Kong, and Y. Zhang. Diffucoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv:2506.20639*, 2025.
- Google DeepMind. Gemini diffusion. <https://blog.google/technology/google-deepmind/gemini-diffusion/>, 2025. Accessed: 2024-07-24.
- A. Graves, R. K. Srivastava, T. Atkinson, and F. Gomez. Bayesian flow networks. *arXiv preprint arXiv:2308.07037*, 2023.
- I. Gulrajani and T. B. Hashimoto. Likelihood-based diffusion language models. *Advances in Neural Information Processing Systems*, 36:16693–16715, 2023.
- D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- F. Hong, G. Yu, Y. Ye, H. Huang, H. Zheng, Y. Zhang, Y. Wang, and J. Yao. Wide-in, narrow-out: Revokable decoding for efficient and effective dllms. *arXiv preprint arXiv:2507.18578*, 2025.
- A. Hosseini, X. Yuan, N. Malkin, A. Courville, A. Sordoni, and R. Agarwal. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*, 2024.

- J. Hu, Y. Zhang, Q. Han, D. Jiang, X. Zhang, and H.-Y. Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025a.
- Z. Hu, J. Meng, Y. Akhauri, M. S. Abdelfattah, J.-s. Seo, Z. Zhang, and U. Gupta. Accelerating diffusion language model inference via efficient kv caching and guided diffusion. *arXiv preprint arXiv:2505.21467*, 2025b.
- Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.
- S. Jaghouar, J. M. Ong, M. Basra, F. Obeid, J. Straube, M. Keiblinger, E. Bakouch, L. Atkins, M. Panahi, C. Goddard, et al. Intellect-1 technical report. *arXiv preprint arXiv:2412.01152*, 2024.
- N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.
- P. Jiang, J. Lin, L. Cao, R. Tian, S. Kang, Z. Wang, J. Sun, and J. Han. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. *arXiv preprint arXiv:2503.00223*, 2025.
- B. Jin, H. Zeng, Z. Yue, J. Yoon, S. Arik, D. Wang, H. Zamani, and J. Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.
- J. Kim, K. Shah, V. Kontonis, S. Kakade, and S. Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. *arXiv preprint arXiv:2502.06768*, 2025.
- I. Labs, S. Khanna, S. Kharbanda, S. Li, H. Varma, E. Wang, S. Birnbaum, Z. Luo, Y. Miraoui, A. Palrecha, et al. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv:2506.17298*, 2025.
- Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittweiser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. Competition-level code generation with alphacode. *Science*, 378 (6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL <https://www.science.org/doi/10.1126/science.abq1158>.
- H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.
- Z. Liu, Y. Yang, Y. Zhang, J. Chen, C. Zou, Q. Wei, S. Wang, and L. Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. *arXiv preprint arXiv:2506.06295*, 2025.
- M. Luo, S. Tan, R. Huang, A. Patel, A. Ariyak, Q. Wu, X. Shi, R. Xin, C. Cai, M. Weber, C. Zhang, L. E. Li, R. A. Popa, and I. Stoica. Deepcoder: A fully open-source 14b coder at o3-mini level. <https://pretty-radio-b75.notion.site/DeepCoder-A-Fully-Open-Source-14B-Coder-at-O3-Mini-Level-1cf81902c14680b3bee5eb349a512a51>, 2025. Notion Blog.
- X. Ma, R. Yu, G. Fang, and X. Wang. dkv-cache: The cache for diffusion language models. *arXiv preprint arXiv:2505.15781*, 2025.

Mathematical Association of America, American Mathematics Competitions. American invitational mathematics examination (aime) 2024: Aime i and aime ii. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions, 2024. Competition problems used as an evaluation dataset; original problems by MAA AMC.

