



Read the Trace, Steer the Path: Trajectory-Aware Reinforcement Learning for Diffusion Language Models

Anant Khandelwal, Manish Gupta

MAI Microsoft AI, India

{anantk, gmanish}@microsoft.com

<https://github.com/infusion-zero-edit/CAPR>

<https://huggingface.co/collections/anantk/capr-checkpoints>

Abstract

Diffusion large language models (dLLMs) generate responses by iteratively unmasking and revising many positions in parallel. This process leaves a rich denoising trace depicting which tokens become confident, which remain unstable, and when commitments form. Existing dLLM reinforcement learning methods use this signal only weakly. Flat rollouts are cheap, but assign a single outcome reward to the whole trajectory. Tree rollouts provide finer, verifiable training signals by branching partial trajectories and propagating leaf rewards upward, but is compute intensive. We ask whether the denoising trace itself can provide tree-like supervision without tree-level compute. We introduce CAPR (Cached-Amortized Path Refinement), a dLLM-RL algorithm that summarizes the denoising trace into a compact path state, uses cached trajectory states to generate cheap sibling continuations, and trains a block-level value head for local block-wise supervision. Under a block-wise unmasking schedule, CAPR records path-state and block-progress features, then redistributes the final outcome reward across blocks according to the tokens revealed in each block. This trains the value head to convert one sparse reward into block-level PPO weights. CAPR therefore recovers much of the granularity of tree search while avoiding full tree expansion, reducing rollout-generation cost to roughly $0.75\times$ of flat rollouts and $0.6\times$ of tree rollouts (at standard setting). Across 4×4 Sudoku, Countdown, GSM8K, and Math500, on dense and mixture-of-experts LLaDA backbones, CAPR sets a new state of the art for RL-tuned dLLMs at 256- and 512-token budgets. On Sudoku, it matches the strongest tree-structured baseline at less than one third of the per-step compute.

1 Introduction

Diffusion large language models (dLLMs) (Nie et al., 2025b; Zhu et al., 2025b) generate text by denoising a fully masked sequence over many parallel

steps. At every step, the model exposes a predictive distribution for every masked position. This is a rich trace of how the answer forms, but current reinforcement learning (RL) based post-training for dLLMs (Zhao et al., 2025a; Zhu et al., 2025a; Tang et al., 2026; Xie et al., 2026; Pan et al., 2026) uses little of it. An end-of-generation reward must still be assigned across many denoising steps and token blocks, so the learner has no direct signal for which step fixed an important token and which step merely changed an irrelevant one.

Existing RL methods for dLLMs sit at two extremes. Flat-rollout methods assign the same outcome reward to all steps. They are cheap, but they cannot tell which denoising decisions mattered. Tree-rollout methods (Pan et al., 2026) fork alternative continuations and compare their final rewards. They obtain finer advantages, but their cost grows quickly with branching factor and depth. Figure 1 (right) shows this cost-granularity trade-off. The denoising trace offers a third route. At each step, a position either becomes more stable and confident, or it keeps changing. Figure 1 (left) illustrates this on a simple prompt (“7+5:”). By step three the tens digit has settled at 1, while the units digit is still oscillating. The trace therefore tells us which positions are becoming committed, without running a tree search. The key question is how to convert this trace into a learning signal.

We answer this question with CAPR (Cached-Amortized Path Refinement), an RL algorithm for block-based dLLMs. CAPR consists of three broad mechanisms. a) *Cache & Steer*: CAPR records the early denoising steps, builds a compact path state from confidence and stability at each position, and reuses this cached prefix for producing sibling rollouts or branches. The path state is used immediately at every denoising step to steer the next-step logits, reinforcing stable commitments and damping oscillating ones. b) *Branch & Prune*: At a fixed mid-trajectory step, CAPR branches a

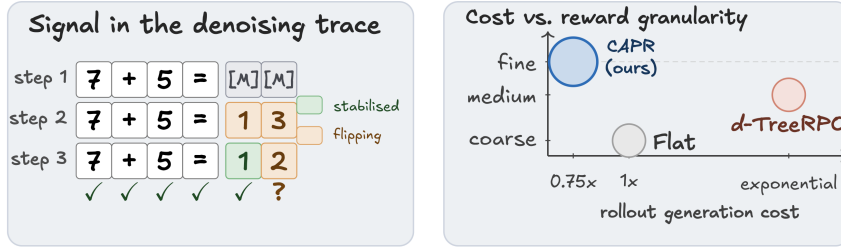


Figure 1: The denoising trace contains information that current RL methods do not use. Left: each masked position has a sequence of provisional predictions however stable and confident positions can be separated from positions that keep changing. Right: prior dLLM-RL methods occupy two extremes of the cost–granularity plane. Flat rollouts are cheap but give one scalar advantage per trajectory, *d-TreeRPO* (Pan et al., 2026) pays tree-expansion cost for a small number of segment-level advantages. The approach introduced in this paper named, CAPR, uses the trace to obtain per-block advantages at roughly $0.75\times$ flat-rollout cost.

small number of Gumbel-perturbed sibling continuations (rollouts when completed) from the cached prefix. At block boundaries, it keeps the siblings whose path states indicate the strongest commitments while pruning others. Finally, a small c) *Block Critic* that reads the path state at each block boundary and converts the outcome reward into one advantage per block for the PPO update. Figure 2 depicts the three mechanisms. This design keeps the useful part of tree methods which makes local comparison among alternatives, without expanding a full tree.

Due to caching and pruning, CAPR is much more efficient compared to both flat rollouts as well as tree rollouts. Figure 1(right) shows that CAPR consumes $0.75\times$ of the flat rollout generation cost. At standard *d-TreeRPO* tree rollout settings of branching factor= 4 and tree depth= 2, CAPR incurs $0.6\times$ of the tree rollout generation cost as shown in Table 6 in Appendix B.

CAPR is not just efficient but also more accurate compared to both flat and tree rollouts. On four reasoning benchmarks (4×4 Sudoku, Countdown, GSM8K, Math500) and two open-source LLaDA backbones spanning dense and mixture-of-experts variants, CAPR sets a new state of the art for RL-tuned dLLMs at both the 256-token and 512-token decoding budgets. On Sudoku, it matches the strongest tree-structured baseline at less than a third of the per-step compute (Table 2). Ablations show that all three mechanisms help, with per-block advantages providing the largest gain.

Contributions.

- A denoising trace-based view of RL for masked dLLMs: confidence and stability along the denoising path can be turned into per-block advantages, without full tree expansion.
- CAPR, a block-based RL algorithm that caches

early denoising outputs, amortizes sibling rollouts from the cached prefix, steers each next denoising step with a path state, and uses a block critic to turn the final reward into one advantage per block for PPO.

- State-of-the-art results for RL-tuned dLLMs at both 256-token and 512-token decoding budgets across four reasoning benchmarks and two LLaDA backbones, with less than a third of the per-step compute of the strongest tree-structured baseline on Sudoku.

2 Related Work

Diffusion language models. Diffusion language models generate text by iteratively denoising masked tokens rather than predicting strictly left to right (Nie et al., 2025b; Zhu et al., 2025b; Sahoo et al., 2024; Shi et al., 2024; Nie et al., 2025a; Ye et al., 2025b,a; Labs et al., 2025; Song et al., 2025). This decoding process exposes a distribution over every masked position at every step, which makes dLLMs a natural setting for trace-based training signals. Recent open models and paradigms, including LLaDA-1.5 (Zhu et al., 2025a), LLaDA-MoE (Zhu et al., 2025b), Dream-7B (Ye et al., 2025a), and SDAR (Cheng et al., 2025), have made dLLMs strong reasoning backbones. We use the dense and MoE LLaDA variants in our experiments.

RL post-training for dLLMs. Most RL methods for dLLMs differ in how they assign an outcome reward to the denoising process. Flat-rollout methods, including Diffu-GRPO (Zhao et al., 2025a), VRPO (Zhu et al., 2025a), wd1 (Tang et al., 2026), and d2-stepMerge (Wang et al., 2026), keep sampling cheap by applying one trajectory-level reward to all token updates. This makes the update simple, but it does not identify which denoising steps

or blocks mattered. Step-aware methods such as SAPO (Xie et al., 2026) and TraceRL (Wang et al., 2025) attach more local signals inside a rollout, but they do not use the denoising trace to learn a block-level value function. Tree-based methods, especially d -TreeRPO (Pan et al., 2026), compare branched continuations and produce segment-level advantages. They are effective, but the cost grows rapidly with branching factor and depth. CAPR targets the gap between these regimes. It keeps the rollout structure close to flat sampling, but uses the trace to produce one advantage per block.

Concurrent work explores related ways to stabilize or enrich dLLM-RL, including consistency-trajectory regularization (Yang et al., 2025a), lateral-thought reinforcement (Huang et al., 2025b), multimodal RL for dLLMs (Yang et al., 2025b), entropy and anchoring objectives (Yang et al., 2026; Cui et al., 2025; Li et al., 2026), boundary-aware updates (Lin et al., 2025), MDPO (He et al., 2025), principled denoising objectives (Ou et al., 2025), and inpainting-style RL (Zhao et al., 2025b). CAPR is complementary to these directions. Its main contribution is to convert the denoising trace into reusable path states and per-block PPO weights.

Connections to policy optimization and guided decoding. CAPR’s block critic is a lightweight actor–critic component (Sutton, 1988; Schulman et al., 2016, 2017) adapted to sparse, outcome-only rewards. It reads features that are specific to masked decoding: path-state statistics, block progress, prompt-level reward statistics, and the tokens revealed so far. A slowly-updated target copy (Lillicrap et al., 2016) and a small replay buffer keep this online value head stable.

The steering shift is related to guided decoding, but it does not use an external classifier or reward model. It uses the model’s own trace. Stable, confident positions receive a small positive bias on the next denoising step, while oscillating positions are damped. The form follows the additive logit control induced by linearly-solvable MDPs (Todorov, 2006). Finally, CAPR uses a one-sided χ^2 trust region to suppress rare token-level log-ratio spikes without penalizing all policy movement equally. This is related in spirit to asymmetric clipping (Yu et al., 2025; Xie et al., 2026) and policy-gradient stabilization (Cui et al., 2025; Zhang et al., 2025; Tan et al., 2026; Zheng et al., 2025), but it is applied to token-level dLLM updates with an explicit

upper-tail trust-region interpretation.

