ReST-RL: Reinforcing LLM Reasoning through Unified Self-Training and Value-Guided Search

Sining Zhoubian[*], **Dan Zhang**[†], **Jie Tang**[†]
Department of Computer Science and Technology, Tsinghua University
zbsn21@mails.tsinghua.edu.cn
https://github.com/THUDM/ReST-RL

Abstract

With respect to improving the reasoning accuracy of LLMs, the representative reinforcement learning (RL) method — Group Relative Policy Optimization (GRPO) — has achieved critical success, yet it still suffers from the issue of insignificant reward signals. This paper introduces **ReST-RL**, a unified Reinforced Self-Training (ReST) policy–value framework that reconnects policy optimization and value-guided search to improve LLM reasoning ability. Firstly, *ReST-GRPO* adopts an optimized ReST-style algorithm to reshape the policy-induced trajectory distribution by increasing the reward variance of GRPO sampling and exposing the policy to more informative partial states, thereby improving training efficiency and effectiveness. Then, we further introduce a decoding optimization method, *VM-MCTS*, which trains a Value Model (VM) from self-collected Monte-Carlo Tree Search (MCTS) targets and deploys it through an adapted MCTS algorithm to provide precise process signals and verification scores, further enhancing reasoning accuracy. These two stages are internally dependent — ReST-GRPO yields higher-quality trajectories for value learning with VM-MCTS, which in turn enables more effective inference-time search. We validate our framework on multiple coding benchmarks (e.g., APPS, BigCodeBench, and HumanEval), where it significantly outperforms other reinforcement training baselines (naive GRPO, DAPO, and ReST-DPO), as well as decoding and verification baselines (e.g., PRM-BoN and ORM-MCTS), indicating its power to strengthen LLM reasoning capability. Moreover, we further evaluate ReST-RL on out-of-domain math and science reasoning tasks, where it achieves improved performance without target-domain tuning and favorable end-to-end efficiency trade-offs, providing preliminary transfer evidence beyond our primary coding domain.

1 Introduction

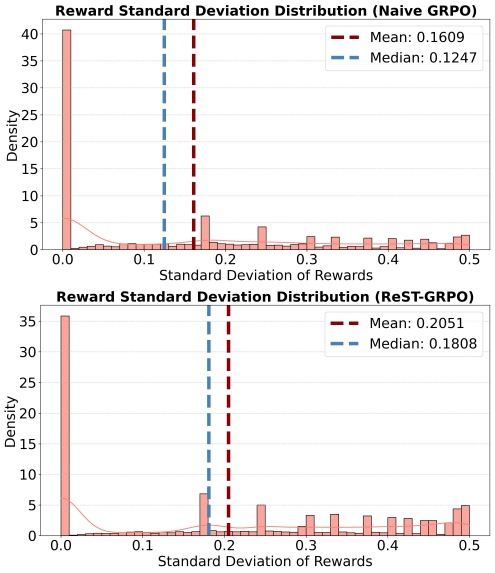
Recent advances in Large Language Models (LLMs) have made remarkable progress in reasoning tasks [20; 5; 42], yet substantial challenges remain for solving complex ones [6; 45]. Reinforcement Learning (RL) has emerged as a promising approach to further improve the reasoning ability of LLMs [18; 31; 24; 4; 22; 25; 26; 39], with growing attention on both online methods like Group Relative Policy Optimization (GRPO) [25] that data sampling and model updating take place simultaneously [24; 22; 25], and offline self-training paradigms like Reinforced Self-Training (ReST) [4; 39] that obtains training data through offline sampling and filtering mechanisms [4; 26; 39]. In addition, the reward models (RMs) are gradually attracting more attention due to their function in output verification, with outcome reward models (ORMs) that verify the final output of LLMs [29; 13] and

[*]Work done when interned at Z.ai.
[†]Corresponding: DZ and JT

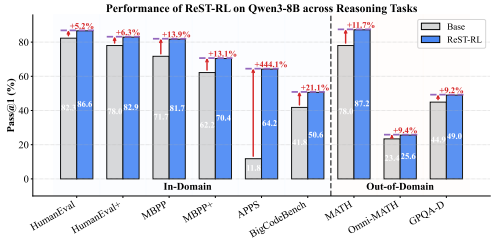
40th Conference on Neural Information Processing Systems (NeurIPS 2026).

process reward models (PRMs) that provide feedback for intermediate steps and indicate better verification accuracies than ORMs [13; 31; 42; 39], improving reasoning accuracy of LLMs. Moreover, integrating RMs with advanced decoding algorithms, such as Monte-Carlo Tree Search (MCTS), has shown further enhancements in LLM reasoning performance [39; 16].