- S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.
- J. Ou, S. Nie, K. Xue, F. Zhu, J. Sun, Z. Li, and C. Li. Your absorbing discrete diffusion secretly models the conditional distributions of clean data. *arXiv preprint arXiv:2406.03736*, 2024.
- S. Sahoo, M. Arriola, Y. Schiff, A. Gokaslan, E. Marroquin, J. Chiu, A. Rush, and V. Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.
- J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- J. Shi, K. Han, Z. Wang, A. Doucet, and M. Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.
- Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.
- Y. Song, Z. Zhang, C. Luo, P. Gao, F. Xia, H. Luo, Z. Li, Y. Yang, H. Yu, X. Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.
- Y. Sui, Y.-N. Chuang, G. Wang, J. Zhang, T. Zhang, J. Yuan, H. Liu, A. Wen, S. Zhong, H. Chen, et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.
- K. Team, A. Du, B. Gao, B. Xing, C. Jiang, C. Chen, C. Li, C. Xiao, C. Du, C. Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.
- X. Wang, C. Xu, Y. Jin, J. Jin, H. Zhang, and Z. Deng. Diffusion llms can do faster-than-ar inference via discrete diffusion forcing. *arXiv preprint arXiv:2508.09192*, 2025a.
- Y. Wang, L. Yang, Y. Tian, K. Shen, and M. Wang. Co-evolving llm coder and unit tester via reinforcement learning. *arXiv preprint arXiv:2506.03136*, 2025b.
- J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- C. White, S. Dooley, M. Roberts, A. Pal, B. Feuer, S. Jain, R. Shwartz-Ziv, N. Jain, K. Saifullah, S. Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.
- C. Wu, H. Zhang, S. Xue, Z. Liu, S. Diao, L. Zhu, P. Luo, S. Han, and E. Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. *arXiv preprint arXiv:2505.22618*, 2025.
- Z. Xie, J. Ye, L. Zheng, J. Gao, J. Dong, Z. Wu, X. Zhao, S. Gong, X. Jiang, Z. Li, and L. Kong. Dream-coder 7b, 2025. URL <https://hkunlp.github.io/blog/2025/dream-coder>.

- A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.
- C. Yang, H. J. Kang, J. Shi, and D. Lo. Acecode: A reinforcement learning framework for aligning code efficiency and correctness in code language models. *arXiv preprint arXiv:2412.17264*, 2024b.
- L. Yang, Z. Yu, T. Zhang, S. Cao, M. Xu, W. Zhang, J. E. Gonzalez, and B. Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024c.
- L. Yang, Z. Yu, T. Zhang, M. Xu, J. E. Gonzalez, B. Cui, and S. Yan. Supercorrect: Advancing small llm reasoning with thought template distillation and self-correction. *arXiv preprint arXiv:2410.09008*, 2024d.
- L. Yang, Y. Tian, B. Li, X. Zhang, K. Shen, Y. Tong, and M. Wang. Mmada: Multimodal large diffusion language models. *arXiv preprint arXiv:2505.15809*, 2025a.
- L. Yang, Z. Yu, B. Cui, and M. Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*, 2025b.
- J. Ye, J. Gao, S. Gong, L. Zheng, X. Jiang, Z. Li, and L. Kong. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint arXiv:2410.14157*, 2024.
- J. Ye, Z. Xie, L. Zheng, J. Gao, Z. Wu, X. Jiang, Z. Li, and L. Kong. Dream 7b, 2025. URL <https://hkunlp.github.io/blog/2025/dream>.
- Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025a.
- R. Yu, X. Ma, and X. Wang. Dimple: Discrete diffusion multimodal large language model with parallel decoding. *arXiv preprint arXiv:2505.16990*, 2025b.
- E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman. Star: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proc. the 36th International Conference on Neural Information Processing Systems*, volume 1126, 2024.
- Y. Zhang, J. Gu, Z. Wu, S. Zhai, J. Susskind, and N. Jaitly. Planner: Generating diversified paragraph via latent language diffusion model. *Advances in Neural Information Processing Systems*, 36:80178–80190, 2023.
- S. Zhao, D. Gupta, Q. Zheng, and A. Grover. d1: Scaling reasoning in diffusion large language models via reinforcement learning. *arXiv preprint arXiv:2504.12216*, 2025.
- K. Zheng, Y. Chen, H. Mao, M.-Y. Liu, J. Zhu, and Q. Zhang. Masked diffusion models are secretly time-agnostic masked models and exploit inaccurate categorical sampling. *arXiv preprint arXiv:2409.02908*, 2024.
- F. Zhu, R. Wang, S. Nie, X. Zhang, C. Wu, J. Hu, J. Zhou, J. Chen, Y. Lin, J.-R. Wen, et al. Llada 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv:2505.19223*, 2025.
- J. Zou, L. Yang, J. Gu, J. Qiu, K. Shen, J. He, and M. Wang. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2506.18896*, 2025.