3 Method

3.1 Preliminaries

A masked dLLM π_θ generates a completion y for prompt q over N denoising steps, moving from $z_N = [q; [\text{MASK}]^L]$ to $z_0 = [q; y]$. Block decoding partitions the L completion tokens into blocks. Flat GRPO (Shao et al., 2024) optimizes a clipped policy-gradient objective with a rollout-level outcome advantage shared by all tokens in the completion. Tree methods (Pan et al., 2026) obtain more local advantages by comparing branched continuations. CAPR keeps the sampling pattern close to flat GRPO, but replaces the single rollout-level advantage with one advantage per block. We write $r_{ij}(\theta) = \pi_\theta(y_{ij})/\pi_{\theta_{\text{old}}}(y_{ij})$ for the token-level importance ratio used by the PPO update in CAPR.

The CAPR methodology is illustrated in Fig. 2. During the early denoising steps it records how each masked position behaves across steps. Stable, confident positions are treated as emerging commitments, oscillating positions are treated as unresolved. This record is compressed into a path state. The path state is then used in three ways: it steers the transition z_t to z_{t-1} , later branches reuse a cached prefix, and it gives the block critic the information needed to turn one final reward into one advantage per block.

3.2 CACHE & STEER

At each denoising step t , CAPR reads how each position j is behaving by recording confidence in the current top token, entropy of the probability distribution per position at the current step, and whether the top token stayed stable across denoising steps. These are packed into a small feature vector $\phi_j^{(t)} = (c_j^{(t)}, s_j^{(t)}, e_j^{(t)}, a_j^{(t)})$ where c_j is confidence, s_j is top-token stability, e_j is normalized entropy, and a_j is certainty. The feature computations are defined in App. A.1.

The path state $h_j^{(t)}$ is updated in GRU-style (Cho et al., 2014). A reset gate forgets stale evidence when entropy is high, and an update gate preserves memory when the token has remained stable and certain. Thus h_j acts as a signed memory of commitment. Positive h_j means the position is stable, negative h_j means the model is still oscillating. The exact analytic gates are defined in App. A.1.

$$h_j^{(t)} \leftarrow \text{PathGRU}(h_j^{(t-1)}, \phi_j^{(t)}), \quad (1)$$

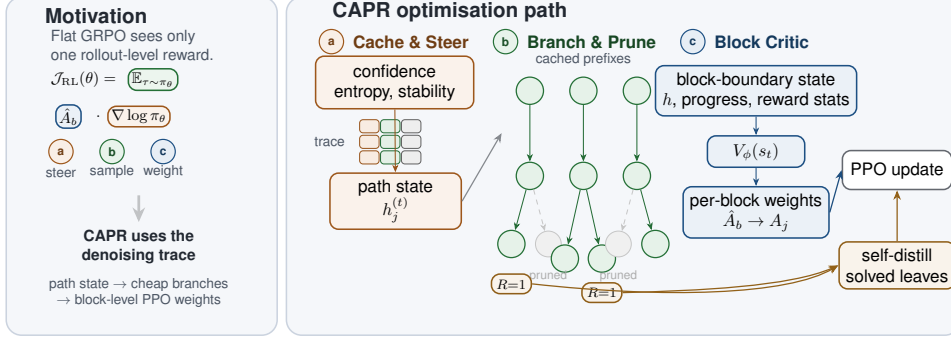


Figure 2: Overview of CAPR. **(a) Trace State / Cache & Steer:** CAPR summarizes per-position confidence, entropy, and stability into a path state, then uses it to steer the next reverse step while carrying only the previous clean-token prediction and path state. **(b) Branch & Prune:** a shared denoising prefix is forked once at the branch step, and siblings are kept by path-state quality. **(c) Block Critic:** a value head reads block-boundary states and converts the final reward into per-block PPO weights. Self-distillation uses solved rollouts as supervised targets so zero-variance groups still provide learning signal.

At step t , CAPR uses z_t to steer the resulting logits and then identifies the positions and sampled token predictions to form z_{t-1} . Across steps, only z_t and the path state $h_j^{(t)}$ are carried over and the logits themselves are recomputed each step. CAPR steers the logits as follows.

$$\ell_j^{(t)} += \frac{h_j^{(t)} c_j^{(t)} (2s_j^{(t)} - 1)}{\lambda} \quad (2)$$

where λ is a path state steering scaling constant. A stable and confident token receives a positive push and an oscillating token is dampened.

The same path state adjusts the confidence used to choose which masked positions are revealed as well as which unmasked positions should be re-masked. First, we compute $\kappa_j^{(t)}$ as follows.

$$\kappa_j^{(t)} = c_j^{(t)} + \gamma h_j^{(t)} a_j^{(t)} \frac{1 + s_j^{(t)}}{2}, \quad (3)$$

where γ is a path state injection scaling constant.

κ_j steers which positions are trusted enough to unmask. If the denoising schedule decides to unmask K_t tokens, CAPR selects

$$\mathcal{U}_t = \text{TopK}_{j:z_{t,j}=[\text{MASK}]}(\kappa_j^{(t)}, K_t), \quad (4)$$

For these positions in \mathcal{U}_t identified for unmasking, the logit bias steers which token is sampled. After sampling, the new prediction replaces z_t for the following denoising step and the logits themselves are not carried forward. Both steering signals in Eq. 3 and 4 come from the model’s own denoising trace rather than an external classifier.

3.3 BRANCH & PRUNE

To explore alternative completions (see App. D), and to assign granular rewards for every block in every branch, at low compute cost, we perform branching at the midpoint of the denoising trajectory in CAPR. To reduce compute costs for each branch, CAPR caches a prefix so that these sibling continuations (i.e. branches) can start without any extra forward pass. The prefix is a combination of logits distribution at the branch step, along with the predicted tokens, and path state $h^{(t)}$.

At a fixed fraction f_b (we set it to 0.5) of the denoising budget N , it forks G_b branches from the shared prefix. The fork reuses the stored branch-step prefix and changes only the predictions for the G_b branches by sampling a different \mathcal{G}_v for every vocab token v in each branch. The predictions for each branch are then calculated as $\arg \max_v (\ell_{j,v}^{(t)}/\tau + \mathcal{G}_v)$ for every position j in \mathcal{U}_t at $t = f_b \times N$ with $\mathcal{G}_v \sim \text{Gumbel}(0, 1)$. τ is the sampling temperature. Each sibling copies the current path state and then completes the remaining denoising steps independently.

This prefix caching gives the compute advantage. One branch group costs $[f_b + G_b(1 - f_b)]N$ forward steps, detailed comparison with baselines on forward cost is given in App. B. At $G_b=2$, and $f_b=0.5$, cost per rollout is $\frac{1}{G_b} [f_b + G_b(1 - f_b)]N = 0.75N$. To produce G rollouts, we run $M = \lceil G/G_b \rceil$ such groups.

At a regular frequency schedule after the fork, CAPR keeps the most useful branches rather than carrying all of them forward. The score is computed based on mean path state $h^{(t)}$ of the branch and hence branches with more positions settled

into stable predictions receive higher scores. This pruning is deliberately local and cheap. It does not require evaluating a reward model at intermediate steps, it uses the same trace signal already maintained by “Cache & Steer”.

Tree method branches many times and compares final rewards of many continuations, but this expands compute quickly. CAPR instead creates a small local comparison set by forking sibling continuations from a cached mid-trajectory state. Since the siblings share the same prefix and differ only in the suffix, their outcomes and path-state quality give useful evidence about which later denoising decisions are promising. Pruning them prevents the small branch set from growing into a full tree.

3.4 BLOCK CRITIC

The block critic answers a simple question: after seeing only the final reward, which blocks were most useful? At each block boundary, CAPR stores a compact state containing the path state, block progress (number of unmasked tokens within the block), prompt-level reward statistics, and tokens revealed so far. Using this compact state, the value head is trained as discussed in App. A.2.

After the rollout finishes, CAPR redistributes the final reward R over blocks according to how many tokens each block revealed, with the final block also receiving the terminal outcome signal. TD/GAE (Schulman et al., 2016; Sutton, 1988) with a slowly updated target head $V_{\bar{\phi}}$ (Lillicrap et al., 2016) then produces one block advantage \hat{A}_b for each block b . Thus, the value head converts one sparse outcome reward into block-level weights over the denoising path.

For PPO, these block advantages are used only to reweight the rollout-level GRPO advantage. Let $b_i(j)$ denote the block in which token j of rollout i is revealed. CAPR computes

$$\begin{aligned} w_{i,b} &= B \operatorname{softmax}(\hat{A}_{i,1:B})_b, \\ \tilde{A}_{i,j} &= A_i^{\text{GRPO}} [(1 - \beta_{\text{TD}}) + \beta_{\text{TD}} w_{i,b_i(j)}], \end{aligned} \quad (5)$$

where B is the number of blocks. The factor B keeps the mean block weight near one, so the block critic redistributes the rollout-level signal rather than changing its overall scale. Every token revealed in the same block receives the same block weight. When the value head is poorly calibrated, the blend β_{TD} is reduced and PPO falls back toward the flat GRPO advantage. Appendix A.2 gives the value-head features, TD targets, and blending

gate. At inference, the value head is removed and only the trained LoRA-adapted policy remains.

3.5 Training Objective and CAPR Algorithm

The complete CAPR objective is

$$\begin{aligned} \mathcal{L}_{\text{CAPR}} &= \mathcal{L}_{\text{PPO}} + \beta \mathcal{L}_{\text{KL}} + \lambda_f \mathcal{L}_{\chi^2} \\ &\quad + \mathcal{L}_{\text{NLL}} + c_v \mathcal{L}_V. \end{aligned} \quad (6)$$

In Eq. 6, \mathcal{L}_{PPO} is the clipped PPO loss using the block-weighted advantages from Eq. 5, $\beta \mathcal{L}_{\text{KL}}$ keeps the policy near the reference model, $\lambda_f \mathcal{L}_{\chi^2}$ is a chi-square regularizer that damps large positive log-ratio deviations, \mathcal{L}_{NLL} is the self-distillation loss on solved rollouts, and $c_v \mathcal{L}_V$ trains the block critic. The joint $\text{KL} + \chi^2$ design is motivated by the observation that KL regularization alone is too weak to prevent over-optimization of sparse rewards (Huang et al., 2025a). Appendix A.2 gives the value-head features and calibration, Appendix A.3 defines \mathcal{L}_{NLL} and its schedule, and Appendix A.4 defines \mathcal{L}_{χ^2} . Algorithm 1 summarizes one optimization iteration.