However, there are still some shortcomings in these methods. ***(1) GRPO can provide low-information updates when all responses in a sampled group receive similar rewards [36; 40] (cf. Figure 1a).*** Dynamic sampling and step-wise reward designs mitigate this issue [36; 40], but may require additional online rollouts or reward design and still operate primarily at the original-prompt level. In particular, DAPO repeatedly samples until enough nonzero-variance groups are retained, whereas a prompt-level curriculum changes which questions are presented. Neither mechanism directly revisits informative intermediate states within a successful trajectory. ***(2) A key limitation of PRMs is their reliance on high-quality annotated data [13; 31], which is costly and labor-intensive to obtain, hindering scalability and reliability.*** To tackle this problem, some methods propose to estimate and collect process rewards through Monte-Carlo simulations [31; 39]. However, their reliance on task-specific answer matching and fixed notions of step correctness can limit direct application to richer reasoning settings such as coding [39]. We provide a more comprehensive review of these LLM reinforcement methods (see Appendix A.9). Overall, existing approaches expose a practical trade-off among data-collection cost, informative policy updates, and fine-grained inference-time guidance. This motivates a design that preserves GRPO's lightweight policy update while recovering informative state-level learning signals. Such a design should distinguish two roles that are often conflated: improving the policy's training distribution, and learning a value signal for allocating inference-time computation. Connecting these roles is useful only if the first-stage policy improvement survives a matched value-learning pipeline rather than merely producing a better policy-only checkpoint.



(a) Reward variance becomes more informative during ReST-GRPO.



(b) ReST-RL improves Qwen3-8B on code and provides transfer evidence on math and science tasks without target-domain tuning.

Figure 1: Motivation and overall results.

In this paper, we introduce ReST-RL, a unified, state-localized policy–value framework for Reinforced Self-Training, comprising ReST-GRPO and VM-MCTS (Figure 2). Rather than treating policy training and reward learning as two isolated enhancements, the core principle of ReST-RL is to reconnect them in a lightweight manner: ReST-GRPO reshapes the policy-induced trajectory distribution toward more informative high-reward regions, and this improved distribution in turn enables more reliable value estimation and more effective inference-time search in VM-MCTS. As the first stage, ReST-GRPO adopts ReST-style data filtering and assembly to improve GRPO on sparse-reward reasoning tasks. Selected prefixes are not treated as imitation targets: they become additional online-GRPO contexts from which the policy samples and optimizes fresh suffixes. This increases reward variation while retaining exploration over continuations. As the second stage, VM-MCTS trains a value model (VM) from self-collected MCTS targets without extra annotation. Under a fixed policy, the VM estimates $V^{\pi}(s) = \mathbb{E}_{\pi}[R \mid s]$ and is used both to allocate tree search and to select final outputs. In this way, ReST-RL aims to recover fine-grained value guidance for reasoning without returning to the full cost of online actor-critic learning. Unlike ReST-style methods that train on accepted complete responses [4; 26], ReST-GRPO uses accepted prefixes only as contexts and freshly samples their suffixes online. It also differs from prompt-level curricula and dynamic sampling [21; 36]: these methods change which original

prompts are presented, whereas ReST-GRPO additionally changes where within a trajectory online RL begins. Our primary evaluation covers coding benchmarks of varying difficulty, including APPS [6], BigCodeBench [45], and HumanEval [2]. We additionally study transfer without target-domain tuning on MATH [7], Omni-MATH [3], and GPQA-Diamond [23]. Table 1 shows improvements over ReST-DPO, GRPO, and DAPO, while Table 2 compares against ORM-, PRM-, and MCTS-based decoding. The controlled results in Table 3 further isolate policy components, matched value learning, and approximately latency-matched search. Together, these results support a coherent policy–value pipeline with favorable end-to-end compute trade-offs and cross-domain transfer evidence.