A. Theoretical Results

Proposition 1 (Token-wise return and advantage from step-wise recursions). *Let a trajectory τ be partitioned into trace steps $\tau(1), \dots, \tau(|\tau|)$, and let t_j denote the unique step index with $j \in \tau(t_j)$. For token-wise rewards r_j and token-wise values V_j^{old} , define the step-wise aggregates*

$$r_t^\star := \frac{1}{|\tau(t)|} \sum_{l \in \tau(t)} r_l, \quad V_t^{\star, \text{old}} := \frac{1}{|\tau(t)|} \sum_{l \in \tau(t)} V_l^{\text{old}}.$$

Let the step-wise return and GAE be given by

$$\begin{aligned} R_t^\star &= r_t^\star + \gamma R_{t+1}^\star, \quad R_{|\tau|+1}^\star = 0, \quad \delta_t^\star = r_t^\star - V_t^{\star, \text{old}} + \gamma V_{t+1}^{\star, \text{old}}, \\ A_t^\star &= \sum_{k=0}^{|\tau|-t} (\gamma \lambda)^k \delta_{t+k}^\star, \quad A_{|\tau|+1}^\star = 0, \quad V_{|\tau|+1}^{\star, \text{old}} = 0. \end{aligned}$$

Define token-wise quantities

$$R_j := r_j + \gamma R_{t_j+1}^\star, \quad A_j := r_j - V_j^{\text{old}} + \gamma V_{t_j+1}^{\star, \text{old}} + \gamma \lambda A_{t_j+1}^\star.$$

Then the following explicit expressions hold:

$$R_j = r_j + \sum_{k=1}^{|\tau|-t_j} \gamma^k \frac{1}{|\tau(t_j+k)|} \sum_{l \in \tau(t_j+k)} r_l, \tag{1}$$

$$A_j = r_j - V_j^{\text{old}} + \sum_{k=1}^{|\tau|-t_j} (\gamma \lambda)^k \frac{1}{|\tau(t_j+k)|} \sum_{l \in \tau(t_j+k)} \left(r_l + \frac{1-\lambda}{\lambda} V_l^{\text{old}} \right). \tag{2}$$

For the boundary case $\lambda = 0$, one has directly $A_j = r_j - V_j^{\text{old}} + \gamma V_{t_j+1}^{\star, \text{old}}$.

Proof. Return. Unroll the recursion $R_{t_j+1}^\star = \sum_{k=0}^{|\tau|-t_j-1} \gamma^k r_{t_j+1+k}^\star$ and substitute into $R_j = r_j + \gamma R_{t_j+1}^\star$ to obtain $R_j = r_j + \sum_{k=1}^{|\tau|-t_j} \gamma^k r_{t_j+k}^\star$. Using $r_t^\star = \frac{1}{|\tau(t)|} \sum_{l \in \tau(t)} r_l$ gives (1).

Advantage. Write $A_{t_j+1}^\star = \sum_{k=0}^{|\tau|-t_j-1} (\gamma \lambda)^k \delta_{t_j+1+k}^\star$ and substitute $\delta_t^\star = r_t^\star - V_t^{\star, \text{old}} + \gamma V_{t+1}^{\star, \text{old}}$. Reindex to start at $k = 1$:

$$A_j = (r_j - V_j^{\text{old}}) + \sum_{k=1}^{|\tau|-t_j} (\gamma \lambda)^k r_{t_j+k}^\star + \gamma(1-\lambda) \sum_{k=1}^{|\tau|-t_j} (\gamma \lambda)^{k-1} V_{t_j+k}^{\star, \text{old}}.$$

The last line follows from collecting the $V^{\star, \text{old}}$ terms into a telescoping series whose coefficient is $\gamma(1-\lambda)(\gamma \lambda)^{k-1}$ for step t_j+k (including $k=1$). Now use $r_t^\star = \frac{1}{|\tau(t)|} \sum_{l \in \tau(t)} r_l$ and $V_t^{\star, \text{old}} = \frac{1}{|\tau(t)|} \sum_{l \in \tau(t)} V_l^{\text{old}}$, and note that $\gamma(1-\lambda)(\gamma \lambda)^{k-1} = (\gamma \lambda)^k \cdot \frac{1-\lambda}{\lambda}$ for $\lambda > 0$. This yields (2). For $\lambda = 0$, the series vanish and the definition gives the stated one-step TD form. \square