Inference. At deployment, CAPR uses only the trained LoRA policy with the standard single-path dLLM denoising loop. Cache & Steer, branching, pruning, the block critic, and self-distillation are training-only mechanisms. Their signal has already been distilled into the policy weights at train time. Inference therefore uses the same number of model forwards as the base dLLM, with no value head or training-time path state.

4 Experiments

We evaluate CAPR on four verifiable reasoning benchmarks under matched decoding budgets. The experiments ask three questions: whether block-level attribution improves final accuracy, whether the gains hold across dense and mixture-of-experts dLLMs, and whether the extra training signal is cheaper than tree-structured rollouts.

4.1 Experimental Setup

Models and tasks. We evaluate LLaDA-8B-Instruct (Nie et al., 2025b) and LLaDA-MoE-7B-A1B-Instruct (Zhu et al., 2025b). Both are trained with rank-64 LoRA adapters on frozen base weights, bf16 activations, fp32 optimizer state, and DeepSpeed ZeRO-2. Benchmarks are 4×4 Sudoku, Countdown, GSM8K (Cobbe et al., 2021), and Math500 (Lightman et al., 2024). Sudoku

Table 1: Pass@1 on four reasoning benchmarks at 256- and 512-token generation budgets. CAPR is evaluated with the same zero-shot setting, including prompts, parser, deterministic decoding, and token budget as in d -TreeRPO. Gray values show gains over the corresponding base model.

Method	Sudoku		Countdown		GSM8K		Math500	
	256	512	256	512	256	512	256	512
<i>LLaDA-8B-Instruct</i>								
Base	6.7	5.5	19.5	16.0	76.7	78.2	32.4	36.2
+ Diffu-GRPO	12.9	11.2	31.3	37.1	79.8	81.9	34.1	39.0
+ VRPO	12.8	9.6	22.3	18.0	80.1	81.5	35.6	34.8
+ wd1	25.2	24.2	51.2	46.1	80.8	82.3	34.4	39.0
+ SAPO	20.3	16.1	52.0	56.3	80.6	82.1	33.8	38.4
+ d2-stepMerge	76.1	66.2	52.4	52.1	81.1	82.0	34.4	38.5
+ TraceRL	25.6	25.4	50.4	52.6	80.3	82.4	35.6	39.1
+ d -TreeRPO	92.9	80.3	71.1	62.1	81.2	82.6	37.7	38.9
	93.0	92.7	72.7	69.9	82.4	82.4	37.9	39.8
+ CAPR	+86.3	+87.2	+53.2	+53.9	+5.7	+4.2	+5.5	+3.6
<i>LLaDA-MoE-7B-A1B-Instruct</i>								
Base	12.7	8.1	42.6	41.4	71.8	71.0	30.1	42.2
+ Diffu-GRPO	17.3	15.4	50.1	48.2	74.2	73.7	38.1	43.4
+ wd1	40.6	35.4	56.6	58.7	76.2	76.4	39.8	43.4
+ SAPO	21.2	18.7	54.2	54.8	75.2	76.8	38.6	40.4
+ GDPO	31.2	30.6	58.1	53.4	74.8	75.2	38.4	41.2
+ TraceRL	27.4	25.2	54.2	49.1	75.4	76.1	40.0	44.1
+ d -TreeRPO	78.3	72.0	67.2	60.6	75.5	77.4	41.2	46.3
	84.8	81.0	68.8	72.7	79.4	77.5	41.6	50.4
+ CAPR	+72.1	+72.9	+26.2	+31.3	+7.6	+6.5	+11.5	+8.2

and Countdown stress exact constraint satisfaction, while GSM8K and Math500 test mathematical reasoning from natural-language prompts.

Evaluation. We follow the d -TreeRPO evaluation script (Pan et al., 2026): zero-shot pass@1, deterministic decoding ($\tau=0$), block length $\ell_{\text{blk}}=32$, and denoising steps $N=L/2$.

Baselines and hyperparameters. Baselines include the base model, Diffu-GRPO (Zhao et al., 2025a), VRPO (Zhu et al., 2025a), wd1 (Tang et al., 2026), SAPO (Xie et al., 2026), d2-stepMerge (Wang et al., 2026), TraceRL (Wang et al., 2025), GDPO where available, and d -TreeRPO (Pan et al., 2026). CAPR uses $G=8$ rollouts, Branch & Prune with fork fraction $f_b=0.5$ and $G_b=2$ sibling continuations, GAE $(\gamma, \lambda_{\text{gae}})=(0.99, 0.95)$, value coefficient $c_v=0.5$, chi-square regularizer coefficient $\lambda_f=0.01$, reference-KL coefficient $\beta=0.01$, learning rate 3×10^{-5} , and max self-distillation weight $w_{\text{sd}}^{\text{max}}=0.10$. Baselines reproduced from d -TreeRPO (Pan et al., 2026). App. C gives the full configuration.

4.2 Main Results

Table 1 reports results for the two decoding budgets $L=256$ and $L=512$ in separate columns so short- and long-budget decoding are compared under the

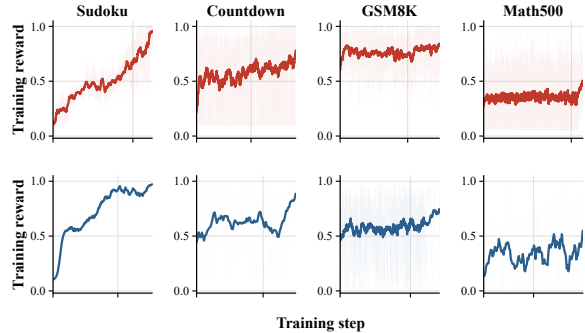


Figure 3: Training reward for the eight CAPR runs. Curves are centered rolling means with raw reward shown faintly. Sudoku shows the clearest cold start, GSM8K and Math500 start from stronger base-model competence. (Top) LLaDA-8B-Instruct (Bottom) LLaDA-MoE-7B-A1B-Instruct. Refer App. F for contrast with baselines

same parser, prompt format, and token limit for every method. At $L=256$, CAPR is best or tied-best on all reported tasks for both backbones.

On LLaDA-8B-Instruct, at $L=256$, CAPR reaches 93.0 on Sudoku, 72.7 on Countdown, 82.4 on GSM8K, and 37.9 on Math500. Compared with d -TreeRPO at $L=256$, the gains are +0.1, +1.6, +1.2, and +0.2 points, respectively. The largest absolute improvements over flat-rollout baselines occur on Sudoku and Countdown, where the final reward is sparse and a single rollout-level advantage gives weak reward assignment.

Fig. 3 shows that training rewards improve consistently across most tasks, with Sudoku exhibiting the strongest cold-start effect and the steepest gains during training. In contrast, GSM8K and Math500 begin from substantially higher initial rewards, indicating stronger base-model competence but comparatively smaller RL-driven improvements. The MoE backbone generally shows noisier optimization dynamics than the dense model, though both converge to stable high-reward regimes.

On LLaDA-MoE-7B-A1B-Instruct, CAPR obtains 84.8 on Sudoku, 68.8 on Countdown, 79.4 on GSM8K, and 41.6 on Math500. This is +6.5 over d -TreeRPO on Sudoku and +1.6 on Countdown, +4.1 on GSM8k while slightly improving the best prior scores on Math500. Thus, same training recipe transfers from dense backbone to sparse-MoE backbone without task-specific retuning. Case study in App. H

Table 2 reports wall-clock cost on $8 \times A100$ -80GB. CAPR takes 165 seconds per optimization step versus 596 seconds for d -TreeRPO. The saving comes from sharing the early denoising prefix

Table 2: Compute cost on 4×4 Sudoku with LLaDA-8B-Instruct. **Batch** = wall-time per rollout batch; **Update** = wall-time per parameter update; **Conv.** = total training time to convergence. Baselines reproduced from (Pan et al., 2026). For all batch size = 4/device

Method	Batch (s)	Update (s)	Conv. (h)	Acc (%)
Diffu-GRPO	111	9.12	≈ 24	12.9
wd1	86	7.21	≈ 24	25.2
SAPO	423	35.25	≈ 72	20.3
GDPO	153	12.72	≈ 48	25.7
TraceRL	604	43.14	≈ 48	25.6
<i>d</i> -TreeRPO	596	9.91	≈ 48	92.9
CAPR	165	10.10	≈ 24	93.0

Table 3: Module ablation on 4×4 Sudoku (LLaDA-8B-Instruct). Br/Pr/Cr/SD denote Branch, Prune, Block Critic, and self-distillation. Metrics are computed over the last 200 steps.

Br	Pr	Cr	SD	r_{peak}	r_{last200}	D_{KL}	zsr
✓				0.27	0.20	0.61	0.00
✓	✓			0.36	0.34	1.30	0.00
		✓		0.48	0.50	3.07	0.00
✓		✓		0.53	0.46	0.81	0.00
✓	✓	✓		0.85	0.69	0.16	0.00
✓	✓	✓	✓	1.00	0.83	1.15	0.22
✓	✓	✓	✓	0.90	0.83	0.21	0.10

and spending extra forwards only on a small number of suffix continuations, while the block critic supplies the per-block attribution.

4.3 Ablation: Contribution of Each Mechanism

Table 3 and Figures 4 and 5 isolate the CAPR mechanisms on Sudoku, where sparse rewards make reward assignment most visible. Branch-only training improves over Diffu-GRPO but stays in the low-reward regime, adding pruning gives only a modest lift. The block critic is the first component that changes the learning curve substantially. By itself it reaches 0.48 peak reward, and together with Branch & Prune it reaches 0.85. This shows that selective suffix exploration and per-block attribution are complementary rather than interchangeable.