Contributions. First, we introduce a state-localized policy-training procedure that combines reward-based filtering with online GRPO restarts from selected partial states, rather than imitating complete accepted outputs. Second, we formulate the downstream verifier as a static-policy value model and use the same signal for search allocation and final selection. Third, we provide controlled evidence beyond the original end-to-end comparisons: full DAPO results, component and curriculum ablations, matched policy–VM pipelines, and approximately latency-matched VM-BoN. Our main empirical evidence remains code reasoning, supplemented by math and science evaluations without target-domain tuning.

2 The ReST-RL Method

To connect policy optimization and reward learning for LLM reasoning while addressing low-information group rewards and process-level guidance, we develop ReST-RL as a unified, state-localized policy–value framework for Reinforced Self-Training. It is composed of two main components, namely ReST-GRPO and VM-MCTS, as shown in Figure 2. Importantly, these stages are sequentially connected rather than jointly optimized: ReST-GRPO first reshapes the trajectory distribution over sparse reward spaces, after which VM-MCTS estimates values under the resulting static policy. This separation recovers state-level guidance without returning to the full cost and instability of online actor–critic learning. Appendix A.1 tests the dependency under matched value learning, and Appendix A.8 discusses its practical cost.

- *ReST-GRPO* is the policy optimization stage, which integrates ReST-style filtering and partial-state starts into GRPO for sparse-reward reasoning.
- *VM-MCTS* is the reward learning stage, which focuses on specifically training and utilizing a value model (VM) that assists the policy in inference time search and decoding.

2.1 Task Setup

For a specific reasoning task, there is generally an instruction or question, denoted as q, which describes the background information and problem details in text. There may exist a ground truth answer g. For coding tasks, there may also be some test cases $t_{1,2,\ldots,m}$. Using an LLM policy known as π_θ, we can gradually generate solutions to the problem, following the predicted output token probabilities of the policy. This process can be regarded as a Markov Process, where the policy takes an action a_i step by step following $\pi_\theta(a_i|a_{1,2,\ldots,i-1}, q)$, based on previously generated content $a_1, a_2, \ldots, a_{i-1}$ and the instruction q. Eventually, when the eos token or stop strings are generated, the process stops, forming a final solution $A = (a_1, a_2, \ldots, a_k)$. In our approach, to align with the definitions in common RL, we define an action as a single line of text, which is separated by a line break. In Appendix A.8, we further illustrate the reasons for choosing this granularity. We also define an intermediate state as the combination of the instruction and a partial solution that contains lines of text, i.e. $S_i = (q, a_{1,2,\ldots,i})$. This means an intermediate state always terminates with a line break. In comparison, an end state stands for the combination of the instruction and a full solution that terminates with eos token, i.e. $S_{end} = S_k = (q, a_{1,2,\ldots,k})$. In consistency with conventional settings for reasoning scenarios, the transition to the next state after an action was taken is considered to be deterministic. In terms of reward, we assign rewards only for end states. The reward function for an end state is denoted by $R = R(S_{end})$, which can be either rule-based or model-based. Moreover, the policy value function defined by Eq. (1) can be unified with the Q function.

$$V^\pi(S_i) = \mathbb{E}_\pi[R(S_{end})|S_i], Q^\pi(S_i, a_{i+1}) = \mathbb{E}_\pi[R(S_{end})|S_{i+1}] = V^\pi(S_{i+1}). \tag{1}$$