Remark 1 (Special cases). (i) $(\gamma, \lambda) = (1, 1)$. One has

$$R_j = r_j + \sum_{k=1}^{|\tau|-t_j} \frac{1}{|\tau(t_j+k)|} \sum_{l \in \tau(t_j+k)} r_l, \quad A_j = R_j - V_j^{\text{old}},$$

i.e., undiscounted Monte Carlo return and advantage with a token-wise baseline.

(ii) $(\gamma, \lambda) = (1, 0)$. The return remains the undiscounted form with $\gamma = 1$ as above, and

$$A_j = r_j - V_j^{old} + V_{t_j+1}^{\star, old},$$

which is the one-step TD(0) advantage using the trace-level baseline $V_{t_j+1}^{\star, old}$.

B. Experimental Details

B.1. Supervised Finetuning Methods Explorations

In the demonstration of Section 3, the training data consist of CoT responses generated by the Qwen2.5-32B-Instruct model (Yang et al., 2024a) on 2,000 randomly selected tasks from the OpenR1-MATH training set (Hugging Face, 2025), filtered to exclude problems solvable by the Qwen2.5-7B-Instruct model. We then collect the traces of these data using the model under evaluation. For each instance, the model is used to iteratively select the two tokens with the highest confidence, conditioned on all previously processed tokens. This procedure produces a trace, which is then aggregated by grouping every $l/2$ neighboring tokens to obtain the final trace, where each step has length l . We train on the collected data for one epoch using 64 A100 GPUs, with a learning rate of 1×10^{-6} for Dream and 1×10^{-5} for SDAR.

B.2. Prompt Templates

Math Prompt Templates	
Dream	<pre>'''< im_start >system\nYou are a helpful assistant.< im_end >\n< im_start >user\nYou ↪ need to put your final answer in \\boxed{ }. This is the problem:\n{{problem ↪ }}< im_end >\n< im_start >assistant\n'''</pre>
LLaDA	<pre>"""< startoftext >< start_header_id >user< end_header_id >You need to put your final ↪ answer in \\boxed{ }. This is the problem:\n{{problem}}< eot_id >< startoftext ↪ >< start_header_id >assistant< end_header_id >\n"""</pre>
TraDo (non-thinking)	<pre>'''< im_start >user\n{{problem}}\nPlease reason step by step, and put your final ↪ answer within \\boxed{ }.< im_end >\n< im_start >assistant\n'''</pre>
TraDo (thinking)	<pre>'''< im_start >user\nYou need to put your final answer in \\boxed{ }. This is the ↪ problem:\n{{problem}}< im_end >\n< im_start >assistant<think>\n'''</pre>
TraDo (non-CoT)	<pre>'''< im_start >user\nYou need to put your final answer in \\boxed{ }. This is the ↪ problem:\n{{problem}}< im_end >\n< im_start >assistant\n'''</pre>

Code Prompt Templates

Dream

```
'''<|im_start|>system\nYou are a helpful assistant.<|im_end|>\n<|im_start|>user\
→ nThis is the problem:\n{{problem}}\nYou should put your code in '''python'''.
→ Use input() to read input and print() to produce output in your script. <|
→ im_end|>\n<|im_start|>assistant\n'''
```

LLaDA

```
'''<|startoftext|><|start_header_id|>user<|end_header_id|>This is the problem:\n{{
→ problem}}\n You should put your code in '''python''' . Use input() to read
→ input and print() to produce output in your script. <|eot_id|><|startoftext
→ |><|start_header_id|>assistant<|end_header_id|>\n'''
```

TraDo (non-thinking)

```
'''<|im_start|>user\nThis is the problem:\n{{problem}}\nYou should put your code in
→ '''python''' . Use input() to read input and print() to produce output in your
→ script. <|im_end|>\n<|im_start|>assistant\n'''
```