The full model adds self-distillation on top of this branch-critic stack. In Figure 4, the full CAPR curve rises into the high-reward band early and remains above the non-distilled variants for most of training. Figure 5 shows the same progression as peak rewards: 0.27 for Branch only, 0.36 with pruning, 0.85 with Branch & Prune plus the Block Critic, and 0.90–1.00 for full CAPR depending on maximum self-distillation weight $w_{\text{sd}}^{\text{max}}$. We select $w_{\text{sd}}^{\text{max}}=0.10$ because it keeps the last-200 reward equal to the highest-peak variant while reducing KL from 1.15 to 0.21.

Figure 6 explains the role of self-distillation. Without self-distillation, zero-variance groups con-

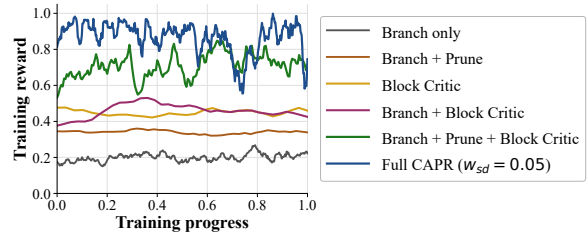


Figure 4: Sudoku training reward under progressive activation of CAPR mechanisms. Branching and pruning alone give small gains, the block critic changes the slope, and self-distillation moves the curve into the high-reward band.

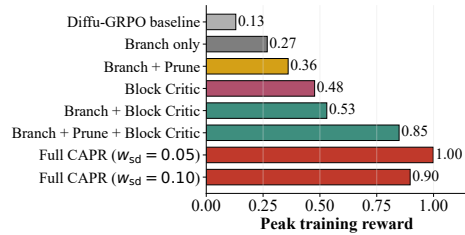


Figure 5: Peak Sudoku reward by ablation configuration. The bar chart mirrors Table 3. The largest jump occurs when Branch & Prune is paired with the Block Critic, followed by self-distillation.

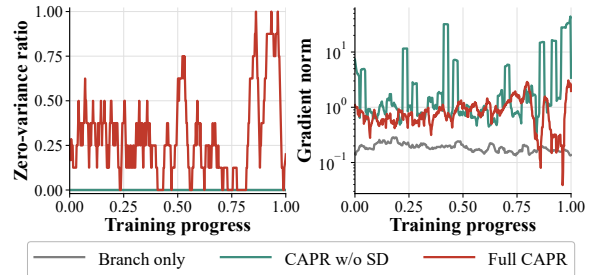


Figure 6: Sudoku stability diagnostics. Left: self-distillation (SD) turns solved zero-variance groups into usable signal. Right: full CAPR avoids the large gradient spikes seen without self-distillation.

tribute no relative advantage and therefore provide little training signal. Full CAPR reuses those solved groups as supervised targets, so the zero-variance ratio rises when more rollouts are solved instead of becoming dead data. The gradient panel shows the trade-off. Branch only has small gradients but weak learning, CAPR without self-distillation has large spikes, and full CAPR keeps useful solved-group signal while producing a smoother late-training gradient profile.

4.4 Training Dynamics

Table 4 and Figure 7 check whether the accuracy gains come from unstable policy drift. For the dense backbone, reference KL stays below the 0.4 guide line in all four tasks. Countdown and

Table 4: Training-time diagnostics for the eight main CAPR experiments. Reward statistics use the last 200 steps up to the selected peak; the remaining columns are medians over the same window.

Benchmark	r_{peak}	r_{last200}	D_{KL}	χ^2_{reg}	$\mathcal{L}_V \times 10^3$	$ \nabla $
<i>LLaDA-8B-Instruct</i>						
Sudoku	0.90	0.83	0.29	0.25	1.0	0.00
Countdown	0.87	0.63	0.07	0.18	4.0	0.05
GSM8K	0.81	0.78	0.06	0.14	6.0	0.08
Math500	0.45	0.36	0.07	0.21	5.0	0.06
<i>LLaDA-MoE-7B-A1B-Instruct</i>						
Sudoku	0.97	0.72	1.02	0.09	17.0	0.13
Countdown	0.94	0.60	0.57	0.15	13.0	0.13
GSM8K	0.86	0.64	0.06	0.28	7.0	0.12
Math500	0.55	0.34	0.97	0.03	7.0	0.14

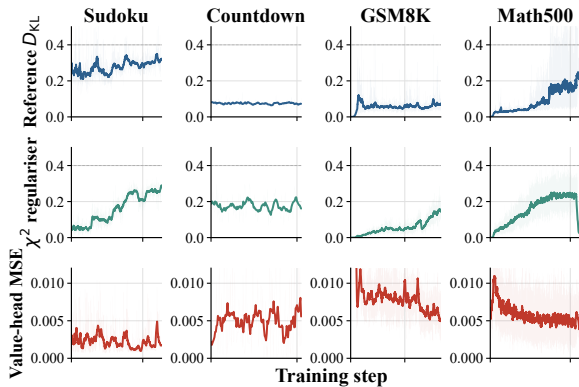


Figure 7: Dense-backbone diagnostics. Reference KL stays below the guide line, the chi-square regulariser remains bounded, and the value-head MSE stays small across benchmarks.

GSM8K remain especially close to the reference model, while Sudoku and Math500 move more because they require larger changes from the base policy. The chi-square regularizer statistic rises when learning accelerates but stays bounded, and the value-head MSE remains small, indicating that the block critic is tracking the changing policy rather than diverging.

The MoE rows in Table 4 show the same qualitative pattern with higher task-dependent KL on Sudoku and Math500, which is expected for routing-sensitive logits. Importantly, the reported checkpoints still have controlled gradient norms and stable last-window rewards. Figure 8 gives the complementary view. The block critic loss typically peaks when the policy is changing fastest and then falls or stabilises, while the self-distillation NLL tracks how predictable solved rollouts become. These modules are therefore useful as training-time attribution and stabilization signals, but they are removed at inference, only the trained LoRA policy is used.

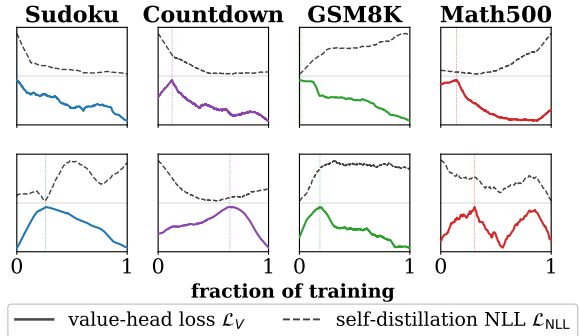


Figure 8: Block Critic loss and self-distillation NLL for the eight CAPR runs. Curves are min-max normalized within each panel, peaks mark phases where the policy changes fastest. (Top) LLaDA-8B-Instruct (Bottom) LLaDA-MoE-7B-A1B-Instruct.

Table 5: Hyperparameter sensitivity on 4×4 Sudoku (LLaDA-8B-Instruct). Entries are peak training reward for maximum self-distillation weight $w_{\text{sd}}^{\text{max}}$ and learning rate η , other settings are in App. C.

Configuration	$\eta=3 \times 10^{-5}$	$\eta=5 \times 10^{-5}$
$w_{\text{sd}}^{\text{max}}=0$	0.85	0.63
$w_{\text{sd}}^{\text{max}}=0.05$	1.00	0.85
$w_{\text{sd}}^{\text{max}}=0.10$	0.90	0.78

4.5 Hyperparameter Sensitivity

Table 5 varies the two most important optimization knobs on Sudoku. The lower learning rate is consistently better. The selected setting, $w_{\text{sd}}^{\text{max}}=0.10$ and $\eta=3 \times 10^{-5}$, gives slightly lower peak reward than $w_{\text{sd}}^{\text{max}}=0.05$ but substantially tighter KL and smoother gradients, hence used in all exps.

5 Conclusion

Masked diffusion language models expose a denoising trace that standard RL post-training largely ignores. CAPR turns this trace into a training signal. It records confidence, entropy, and token stability in a compact path state, uses that state to steer reverse denoising steps, reuses cached prefixes to create cheap sibling continuations, and trains a Block Critic to convert one final outcome reward into per-block PPO weights. In this way, CAPR preserves the locality of tree-style comparisons without expanding a full search tree.

Across dense and mixture-of-experts LLaDA backbones, CAPR improves reasoning under the same decoding setting as the baselines. At the matched 256-token budget, it is best or tied-best on all reported tasks, with the largest gains on Sudoku and Countdown, where flat rollout-level signals are weakest. The 512-token results show that the same pattern holds at longer generation

lengths. On Sudoku, CAPR matches the strongest tree-structured baseline while using less than one third of its per-step wall-clock time. The ablations show that the gains do not come from branching alone. Branching and pruning add useful diversity, the Block Critic provides the largest intermediate improvement by localizing the reward signal, self-distillation keeps solved groups useful after group-relative advantages vanish, and the chi-square regularizer limits unstable ratio spikes. CAPR therefore offers a practical middle ground for dLLM-RL, with finer reward assignment than flat rollouts at substantially lower cost than full tree expansion.

Limitations

CAPR is evaluated on automatically verifiable reasoning tasks. This is a controlled setting for studying reward assignment, but it does not establish performance for open-ended instruction following, factuality, dialogue safety, or preference-based objectives where rewards are noisier and less local. The experiments also use zero-shot pass@1 decoding, pass@ k evaluation, robustness under different sampling settings, and out-of-domain prompts remain important tests of generalization.

The block critic is learned online from the same rollouts used to update the policy. This keeps CAPR lightweight, but the value estimates may be less reliable for very long reasoning chains, delayed subgoals, or domains where many partial trajectories can lead to the same final reward. CAPR is also tested only on LLaDA-family masked diffusion language models. Other dLLM architectures may expose different trace statistics, and branch fraction, block size, self-distillation weight, and value-head calibration may need retuning.

Finally, the compute measurements reflect our software stack and $8 \times A100$ -80GB hardware. CAPR reduces denoising continuations relative to tree expansion, but it still requires RL fine-tuning of large language models. Memory layout, kernel efficiency, and distributed-training overhead can change wall-clock cost on other systems.

Ethical Considerations

CAPR is a training algorithm for improving reasoning in masked diffusion language models. The experiments use public reasoning benchmarks with automatically checkable rewards, and we do not introduce new user data or sensitive annotations. However, stronger reasoning models can be mis-

used in the same ways as other capable language models, including producing misleading explanations, assisting harmful workflows, or making incorrect answers appear more persuasive. CAPR does not add a safety-alignment layer, so user-facing deployments should retain the safeguards, monitoring, refusal behaviour, and domain-specific evaluations required for the underlying model.