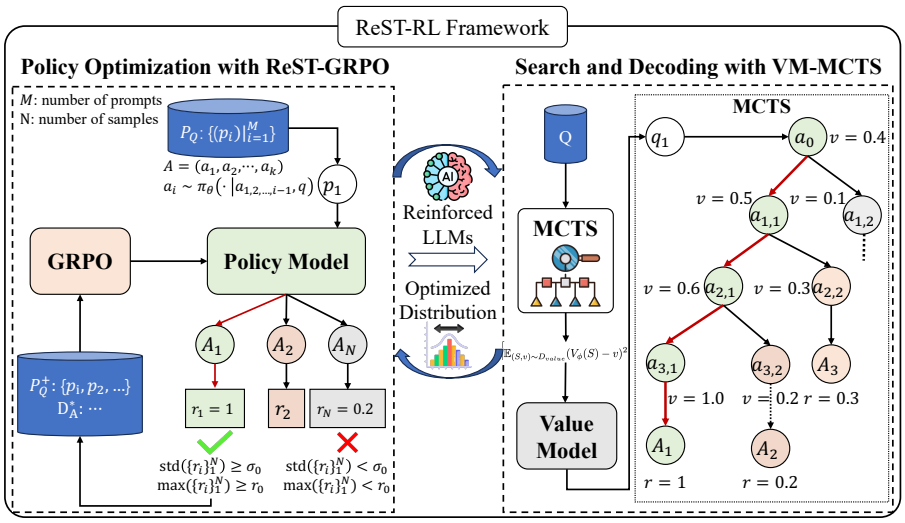


Figure 2: Framework of ReST-RL. ReST-GRPO reshapes the policy-induced trajectory distribution by filtering and assembling high-value training data. This optimized distribution improves VM learning and enables VM-MCTS to perform more reliable value-guided search and decoding.

2.2 ReST-GRPO

Building on ReST [4] and GRPO [25], Reinforced Self-Training with Group Relative Policy Optimization (ReST-GRPO) improves GRPO training through reward-based filtering and partial-state starts. Unlike ReST-style supervised training on accepted completions, an accepted prefix is used only as context: its suffix is freshly sampled online and optimized with the GRPO objective. Thus, the stored trajectory determines where online RL may restart but does not prescribe what the policy should generate afterward. The original prompt is also retained whenever it passes filtering, so the assembled set mixes ordinary prompt starts with state-localized starts. This extension reshapes the training distribution and mitigates low-information group rewards [36], while preserving on-policy exploration over newly sampled suffixes. It also increases the policy's coverage of high-reward trajectories used by the subsequent value-learning stage. ReST-GRPO is designed to reinforce the policy through multiple self-training iterations, each initialized from the checkpoint produced by the previous round. Every iteration contains three steps: (1) pre-train solution sampling, (2) train-data filtering and assembly, and (3) optimization with the GRPO objective. Algorithm 1 provides the detailed pseudocode.

2.2.1 Pre-train Solution Sampling

Similar to the ReST algorithm that augments the initial training dataset with samples from the policy itself, ReST-GRPO also utilizes the sample solutions generated from the policy for further train data selection and assembly. Within each iteration, we first collect N solutions for every instruction prompt in the original dataset using the current policy. By configuring the LLM's sampling temperature, we can control the randomness and diversity of these solutions. Then, we utilize a fixed reward function to obtain rewards for all solutions, which are used for subsequent filtering processes.

2.2.2 Data Filtering by Reward

An LLM policy's output solutions and their corresponding rewards contain a wealth of information indicating its strength and weakness toward the target domain, which can be leveraged to filter out ineffective training data. For GRPO, policy update relies on group relative advantage. This means effective training signals depend on discrepancies between rewards, offering three key enlightenments:

- If a policy's outputs obtain similar rewards on a question, there's nothing much for it to learn from.
- For questions where the current policy faces an enormous action space but only a few output traces lead to a substantial reward, normal sampling from the initial state may be ineffective for training.

- For a question that the current policy does not perform well, high-reward solution traces are crucial for training. Since high-reward solutions often share some common patterns, sampling from a partial solution state of a high-reward solution may be helpful for obtaining more high-reward traces, thus beneficial for training.

We design our algorithm's filtering and train data assembly process based on these enlightenments. Firstly, we use standard deviation to evaluate the variety of rewards. For question prompts that the policy's solutions achieve a reward standard deviation less than a given threshold σ_0, we filter them out from the train dataset, because they will possibly result in very little improvement of the policy. For other prompts, we add them to the train dataset. Secondly, we focus on high-reward solution traces that may be beneficial for training. Specifically, we filter out question prompts that the policy only achieves rewards lower than a certain threshold r_0. For other prompts, we extract their corresponding solution trace with the highest reward. Finally, its partial solution states are utilized to assemble new train data, as illustrated in Section 2.2.3. Figure 1a shows that ReST-GRPO increases reward variance, but this statistic is only a proxy for the informativeness of group-relative updates. We therefore separate it from downstream attribution: Table 3(a) evaluates the policy components on APPS-500 and BCB, while Appendix A.1 characterizes the induced trajectory distribution. Table 3(b) then tests whether the first-stage advantage remains after matched value learning and search.