TraDo (thinking)

```
'''<|im_start|>user\nThis is the problem:\n{{problem}}\nYou should put your code in
→ '''python''' . Use input() to read input and print() to produce output in your
→ script. <|im_end|>\n<|im_start|>assistant<think>\n'''
```

We use the same prompt template for both reinforcement learning and evaluation. TraDo and SDAR models share the same prompt templates. For long-CoT mode, we adopt the “thinking” prompt. The non-CoT prompt is only used for SDAR in our toy experiment in Section 3, which evaluates the effectiveness of different SFT methods in terms of how efficiently they teach the model to use CoT. The prompts are chosen to align as closely as possible with the model’s preferences and to reduce formatting errors (e.g., failure to extract the final answer).

B.3. Evaluation Details

For the LLaDA model, we use a block size of 32, a response limit of 1024 (no output truncation observed), a temperature of 0.1, and a further horizon size of 128. The further horizon size is defined as the additional number of tokens forwarded beyond the target block, excluding the tokens within the target block itself. We find that using this extra window reduces the performance drop caused by employing the KV-cache. When applying the dynamic sampling method, we set the unmasking threshold to $\mathcal{T} = 0.95$.

For the Dream model, we also use a temperature of 0.1, and a further horizon size of 128. For math problems, the response limit is set to 1600, and for coding problems, it is set to 1024. When using dynamic sampling, we use a block size of 4 with threshold $\mathcal{T} = 0.95$. Under static sampling, we use a block size of 32.

For the SDAR and TraDo instruction models, we keep the pretrained block size of 4, a response limit of 2000, and a temperature of 1.0. With dynamic sampling, we use a threshold of $\mathcal{T} = 0.9$ and $top-k = 0$ (i.e., all tokens are kept). For static sampling, we set $top-k = 1$, following (Cheng et al., 2025). For the long-CoT model TraDo-8B-Thinking, we set the response limit to 30,000 during evaluation, and only use dynamic sampling for evaluation.

For AIME2024, we evaluated each problem 20 times. For all other test datasets, we evaluated 3 times and report the average accuracy.

We use the KV-cache in all evaluations to accelerate inference. For full attention mask models, we adapt and improve the fast-dllm framework (Wu et al., 2025). For block diffusion models, we use jetengine (Cheng et al., 2025) to achieve acceleration.

B.4. RL Sampling Details

We describe here the parameters used in our experiments. For the Dream model, we set the block size to 32, the temperature to 0.8, the further horizon size to 128, and the response limit to 1024. During each step, we sample 56 problems, with 8 responses generated for each problem. We employ static decoding (one token per step) to enhance sampling quality (Gong et al., 2025), using the KV-cache. For the SDAR models, we use the default block size of 4, dynamic decoding with threshold $\mathcal{T} = 0.9$, $top-k = 0$, temperature 1.0, and $top-p = 1.0$ (also applied during evaluation). During each step, we sample 128 problems, with 32 responses generated for each problem.

B.5. Training Details

For full attention models, we use a learning rate of 1×10^{-6} for both semi-autoregressive and fully random masking fine-tuning methods. For adapted models such as Dream, autoregressive (AR) training can be applied with a learning rate of 1×10^{-5} as a cold start. Since Dream shares the same architecture as an AR model, this provides a quick and effective way to teach the model specific language patterns. For block attention models, we use a learning rate of 1×10^{-6} . The masking probability is uniformly sampled between 0.1 and 0.9.

During RL training, we set the learning rate to 1×10^{-6} , with $\beta = 0.01$ and $\epsilon = 0.2$. When using a value model, we find no significant difference between the common parameter choices $(\gamma, \lambda) = (1, 1)$ and $(\gamma, \lambda) = (1, 0)$ (Hu et al., 2025a). By default, we use the $k = 3$ estimator for KL. For math tasks, we use binary outcomes as verifiable rewards and retain only those tasks with accuracy between 0.2 and 0.8 for training (Yu et al., 2025a). For coding tasks, we use as the reward the proportion of unit tests passed by the generated solutions. To accelerate training, we use 64 A100 GPUs for our demonstration and ablation experiments. However, all experiments can be conducted on 8 A100 GPUs.