RL fine-tuning large models also has an environmental cost. CAPR is designed to reduce training cost relative to tree-structured rollouts, and we report hardware and wall-clock measurements to make that cost visible. Future work should continue to report compute transparently and study whether trajectory-aware signals can be reused across tasks, seeds, or training runs.

We downloaded models from huggingface. GSAI-ML/LLaDA-8B-Instruct¹ is available under MIT license. inclusionAI/LLaDA-MoE-7B-A1B-Instruct² is available under Apache license 2.0. The models were used as per their intended use.

References

- Shuang Cheng, Yihan Bian, Dawei Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. 2025. *Sdar: A synergistic diffusion-autoregression paradigm for scalable sequence generation*. *Preprint*, arXiv:2510.06303.
- Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. *Learning phrase representations using RNN encoder-decoder for statistical machine translation*. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. *Training verifiers to solve math word problems*. *Preprint*, arXiv:2110.14168.
- Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, Zhiyuan Liu, Hao Peng, Lei Bai, Wanli Ouyang, Yu Cheng, Bowen Zhou, and Ning Ding. 2025. *The entropy mechanism of reinforcement learning for reasoning language models*. *Preprint*, arXiv:2505.22617.

¹<https://huggingface.co/GSAI-ML/LLaDA-8B-Instruct>

²<https://huggingface.co/inclusionAI/LLaDA-MoE-7B-A1B-Instruct>

- Haoyu He, Katrin Renz, Yong Cao, and Andreas Geiger. 2025. [Mdp0: Overcoming the training-inference divide of masked diffusion language models](#). *Preprint*, arXiv:2508.13148.
- Audrey Huang, Wenhao Zhan, Tengyang Xie, Jason D. Lee, Wen Sun, Akshay Krishnamurthy, and Dylan J. Foster. 2025a. [Correcting the myths of kl-regularization: Direct alignment without overoptimization via chi-squared preference optimization](#). *Preprint*, arXiv:2407.13399.
- Zemin Huang, Zhiyang Chen, Zijun Wang, Tiancheng Li, and Guo-Jun Qi. 2025b. [Reinforcing the diffusion chain of lateral thought with diffusion language models](#). *Preprint*, arXiv:2505.10446.
- Sham M. Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, pages 267–274. Morgan Kaufmann.
- 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. 2025. [Mercury: Ultra-fast language models based on diffusion](#). *Preprint*, arXiv:2506.17298.
- Xianzhi Li, Ethan Callanan, Abdellah Ghassel, and Xiaodan Zhu. 2026. [Entropy-gated branching for efficient test-time reasoning](#). *Preprint*, arXiv:2503.21961.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. [Let’s verify step by step](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. [Continuous control with deep reinforcement learning](#). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.
- Nianyi Lin, Jiajie Zhang, Lei Hou, and Juanzi Li. 2025. [Boundary-guided policy optimization for memory-efficient rl of diffusion large language models](#). *Preprint*, arXiv:2510.11683.
- Shen Nie, Fengqi Zhu, Chao Du, Tianyu Pang, Qian Liu, Guangtao Zeng, Min Lin, and Chongxuan Li. 2025a. [Scaling up masked diffusion models on text](#). *Preprint*, arXiv:2410.18514.
- Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025b. [Large language diffusion models](#). *Preprint*, arXiv:2502.09992.
- Jingyang Ou, Jiaqi Han, Minkai Xu, Shaoxuan Xu, Jianwen Xie, Stefano Ermon, Yi Wu, and Chongxuan Li. 2025. [Principled rl for diffusion llms emerges from a sequence-level perspective](#). *Preprint*, arXiv:2512.03759.
- Leyi Pan, Shuchang Tao, Yunpeng Zhai, Zheyu Fu, Liancheng Fang, Minghua He, Lingzhe Zhang, Zhaoyang Liu, Bolin Ding, Aiwei Liu, and Lijie Wen. 2026. [d-treerpo: Towards more reliable policy optimization for diffusion language models](#). *Preprint*, arXiv:2512.09675.
- Kevin Rojas, Jiahe Lin, Kashif Rasul, Anderson Schneider, Yuriy Nevmyvaka, Molei Tao, and Wei Deng. 2026. [Improving reasoning for diffusion language models via group diffusion policy optimization](#). *Preprint*, arXiv:2510.08554.
- Subham S. Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. 2024. [Simple and effective masked diffusion language models](#). In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. [High-dimensional continuous control using generalized advantage estimation](#). In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. [Proximal policy optimization algorithms](#). *Preprint*, arXiv:1707.06347.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *Preprint*, arXiv:2402.03300.
- Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis K. Titsias. 2024. [Simplified and generalized masked diffusion for discrete data](#). In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.
- Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, and 3 others. 2025. [Seed diffusion: A large-scale diffusion language model with high-speed inference](#). *Preprint*, arXiv:2508.02193.

- Richard S. Sutton. 1988. [Learning to predict by the methods of temporal differences](#). *Mach. Learn.*, 3:9–44.
- Hongze Tan, Zihan Wang, Jianfei Pan, Jinghao Lin, Hao Wang, Yifan Wu, Tao Chen, Zhihang Zheng, Zhihao Tang, and Haihua Yang. 2026. [Gtpo and grpo-s: Token and sequence-level reward shaping with policy entropy](#). *Preprint*, arXiv:2508.04349.
- Xiaohang Tang, Rares Dolga, Sangwoong Yoon, and Ilija Bogunovic. 2026. [wd1: Weighted policy optimization for reasoning in diffusion language models](#). *Preprint*, arXiv:2507.08838.
- Emanuel Todorov. 2006. [Linearly-solvable markov decision problems](#). In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 1369–1376. MIT Press.
- Guanghan Wang, Gilad Turok, Yair Schiff, Marianne Arriola, and Volodymyr Kuleshov. 2026. [d2: Improved techniques for training reasoning diffusion language models](#). *Preprint*, arXiv:2509.21474.
- Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. 2025. [Revolutionizing reinforcement learning framework for diffusion large language models](#). *Preprint*, arXiv:2509.06949.
- Shaoan Xie, Lingjing Kong, Xiangchen Song, Xinshuai Dong, Guangyi Chen, Eric P. Xing, and Kun Zhang. 2026. [Advancing reasoning in diffusion language models with denoising process rewards](#). *Preprint*, arXiv:2510.01544.
- Jingyi Yang, Guanxu Chen, Xuhao Hu, and Jing Shao. 2025a. [Taming masked diffusion language models via consistency trajectory reinforcement learning with fewer decoding step](#). *Preprint*, arXiv:2509.23924.
- Kai Yang, Xin Xu, Yangkun Chen, Weijie Liu, Jiafei Lyu, Zichuan Lin, Deheng Ye, and Saiyong Yang. 2026. [Entropic: Towards stable long-term training of llms via entropy stabilization with proportional-integral control](#). *Preprint*, arXiv:2511.15248.
- Ling Yang, Ye Tian, Bowen Li, Xinchun Zhang, Ke Shen, Yunhai Tong, and Mengdi Wang. 2025b. [Mmada: Multimodal large diffusion language models](#). *Preprint*, arXiv:2505.15809.
- Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2025a. [Dream 7b: Diffusion large language models](#). *Preprint*, arXiv:2508.15487.
- Jiasheng Ye, Zaixiang Zheng, Yu Bao, Lihua Qian, and Quanquan Gu. 2025b. [Diffusion language models can perform many tasks with scaling and instruction-finetuning](#). *Preprint*, arXiv:2308.12219.
- Qiyong Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, and 16 others. 2025. [Dapo: An open-source llm reinforcement learning system at scale](#). *Preprint*, arXiv:2503.14476.
- Hanning Zhang, Pengcheng Wang, Shizhe Diao, Yong Lin, Rui Pan, Hanze Dong, Dylan Zhang, Pavlo Molchanov, and Tong Zhang. 2025. [Entropy-regularized process reward model](#). *Preprint*, arXiv:2412.11006.
- Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. 2025a. [d1: Scaling reasoning in diffusion large language models via reinforcement learning](#). *Preprint*, arXiv:2504.12216.
- Siyan Zhao, Mengchen Liu, Jing Huang, Miao Liu, Chenyu Wang, Bo Liu, Yuandong Tian, Guan Pang, Sean Bell, Aditya Grover, and Feiyu Chen. 2025b. [Inpainting-guided policy optimization for diffusion large language models](#). *Preprint*, arXiv:2509.10396.
- Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. 2025. [Group sequence policy optimization](#). *Preprint*, arXiv:2507.18071.
- Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. 2025a. [Llada 1.5: Variance-reduced preference optimization for large language diffusion models](#). *Preprint*, arXiv:2505.19223.
- Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tiejuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, and 7 others. 2025b. [Llada-moe: A sparse moe diffusion language model](#). *Preprint*, arXiv:2509.24389.

A CAPR Optimisation Iteration

Algorithm 1 states the full per-prompt update used in our experiments. The same loop runs unchanged across the four benchmarks. Only the schedules and weights listed in Table 7 differ. Symbols follow §3: G is the target group size, f_b the branch fraction, G_b the number of branches per cached prefix, B the number of blocks, τ_p the Polyak rate of the slow value head $V_{\bar{\phi}}$, β_{TD} the TD blend, and τ_{sd} the self-distillation reward threshold.

A.1 Path-State Update Details

For masked position j at denoising state z_t , let $p_j^{(t)} = \pi_{\theta}(\cdot | z_t)_j$ and $v_j^{(t)} = \arg \max_v p_j^{(t)}(v)$. Let $\bar{v}_j^{(t-1)}$ be the token prediction carried from the previous denoising update. CAPR forms the same path-state features used in §3.2:

$$\begin{aligned} c_j^{(t)} &= p_j^{(t)}(v_j^{(t)}), & s_j^{(t)} &= \mathbb{1}[v_j^{(t)} = \bar{v}_j^{(t-1)}], \\ e_j^{(t)} &= \frac{H(p_j^{(t)})}{\max_k H(p_k^{(t)})}, & a_j^{(t)} &= c_j^{(t)}(1 - e_j^{(t)}). \end{aligned} \quad (7)$$

Here $c_j^{(t)}$ is confidence, $s_j^{(t)}$ is stability, $e_j^{(t)}$ is normalized entropy, and $a_j^{(t)}$ is certainty. The Path-GRU update in Eq. 1 is implemented as

$$\begin{aligned} r_j^{(t)} &= \sigma(\alpha_e(0.5 - e_j^{(t)})), \\ u_j^{(t)} &= \sigma\left(\alpha_s\left(\frac{s_j^{(t)} + a_j^{(t)}}{2} - 0.5\right)\right), \\ m_j^{(t)} &= \frac{1}{4}(2c_j^{(t)} - 1)(1 + a_j^{(t)})(1 + s_j^{(t)}), \\ \tilde{h}_j^{(t)} &= \tanh\left(r_j^{(t)}h_j^{(t-1)} + (1 - r_j^{(t)})m_j^{(t)}\right), \\ h_j^{(t)} &= \text{RMSNorm}\left(u_j^{(t)}h_j^{(t-1)} + (1 - u_j^{(t)})\tilde{h}_j^{(t)}\right), \end{aligned} \quad (8)$$

where $r_j^{(t)}$ is the reset gate and $u_j^{(t)}$ is the update gate, α_e and α_s are constants. The final RMS normalization is applied across positions in the sequence and prevents prompt-dependent hidden-scale drift.

A.2 Block Critic Features and TD Blending

We use $b \in \{1, \dots, B\}$ for block index and reserve t for denoising time. At block boundary b , the value head receives a compact state with both scalar features and a pooled token-embedding feature:

$$\psi_b = (\mu(h_b), \sigma(h_b), \rho_b^{\text{mask}}, b/(B-1), \bar{R}_q, \sigma_R(q), e_b), \quad (9)$$

where $\mu(h_b)$ and $\sigma(h_b)$ are the mean and standard deviation of the path state inside the current block, ρ_b^{mask} is the fraction of positions in that block that

remain masked, $(\bar{R}_q, \sigma_R(q))$ are the mean and standard deviation of outcome rewards among rollouts for prompt q , and e_b is the average input embedding of the tokens that have already been placed in block b .

For rollout i with final outcome reward R_i , let $n_{i,b}$ be the number of tokens unmasked in block b , i.e., positions that switch from [MASK] to a visible token while block b is being denoised. CAPR forms reward-conserving block rewards

$$r_{i,b} = \omega_{\text{blk}} R_i \frac{n_{i,b}}{\sum_{c=1}^B n_{i,c}} + (1 - \omega_{\text{blk}}) R_i \mathbb{1}[b = B]. \quad (10)$$

The first term gives dense block supervision proportional to the tokens revealed in each block, while the second term keeps the terminal outcome signal on the final block. This mattered in runs that reach EOS before block B , instead of $1/B$ it will be accurate with $n_{i,b}/\sum_c n_{i,c}$, where $n_{i,b}$ is the number of tokens latest unmasked in block b .

The target head $V_{\bar{\phi}}$ computes TD residuals and GAE advantages on the block-boundary chain (Schulman et al., 2016; Sutton, 1988):

$$\begin{aligned} \delta_{i,b} &= r_{i,b} + \gamma V_{\bar{\phi}}(s_{i,b+1}) - V_{\bar{\phi}}(s_{i,b}), \\ \hat{A}_{i,b}^{\lambda} &= \sum_{\ell=0}^{B-b} (\gamma \lambda_{\text{gae}})^{\ell} \delta_{i,b+\ell}, \end{aligned} \quad (11)$$

with $V_{\bar{\phi}}(s_{i,B+1}) = 0$. The value-head regression target is the corresponding λ -return:

$$y_{i,b}^V = \text{sg}\left(\hat{A}_{i,b}^{\lambda} + V_{\bar{\phi}}(s_{i,b})\right), \quad (12)$$

where sg denotes stop-gradient. The online value head V_{ϕ} is trained to predict $y_{i,b}^V$, while $V_{\bar{\phi}}$ is updated as a slow Polyak average (Lillicrap et al., 2016).

The block-level signal is blended into PPO only when the value head is calibrated. Let $\bar{\mathcal{L}}_V$ be the recent value-head loss and b_V be a running baseline tracking the best smoothed value loss so far. CAPR uses

$$q_V = \frac{b_V^2}{b_V^2 + \bar{\mathcal{L}}_V^2}, \quad \beta_{\text{TD}}^{\text{eff}} = \beta_{\text{TD}} q_V. \quad (13)$$

When the value head loss rises far above its baseline, q_V becomes small and CAPR falls back toward the flat GRPO signal. As the value head recovers, the block weighting reopens automatically.

A.3 Self-Distillation Loss

CAPR uses self-distillation only during training. Let \mathcal{A}_i be the active completion tokens of rollout i , i.e., the valid completion tokens selected by the current PPO mask. In our reported runs all tokens in \mathcal{A}_i are averaged uniformly, so no additional token weighting is used.

First, solved rollouts supervise themselves with a small next-token loss. Let $\mathcal{C} = \{i : R_i \geq \tau_{\text{sd}}\}$ be the solved rollouts and let $\rho_{\text{cor}} = |\mathcal{C}|/(QG)$ be their fraction in a batch of Q prompts. CAPR uses

$$\mathcal{L}_{\text{NLL}} = -w_{\text{sd}}^{\text{eff}} \frac{\sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{A}_i} \log \pi_{\theta}(y_{ij})}{\sum_{i \in \mathcal{C}} |\mathcal{A}_i|}. \quad (14)$$

The effective self-distillation weight grows with the solved-rollout fraction and is bounded by the tuned maximum,

$$w_{\text{sd}}^{\text{eff}} = \min(w_{\text{sd}} \rho_{\text{cor}}^p, w_{\text{sd}}^{\text{max}}), \quad (15)$$

so it is near zero when solved rollouts are rare, increases as correct samples become common, and never exceeds $w_{\text{sd}}^{\text{max}}$. In Table 5, we vary this maximum while keeping the remaining self-distillation settings fixed.

A.4 Chi-Square Regulariser

Sparse-reward updates can fail because of a few tokens, not because the whole completion moves too far. In a solved rollout, PPO may strongly increase the probability of a small number of already-likely tokens. These upper-tail log-ratio spikes can dominate the gradient, especially when most other rollouts have zero or identical reward. Standard PPO clipping and KL regularisation (Kakade and Langford, 2002; Schulman et al., 2017) control average policy movement, but they do not specifically target this token-level upper tail.

CAPR therefore adds a small outlier-only χ^2 term. It is not a second full KL penalty: ordinary movement is still handled by PPO and KL. The extra term activates only when a token’s centred log-ratio is unusually large and positive. This follows the robustness motivation of chi-squared preference optimisation (Huang et al., 2025a) and is related in spirit to asymmetric policy-gradient controls that treat positive and negative update directions differently (Yu et al., 2025; Xie et al., 2026). For each sequence, CAPR computes $d_j = \log \pi_{\theta}(y_j) - \log \pi_{\text{ref}}(y_j)$ on the active completion-

token set \mathcal{A} and normalises it by

$$\begin{aligned} \bar{d} &= |\mathcal{A}|^{-1} \sum_{k \in \mathcal{A}} d_k, \\ s_d &= \left(|\mathcal{A}|^{-1} \sum_{k \in \mathcal{A}} (d_k - \bar{d})^2 \right)^{1/2}, \\ \hat{d}_j &= (d_j - \bar{d}) / (s_d + \epsilon). \end{aligned} \quad (16)$$

Only the upper tail is penalized:

$$\mathcal{L}_{\chi^2} = |\mathcal{A}|^{-1} \sum_{j \in \mathcal{A}} \left[\hat{d}_j - 1 \right]_+^2. \quad (17)$$

The one-sided form is important. Positive spikes make the new policy over-commit to a few sampled tokens. Negative movements usually reflect probability mass being moved away from sampled tokens and are already bounded by the clipped PPO objective. Penalising both sides equally would make the regulariser behave like another broad KL term and would unnecessarily resist useful exploration.

A simple trust-region view gives the same intuition. For bounded token advantages, the performance-difference bound contains an error term proportional to how far the new policy moves from the old one (Kakade and Langford, 2002). Since total variation is controlled by chi-squared divergence, keeping the upper-tail χ^2 small bounds the harmful part of the update:

$$J(\pi_{\theta'}) - J(\pi_{\theta}) \gtrsim \hat{\Delta}_{\text{PPO}} - C \sqrt{\mathcal{L}_{\chi^2}}, \quad (18)$$

for a constant C depending on the advantage scale. CAPR uses Eq. 17 only as a soft penalty, so the method keeps the usual PPO/KL update while damping the rare token-level jumps most likely to destabilise sparse-reward training.

B Denoising Forward-Pass Cost

We compare rollout-generation cost by counting denoising model forward passes for one prompt. This is the shared cost paid before the PPO-style update. For shared quantities we follow the d -TreeRPO comparison setting (Pan et al., 2026): completion length $L=256$, and denoising budget $N=256$. CAPR-specific quantities use our reported values in Appendix C: branch fraction $f_b=0.5$ and branches per cached prefix $G_b=2$, $G=8$ terminal completions.

Flat methods sample each completion independently. With G completions and N denoising steps per completion, the cost is

$$C_{\text{flat}} = GN. \quad (19)$$

Algorithm 1 One CAPR optimisation iteration for prompt q .

Require: Policy π_θ , reference π_{ref} , value head V_ϕ and target head $V_{\bar{\phi}}$.
Require: Rollouts G , branches per cached prefix G_b , fork fraction f_b , denoising steps N , blocks B .
1: Set $M = \lceil G/G_b \rceil$ cached prefixes and initialise path state $h = \mathbf{0}$.
2: **for** $m = 1, \dots, M$ **do**
3: Denoise from z_N to the fork step $f_b N$, updating h from confidence, entropy, and stability.
4: Cache the fork-step state and spawn G_b sibling continuations by resampling the cached scores.
5: Continue the siblings to completion with the same trace update and steering rule.
6: At scheduled block boundaries, prune low-quality siblings using mean path-state quality.
7: **end for**
8: Score the completed rollouts with the verifier to obtain outcome rewards R_i .
9: **for** each rollout i and block b **do**
10: Form the reward-conserving block signal
 $r_{i,b} = \omega_{\text{blk}} R_i \frac{n_{i,b}}{\sum_c n_{i,c}} + (1 - \omega_{\text{blk}}) R_i \mathbb{1}[b = B]$.
11: Compute TD residuals, GAE advantages $\hat{A}_{i,b}$, and value targets $y_{i,b}^V$ using Eq. 11 and Eq. 12.
12: **end for**
13: Convert block advantages into token weights with Eq. 5.
14: Optimise the policy and value head with $\mathcal{L}_{\text{CAPR}}$ in Eq. 6, including KL, χ^2 , and self-distillation terms.
15: Update the target head by Polyak averaging: $\bar{\phi} \leftarrow (1 - \tau_p)\bar{\phi} + \tau_p\phi$.