In Figure 6 (a), for Trace RL we use a shrinkage parameter of $s = 8$, together with an average response length of 380 during the RL process, resulting in approximately $380/8 = 47.5 < 50$ forward passes per data point. We augment the training data for fully random masking and coupled methods by applying 25 independent random masks, yielding 50 training samples per data point for the coupled method and 25 training samples per data point for the random masking method. Following Gong et al. (2025), we keep the number of training samples for random masking at half that of the coupled method. Before RL training in Figure 6(a), we collect 1.7k random SFT training samples from CodeContest (Li et al., 2022), with solutions generated by the Qwen2.5-32B-Instruct model. We find that including the eos token in the SFT data is beneficial for stabilizing RL training.

One noteworthy point is the number of pad tokens that need to be trained for each data point, which we denote as n_{pad} . Setting a large n_{pad} leads to excessively large logits for the pad token and can cause the model to terminate inference prematurely. In our RL training, we set $n_{pad} = 0$, which works well. However, in our long-CoT SFT step, we find that setting $n_{pad} = 0$ can potentially cause the model to never stop generating output during inference, although one remedy is to add an eos token as the stop token. Therefore, choosing an appropriate n_{pad} is vital for stable training.

B.6. TraceRL Algorithm Pipeline

Algorithm 1 | TraceRL (Trajectory-Aware RL for DLMs)

```

1: Input:
2:   1) Task set  $\mathcal{D}_{\text{task}} = \{Q_1, \dots, Q_N\}$ .
3:   2) Policy  $\pi_{\theta_p}$  parameterized by  $\theta_p$ .
4:   3) (Optional) Value network  $V_{\theta_v}$  parameterized by  $\theta_v$ ; flag UseValue  $\in \{\text{True}, \text{False}\}$ .
5:   4) Iterations  $M$ , rollouts per task  $G$ , PPO clip  $\epsilon$ , KL coefficient  $\beta$ .
6:   5) Discount  $\gamma$ , GAE parameter  $\lambda$ , learning rates  $(\eta_p, \eta_v)$ .
7:   6) Shrinkage parameter  $s$  (aggregate every  $s$  neighboring trace steps).
8:   7) (Optional) Value update interval  $E_v$  (update value every  $E_v$  iterations).
9: Initialize:  $\theta_p$  (and  $\theta_v$  if UseValue).
10: for  $t = 1$  to  $M$  or not converged do
11:    $\pi_{\text{old}} \leftarrow \pi_{\theta_p}$  // freeze old policy for PPO ratios & KL
12:   if UseValue then
13:      $V_{\text{old}} \leftarrow$  freeze copy of  $V_{\theta_v}$  // stop-gradient baseline
14:   end if
15:   Sample rollouts (trajectory traces):
16:   for each minibatch of tasks  $Q \sim \mathcal{D}_{\text{task}}$  do
17:     for repeat  $G$  times do
18:       Generate a response by policy decoding to obtain a trace  $\tau = (\tau(1), \dots, \tau(|\tau|))$ , where
         $\tau(t)$  is the token set decoded at step  $t$ .
19:       Obtain verifiable reward  $r$  (or optionally process-level token/step rewards).
20:       Shrink the trace:  $\tau^s \leftarrow \text{shrink}(\tau, s)$ , i.e., group every  $s$  neighboring steps.
21:       if UseValue then
22:         Value inference:  $V_j^{\text{old}} \leftarrow (V_{\text{old}}(\tau))_j$  for  $j \in \tau$ ;  $V_t^{\star, \text{old}} \leftarrow \frac{1}{|\tau(t)|} \sum_{j \in \tau(t)} V_j^{\text{old}}$ .
23:         Build returns/advantages on the trace:
24:           Calculate step-wise reward:  $r_t^*$ , return  $R_t^*$ , advantage  $A_t^*$ .
25:           Map to tokens ( $j \in \tau(t)$ ):  $R_j \leftarrow r_j + \gamma R_{t+1}^*$ ,  $A_j \leftarrow (r_j - V_j^{\text{old}}) + \gamma V_{t+1}^{\star, \text{old}} + \gamma \lambda A_{t+1}^*$ .
26:           (Optional) normalize advantages:  $A_j \leftarrow \text{normalize}(A_j)$ .
27:       end if
28:       Store  $(Q, \tau^s, \text{advantages}, \pi_{\text{old}})$  for policy update.
29:     end for
30:   end for
31:   Build grouped dataset  $\mathcal{D}_{\text{grp}} \leftarrow \{(\mathbf{p}_g, O_g, A_g, \pi_{\text{old}})\}$ .
32:   for  $e = 1$  to  $K$  do
33:     Sample minibatch  $\mathcal{M} \subset \mathcal{D}_{\text{grp}}$ ; compute ratios  $r_j = \frac{\pi_{\theta_p}(o_j | \mathbf{p}_g)}{\pi_{\text{old}}(o_j | \mathbf{p}_g)}$ .
34:     Policy step: maximize  $\mathcal{J}_{\text{policy}}(\theta_p)$  (Eq. (3));  $\theta_p \leftarrow \theta_p + \eta_p \nabla_{\theta_p} \mathcal{J}_{\text{policy}}$ .
35:     if UseValue and  $(t \bmod E_v = 0)$  then
36:       Value step: minimize  $\mathcal{J}_{\text{value}}(\theta_v)$  (Eq. (4));  $\theta_v \leftarrow \theta_v - \eta_v \nabla_{\theta_v} \mathcal{J}_{\text{value}}$ .
37:     end if
38:   end for
39: end for
40: Output: Trained policy  $\pi_{\theta_p}$  (and value  $V_{\theta_v}$  if used).