Thus the flat baseline costs $8 \cdot 256 = 2048$ forwards.

For d -TreeRPO, we use the forward-count formula reported in Pan et al. (2026). With tree branching factor B_{tree} and depth H , the reported cost is

$$C_{\text{tree}} = \frac{B_{\text{tree}}(B_{\text{tree}}^H - 1)}{B_{\text{tree}} - 1} \cdot \frac{N}{H}. \quad (20)$$

With $B_{\text{tree}}=4$, $H=2$, (these settings are the minimal requirements as per (Pan et al., 2026)) and $N=256$, this gives $(4 + 16) \cdot 128 = 2560$ forwards.

CAPR uses a smaller comparison set by sharing the early prefix. One branch group runs a shared prefix for $f_b N$ steps, then forks G_b sibling suffixes of length $(1 - f_b)N$. Without pruning, one group costs

$$C_{\text{group}} = [f_b + G_b(1 - f_b)]N. \quad (21)$$

Producing G completions requires G/G_b groups, giving

$$C_{\text{CAPR}} = \frac{G}{G_b} [f_b + G_b(1 - f_b)]N. \quad (22)$$

With $G=8$, $G_b=2$, $f_b=0.5$, and $N=256$, the no-pruning count is $4 \cdot 1.5 \cdot 256 = 1536$ forwards. Scheduled pruning can only reduce this value. If

$0 = p_0 < p_1 < \dots < p_P < p_{P+1} = 1$ are suffix checkpoints and a_m siblings are kept on interval $[p_m, p_{m+1}]$, then one group costs

$$N \left[f_b + (1 - f_b) \sum_{m=0}^P a_m (p_{m+1} - p_m) \right], \quad (23)$$

which is no larger than the no-pruning value whenever $a_m \leq G_b$. Table 6 therefore reports CAPR’s conservative upper bound.

Table 6: Per-prompt denoising forward-pass cost under the d -TreeRPO comparison setting. Shared quantities use, $L=256$, and $N=256$. CAPR-specific quantities use $G_b=2$ and $f_b=0.5$, $G=8$.

Method	Forwards per prompt	# fwd
Diffu-GRPO	GN	2048
d -TreeRPO	$\frac{B_{\text{tree}}(B_{\text{tree}}^H - 1)}{B_{\text{tree}} - 1} \cdot \frac{N}{H}$	2560
CAPR	$\leq \frac{G}{G_b} \cdot [f_b + G_b(1 - f_b)]N$	≤ 1536

C Hyperparameters

Table 7 reports the CAPR hyperparameters used for the results in Table 1. Unless noted otherwise, the values are shared across tasks.

D CAPR Search Efficiency

This appendix explains CAPR’s compute advantage. The goal is not to prove a new convergence theorem, but to clarify why a block-level value head can recover much of the attribution signal that tree rollouts obtain by spending additional forward passes.

D.1 Tree Search reduces Variance by spending Forwards

Tree methods branch a partial denoising path, finish many continuations, and compare their final rewards. If a node has B_{mtree}^H leaves, the reward average at that node has variance roughly

$$\text{Var}(\hat{R}_{\text{tree}} | s_t) \approx \frac{\sigma_{R|s_t}^2}{B_{\text{tree}}^H}, \quad (24)$$

where $\sigma_{R|s_t}^2$ is the outcome-reward variance after state s_t . This is useful, but the variance reduction is purchased by evaluating many continuations.

D.2 CAPR learns the same Signal Once

CAPR keeps only a small number of continuations, but it records the path state at every block boundary. For block boundary state s_b , the value head learns

$$V_\phi(s_b) \approx \mathbb{E}[R | s_b], \quad (25)$$

Table 7: CAPR hyperparameters, organised by mechanism.

Group	Parameter	Value
<i>Generation & rollouts</i>		
	target rollouts G	8
	block length ℓ_{blk}	32 tokens
	number of blocks B	L/ℓ_{blk}
	denoising steps N	$L/2$
	evaluation temperature	0 / pass@1 decoding
	rollout temperature	1.0
<i>Cache & Steer path state (§3.2)</i>		
	α_e (entropy gate scale)	4.0
	α_s (stability gate scale)	5.0
	γ (path-state injection)	0.15
	λ (path-state steering scale)	1.0
	λ_f (χ^2 coefficient)	0.01
	β (reference KL coefficient)	0.01
<i>Branch-and-prune sampler (§3.3)</i>		
	branch fraction f_b	0.5
	branches per cached prefix G_b	2
	first prune checkpoint p_1	0.33
	second prune checkpoint p_2	0.67
	keep ratio	0.75
<i>Block Critic (§3.4)</i>		
	TD discount γ	0.99
	GAE λ_{gae}	0.95
	value-MSE coefficient c_v	0.5
	value-head hidden dim	128
	Polyak coefficient τ_p	0.005
	replay-buffer size	32
	block-shaping mass ω_{blk}	0.20
<i>Self-distillation loss (Appendix A.3)</i>		
	w_{sd} (base NLL scale)	0.10
	$w_{\text{sd}}^{\text{max}}$ (maximum NLL weight)	0.10
	τ_{sd} (reward threshold)	0.99
	power p	2.0
	τ_{NLL} (NLL clamp)	3.0
<i>Optimisation</i>		
	learning rate η	3×10^{-5}
	optimiser	AdamW
	weight decay	0.1
	gradient clip	1.0
	PPO inner epochs μ	2
	PPO clip ϵ	0.4
	LoRA rank (r, α , dropout)	(128, 64, 0.05)
	DeepSpeed stage	ZeRO-2
	precision	bf16 (fp32 master)

so it can estimate which blocks made the final reward more likely before seeing many extra leaves. The TD residual

$$\delta_b = r_b + \gamma V_{\bar{\phi}}(s_{b+1}) - V_{\bar{\phi}}(s_b) \quad (26)$$

turns one final reward into one block-level signal. This is the same role that a tree plays: it tells the optimiser which parts of a path were useful. The difference is that CAPR learns this mapping and reuses it, instead of rebuilding a fresh tree for every prompt.

D.3 Why this is Stable

Two safeguards keep the learned signal from dominating when it is unreliable. First, the target value head $V_{\bar{\phi}}$ is a slow Polyak average of the online head, which smooths noisy block targets. Second, CAPR gates the TD blend using the recent value loss: when the value head is poorly calibrated, $\beta_{\text{TD}}^{\text{eff}}$ becomes small and the update falls back toward the flat GRPO advantage.

The self-distillation loss and chi-square regulariser are local to the policy update and are defined in Appendix A.3 and Appendix A.4. Self-distillation copies solved rollouts back into the LoRA policy, which is why the value head and search machinery can be removed at inference.

D.4 Implication of the Bound

CAPR trades tree depth for a learned block predictor. When the value head explains a nonzero part of the final-reward variance, the resulting block advantage has lower noise than a flat rollout advantage at nearly the same generation cost. When the value head is weak, the calibration gate reduces its influence, so CAPR reverts toward the flat baseline rather than relying on an inaccurate local signal.

This yields the following practical comparison:

tree search : lower variance by more leaves,

CAPR : lower variance by learned block values. (27)

The empirical diagnostics in §4.4 check this prediction. The value loss decreases, the χ^2 statistic stays bounded, and the full CAPR run has the smoothest gradient-norm trajectory.

E Detailed Descriptions of dLLM RL Baselines

This appendix summarises the baselines in Table 1. For each method, we describe its training signal, the advantage assigned to token updates, and the main approximation used for masked diffusion likelihoods. The goal is to make the comparison with CAPR interpretable rather than to reproduce the original derivations.

Diffu-GRPO (Zhao et al., 2025a). A direct port of GRPO to masked dLLMs. The reward is the verifier outcome on the final completion o_i , and the advantage $A_{i,k} = r_i - \text{mean}\{r_j\}_{j=1}^G$ is broadcast to every token of o_i . The only dLLM-specific piece is the log-probability estimator: rather than marginalising over denoising orders, Diffu-GRPO evalu-

ates $\log \pi_\theta$ in a single forward pass on a randomly masked prompt and a fully masked completion, and re-samples the prompt mask each inner update for regularisation. This makes training cheap but gives every token the same rollout-level signal. CAPR instead uses the per-block advantages of §3.4.

VRPO (Zhu et al., 2025a). A DPO-style preference optimiser using offline pairs (q, o^w, o^l) and no explicit reward model. Since $\log \pi_\theta(o | q)$ is intractable, VRPO substitutes an ELBO bound $B_\pi(o | q) \leq \log \pi(o | q)$ and forms the DPO score from differences of $B_{\pi_\theta} - B_{\pi_{\text{ref}}}$ on o^w and o^l . Two ideas reduce the variance of this estimator: an optimal allocation of the sampling budget across diffusion timesteps (more n_t , fewer n_{z_t}) and anti-thetic sampling that shares noise between the live and reference models. The remaining gap is structural: the score is built on a lower bound, not the true log-likelihood. VRPO is the recipe behind the public LLaDA-1.5 release on LLaDA-8B-Instruct.

wd1 (Tang et al., 2026). A ratio-free alternative to GRPO. After scoring a group with the verifier, wd1 maps the group-relative advantages $A_i = R_i - \text{mean}(R_{1:G})$ to softmax weights $w_i^\pm = \text{softmax}(\pm \psi A_i)$ and optimises $\sum_i (-w_i^+ + w_i^-) \log \pi_\theta(o_i | q)$, sidestepping the dLLM ratio entirely. The token-level log-probabilities are still approximated by the mean-field, single-pass estimator of Diffu-GRPO, with the same rollout-level signal for every token.

SAPO (Xie et al., 2026). An augmented Diffu-GRPO with a step-aware reward. SAPO estimates whether an intermediate diffusion state makes progress by comparing the expected outcome of rollouts continued from two intermediate states. In practice, the cheaper variant continues from one intermediate state and the fully masked state. The combined signal $A_i^{\text{total}} = A_i + \mathbf{1}[A_i > 0] R_{\text{process}}$ is again broadcast to all tokens, so the per-step granularity ends at the rollout level. Like Diffu-GRPO, SAPO inherits the mean-field likelihood approximation under randomly masked conditioning.

GDPO (Rojas et al., 2026). A GRPO variant that keeps the broadcast advantage but lifts the importance ratio from token-level (mean-field) to sequence-level via an ELBO surrogate, $r_i = L_{\text{ELBO}}(o_i | q, \theta) / L_{\text{ELBO}}(o_i | q, \theta_{\text{old}})$. Variance is controlled by replacing the outer Monte Carlo over diffusion time with a small set of deterministic quadrature points (typically $N=2$ or 3) and a

lightweight inner Monte Carlo over masked states. As with VRPO, the optimisation runs on a bound rather than on the true log-likelihood. We include GDPO only in the LLaDA-MoE-7B-A1B-Instruct section of Table 1 because Pan et al. (2026) report it only on that backbone under their shared evaluation setting.

TraceRL (Wang et al., 2025). A trajectory-aware PPO that optimises along the model’s actual decoding path rather than under random masking. For each prompt, a rollout exposes a chain of intermediate states (z_1, \dots, z_0) . For a token o_j revealed at the transition $z_t \rightarrow z_{t-1}$, the importance ratio is computed on $\pi(\cdot | z_t)$. A diffusion-based value model V_{θ_v} then produces per-token GAE-style advantages, giving a fine-grained learning signal. The trade-off is the standard one: a learned value model is sensitive to value mis-specification and reward hacking, and the trajectory-conditioned likelihoods do not correct for the marginalisation over alternative decoding orders that lead to the same o .

d-TreeRPO (Pan et al., 2026). A tree-rollout baseline. Each prompt expands into a balanced tree of branching factor B and depth H . Every node represents an N/H -step diffusion segment, and the per-tree-node advantage is averaged bottom-up from leaf rewards. This gives the finest non-CAPR advantage signal in our comparison, but the cost grows as $\frac{B(B^H-1)}{B-1} \cdot \frac{N}{H}$ forwards per prompt. CAPR targets similar attribution granularity with a single flat rollout structure, with cost linear in G and independent of tree depth.

F Reward-Curve Comparison

Figure 9 compares CAPR training reward curves with d -TreeRPO and the open-source Diffu-GRPO (Zhao et al., 2025a) and wd1 (Tang et al., 2026) baselines. CAPR moves from the low-reward region into the same high-reward band as the tree method while using the cheaper cached/branched rollout structure. On GSM8K and Math500, the base model already has stronger task competence and the reward axis is much narrower, so CAPR stays close to the d -TreeRPO curve and may move slightly above or below it at different parts of training. This is the expected behaviour: on easier dense-backbone tasks, the main benefit is stable without paying for full tree expansion, rather than a large absolute change in training reward.

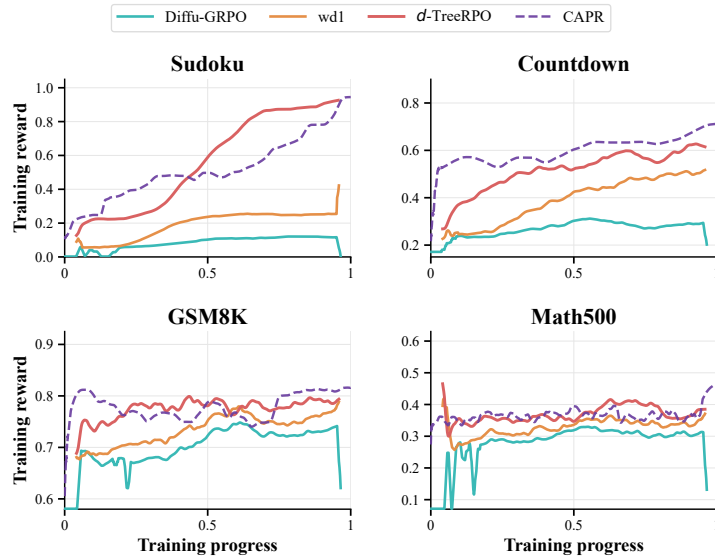


Figure 9: Training reward curves for CAPR compared with *d*-TreeRPO and open-source baselines Diffu-GRPO and wd1 across the four reasoning tasks.

G Training Details

G.1 Prompts

Prompts used in Sudoku, Countdown, GSM8k and Math500 are as follows:

Sudoku

Please solve the following 4x4 Sudoku puzzle. The puzzle is provided as a 16-character string reading left-to-right, top-to-bottom, where '0' represents empty cells.

Rules:

- Fill empty cells with digits 1-4
- Each row must contain digits 1-4 exactly once
- Each column must contain digits 1-4 exactly once
- Each 2x2 box must contain digits 1-4 exactly once

Important: Your solution must be a COMPLETE 16-character string with only the digits 1-4, representing your final solved grid.

Respond in this exact format:

```
<reasoning>
Your step-by-step solving process
</reasoning>
<answer>
[16-character solution string with no spaces
or separators]
</answer>
```

Solve the following Sudoku puzzle: {PUZZLE}

Countdown

Respond in the following format:

```
<reasoning>
...
</reasoning>
```

```
<answer>
...
</answer>
```

Using only the numbers {NUMBERS}, create an arithmetic expression that evaluates to exactly {TARGET}. You must use all numbers from the list, and each number must be used exactly once. You may use the operations +, -, *, and / as needed. After reasoning, provide only your final expression inside <answer></answer> tags without including an equals sign or the target number. For example, if the numbers are [2, 3, 4] and the target is 5, a valid answer is: <answer>2*4-3</answer>.

GSM8k

Respond in the following format:

```
<reasoning>
...
</reasoning>
<answer>
...
</answer>
```

You are a math expert. You will be given a question to solve. Solve it step by step. Put only the final answer (the number) in the <answer> </answer> tag without any other text.

{QUESTION}

Math500

Respond in the following format:

```
<reasoning>
...
</reasoning>
<answer>
...
</answer>
```

You are a math expert. You will be given a question to solve. Solve it step by step. Wrap the final answer in a `\boxed{}`.

{QUESTION}

G.2 Reward Functions

Sudoku. The reward is defined as the percentage of correctly filled previously empty cells, with a maximum value of 1 and a minimum value of 0.

Countdown. The reward is 1.0 if the provided expression achieves the target using the given numbers. If the numbers used are correct but the target is missed, the reward is 0.1. In all other cases, the reward is 0.

GSM8k. During training, we utilize a correctness-based reward, assigning a value of 1.0 if the provided answer matches the ground truth.

Math500. Similarly, during training, we employ a correctness-based reward system, assigning a reward of 1.0 if the given answer matches the ground truth.

G.3 Dataset Links

Following previous works (Zhao et al., 2025a; Tang et al., 2026), we adopt the 4×4 Sudoku training and test sets provided by Diffu-GRPO (Zhao et al., 2025a) for Sudoku. For Countdown, models are trained on the 3to4 Countdown task training set³ and evaluated using the test set provided by Diffu-GRPO. For GSM8K⁴ and Math500⁵, we strictly follow their official training-test splits for both training and evaluation. All datasets were used as per their intended use.

H Case Study

This section presents a case study comparing CAPR against its strongest baseline, d-TreeRPO, on the same GSM8K question, as shown in Figure 10. CAPR produces both a correct reasoning process and the correct final answer.

³<https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4>

⁴<https://huggingface.co/datasets/openai/gsm8k>

⁵train: <https://huggingface.co/datasets/ankner/math-500>, test: <https://huggingface.co/datasets/HuggingFaceH4/MATH-500>

LLaDA-8B-Instruct + CAPR

Question: Digimon had its 20th anniversary. When it came out John was twice as old as Jim. If John is 28 now how old is Jim?

Model Output:

<reasoning>

To determine Jim's current age, we need to follow these steps:

1. Identify John's age when Digimon came out.
2. Determine Jim's age when Digimon came out.
3. Calculate Jim's current age.

First, we know that Digimon had its 20th anniversary, so it came out 20 years ago. If John is currently 28 years old, then his age when Digimon came out was $28 - 20 = 8$ years old.

Next, we know that when Digimon came out, John was twice as old as Jim. Therefore, let's set up the equation: $8 = 2 \cdot \text{Jim's age}$. To find Jim's age, we divide both sides by 2: $\text{Jim's age} = 8/2 = 4$.

Now, we know Jim was 4 years old when Digimon came out. Since Digimon came out 20 years ago, **Jim's current age is $4 + 20 = 24$ years old.** ✓

</reasoning>

<answer>

24 ✓

</answer>

LLaDA-8B-Instruct + *d*-TreeRPO

Question: Digimon had its 20th anniversary. When it came out John was twice as old as Jim. If John is 28 now how old is Jim?

Model Output:

<reasoning>

To determine Jim's current age, we need to follow these steps:

1. Understand that Digimon had its 20th anniversary, 20 years ago.
2. At that time, John was twice as old as Jim.
3. We are currently given that John is 28 years old.

Let's denote Jim's current age as J .

Since Digimon had its 20th anniversary 20 years ago, John was $28 - 20 = 8$ years old at that time.

At that time, John was twice as old as Jim. Therefore, we can set up the equation: $8 = 2J$. To find Jim's current age, we solve for J : $J = \frac{8}{2} = 4$.

Thus, Jim is currently 4 years old. ✗

</reasoning>

<answer>

4 ✗ (ground truth: 24)

</answer>

Figure 10: A case study comparing LLaDA-8B-Instruct trained with CAPR (top) and *d*-TreeRPO (bottom) on the same GSM8K question. Both models correctly derive that Jim was 4 years old 20 years ago. CAPR carries the temporal offset through the final step ($4 + 20 = 24$), whereas *d*-TreeRPO conflates “Jim’s age then” with “Jim’s current age J ” and stops at 4.