```

C. Additional Experimental Details

C.1. Average Response Length of TraDo-8B-Thinking

Table 5 | Average response length of TraDo-8B-Thinking on different benchmarks.

	MATH500	AIME2024	GSM8K	LiveCodeBench	LiveBench
Avg. length	5872	19397	1458	16291	16233

C.2. Example of Long-CoT outputs

TraDo-8B-Thinking Example Output

task: What is the smallest positive perfect cube that can be written as the sum of three consecutive integers? **response:** Okay, so I need to find the smallest positive perfect cube that can be written as the sum of three consecutive integers. Let me start by understanding the problem step by step.

First, a perfect cube is a number that can be expressed as n^3 where n is a positive integer. For example, 1, 8, 27, 64, etc. The question is asking for the smallest one of these that can also be written as the sum of three consecutive integers.

Now, three consecutive integers can be represented as x , $x+1$, and $x+2$, right? So their sum would be $x + (x+1) + (x+2)$. Let me compute that first.

Adding those together: $x + x + 1 + x + 2 = 3x + 3$. So the sum of three consecutive integers is $3x + 3$, which can be factored as $3(x + 1)$.

Hmm, so the sum is always a multiple of 3. Therefore, any perfect cube that can be written as such a sum must be divisible by 3? Because it's three times some integer.

So, the cube must be divisible by 3. Let me note that down.

Therefore, the smallest perfect cube divisible by 3 is 27 (since $3^3 = 27$), but I need to check if 27 can be written as the sum of three consecutive integers. Let me check that.

Suppose $3x + 3 = 27$. Then $3x = 24 \Rightarrow x = 8$. Therefore, the three numbers would be 8, 9, 10. Let's add them up: $8 + 9 + 10 = 27$. Hey, that works! So 27 is indeed the sum of three consecutive integers.

But wait, hold on. The question says the smallest positive perfect cube that can be written as the sum of three consecutive integers. If 27 works, then is that the answer? But before I jump to conclusions, maybe there's a smaller cube that also satisfies this? Let me check.

The next smaller cube after 27 is 8 (2^3). Let's see if 8 can be written as the sum of three consecutive integers.

Setting up $3x + 3 = 8$. Then, $3x = 5 \Rightarrow x = 5/3$, which is not an integer. Since x has to be an integer, this doesn't work. So 8 is out.

The next is 1 (1^3). Let's see, $3x + 3 = 1 \Rightarrow 3x = -2 \Rightarrow x = -2/3$. Again, not an integer. So 1 is out.

Therefore, the next cube is 64 (4^3). Let's check that. Wait, but 64 is bigger than 27. Since 27 already works, but maybe there is another cube between 8 and 27? Since 8, 27, 64 are the cubes after 1, and 27 is the first one here.

Wait, hold on. The cubes are 1, 8, 27, 64, 125, etc. So between 8 and 27 is 27 which works. So 27 is the first one after 8. But does 27 itself work? Yes, as we saw.

But hold on, maybe the question is considering three consecutive integers in a different way? For example, maybe the integers can be negative? Let me check.

Suppose we allow negative integers. Let's see if there's a smaller cube that could be expressed as the sum of three consecutive (including negative) integers.

Let's check 1. 1 can't, as above. 8 can't. 27 can. 64? Let's see. Let me check 64.

$$3x + 3 = 64. \quad 3x = 61 \Rightarrow x = 20.333\dots \text{ Not integer. So no.} \quad 125? \quad 3x + 3 = 125 \Rightarrow 3x = 122 \Rightarrow x = 40.666 \text{ nope.}$$

So even with negative numbers, the next cube after 27 might still be 27. But let me think again. Maybe there's a different way to write three consecutive integers?

Wait, perhaps three consecutive numbers can be written as $n-1, n, n+1$? That would still be the same as $x, x+1, x+2$ where $x = n-1$. So yeah, same thing.

Alternatively, maybe three consecutive even or odd integers? But that would complicate the sum, but the problem didn't specify. The question just says three consecutive integers, so they can be any consecutive integers, whether positive or negative.

Alternatively, maybe I made a mistake in assuming that 27 is the first one. Let me confirm again.

The sum of three consecutive integers is $3x + 3 = 3(x + 1)$. So the cube must be divisible by 3. Therefore, the cube must be a multiple of 3. The first cube divisible by 3 is 27 (3^3). So yes, that must be the case.

Therefore, unless there is a smaller cube that is divisible by 3, but 1 is not, 8 is not, so 27 is the first one.

Therefore, the answer should be 27. But hold on, let me check with another approach to be sure.

Suppose the three consecutive integers are $n, n, n+1$. Wait, no. Three consecutive integers are $n, n+1, n+2$. So their sum is $3n + 3$. So we need $3(n + 1)$ to be a perfect cube.

Therefore, $3(n + 1) = k^3$, where k is an integer. Therefore, $n + 1 = k^3 / 3$.

Since n must be an integer, k^3 must be divisible by 3. Therefore, k must be divisible by 3, because 3 is prime.

Therefore, let $k = 3m$, where m is an integer. Then:

$$n + 1 = (27m^3)/3 = 9m^3.$$

Therefore, $n = 9m^3 - 1$.

Therefore, the three integers would be $9m^3 - 1, 9m^3, 9m^3 + 1$. Their sum is $9m^3 - 1 + 9m^3 + 9m^3 + 1 = 27m^3$, which is indeed a cube.

Therefore, the cubes are all multiples of 27. Therefore, the smallest such cube is when $m=1$, which is $27*1^3 = 27$.

Therefore, 27 is indeed the smallest perfect cube that can be written as the sum of three consecutive integers.

Wait, but let me check $m=0$? If $m=0$, then $n = -1$, so the numbers would be $-1, 0, 1$. Their sum is 0, which is not a positive cube. So $m=1$ is the first positive case.

Therefore, this seems solid.

Therefore, the answer is 27.

But hold on, let me think again. The problem says smallest positive perfect cube. Since 27 is positive and a perfect cube, so it's the first one.

Alternatively, maybe there's a cube before 27 that can be expressed as the sum of three consecutive integers? Let me confirm again. The cubes are 1, 8, 27, 64, 125,...

Checking 1: $3x + 3 = 1$: $3x = -2$, nope.

8: $3x + 3 = 8$: $3x = 5$, nope.

27 works, so yeah, 27 is the first. Therefore, I think that's the answer.

Wait, but maybe I made a mistake in the equation. Let me recast the problem.

..... (We omit some reasoning steps here.)

Therefore, 27 is indeed the first.

Therefore, I think the answer is 27.

Wait, but the problem says smallest positive perfect cube. Since 27 is positive and perfect cube, and there are no smaller cubes that work, then yes, 27 is the answer.

Wait, but just to make sure, is 0 considered? But 0 is not positive. So 27 is the first.

Therefore, I think I can confidently say that the smallest positive perfect cube that can be written as the sum of three consecutive integers is 27.

Final Answer

27