2.2.3 Train Data Assembly

Considering that a partial state belonging to a high-reward solution can possibly be a good state for the policy to begin with, we extract a subset of all these partial solution states for new train data assembly. For simplicity, we only select from partial states of the solution trace A with the highest reward for each prompt p (this can be extended to top-k selection). Denoted by $a_{1,2,\ldots,j}$, a partial state contains the first j actions (i.e., lines) of A taken by the policy. In the assembly process, we first sample a subset of these partial states D_A^* based on a discrete finite exponential distribution, using a fixed positive exponent factor $\alpha < 1.0$. Sampling is performed following the probability mass function defined by Eq. (2). Relative to ordinary GRPO, every selected nonempty prefix moves the starting state along a high-reward trajectory. Within this set of partial states, the distribution below assigns greater mass to shorter, earlier prefixes than uniform sampling, retaining a larger suffix action space for online exploration. We sample $\beta|A|$ partial states and add them to D_A^*, where β limits the training budget, and append each state to the initial prompt p. As α approaches 0, the mass concentrates on the shortest partial prefix, so the resulting starts approach ordinary GRPO-style starts but are not identical to using only the original prompt.

$$p(a_{1,2,\ldots,j}) = \frac{1-\alpha}{1-\alpha^{|A|}}\alpha^{j-1}, j = 1, 2, \ldots, |A| \tag{2}$$

2.2.4 GRPO Training

After training data for each iteration are obtained, online training is performed. Policy update is based on the objective displayed in Eq. (3), which is basically the GRPO objective, but the policy's input prompt can be a combination of question and sampled partial solution from the train set P_Q^+.

$$\begin{aligned}\mathcal{J}(\theta) =& \mathbb{E}[p \sim P_Q^+, \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|p)] \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\{min[\frac{\pi_\theta(o_{i,t}|p,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|p,o_{i,<t})}\hat{A}_{i,t}, \\ & clip(\frac{\pi_\theta(o_{i,t}|p,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|p,o_{i,<t})}, 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}] - \beta'\mathbb{D}_{KL}[\pi_\theta||\pi_{ref}]\}\end{aligned} \tag{3}$$

2.3 VM-MCTS

Value Model based Monte-Carlo Tree Search (VM-MCTS) trains a VM from self-collected MCTS targets under a static policy and uses it for inference-time search and selection. Its target is the expected terminal reward $V^\pi(s)$ of a partial state, not whether the most recent line is independently correct. This distinction is useful in executable-reward settings, where intermediate steps may be difficult to label even though complete solutions can be evaluated automatically. The VM serves two roles: it allocates search toward promising reasoning subspaces and ranks completed candidates. Because V^π is policy-dependent, we pair each VM with the policy that generated its training

trajectories. The matched comparison in Table 3(b) evaluates this two-stage dependency without introducing policy–VM mismatch. We compare VM-MCTS with other MCTS-based approaches in Appendix A.8. VM-MCTS is inspired by but extends beyond previous process-supervision methods such as Math-Shepherd (M-S) [31]. Rather than serving only as a standalone verifier, the VM is introduced as a value estimator trained on self-collected search data under the static policy, enabling test-time search that is both fine-grained and broadly applicable. Although the value target introduces some extra noise, it leads to better performance when decoding with the assistance of MCTS.

2.3.1 The Value Target

In the M-S method, the quality of an intermediate step is defined as its potential to deduce the correct final answer, which is essentially an expectation-based evaluation target. However, this estimation is still treated as a measure of single step correctness. For VM-MCTS, we instead summarize this to the value target defined in Eq. (1), emphasizing the evaluation of the whole partial state including the last action. It naturally reflects the potential for a policy to reach high-reward end states from the current partial state, which can be leveraged to assist policy decoding. In fact, the M-S soft target can be regarded as a special case of value target where the reward function R is defined by Eq. (4).

$$R(S_{end}) = \mathbb{I}(a_A = a^*), \;\; s.t. \;\; S_{end} = (q, A) \tag{4}$$

2.3.2 Value Model Training

We adopt MCTS when collecting training data for the VM, as it balances exploration of different reasoning paths and exploitation of promising intermediate states, aligning with our design principles. In our algorithm, the root node of the search tree is set to the initial state $S_{init} = S_0 = (q, \varnothing)$, while other nodes represent intermediate states or end states $S_i = (q, a_{1,2,\ldots,i})$. As illustrated in Section 2.1, an action represents a line of text. Through multiple rounds of MC simulation, value targets of partial states and end states are gradually updated and precisely estimated, distinguished from the all-at-once estimation used by M-S. Because these targets are collected under the current static policy, the quality of value learning depends on the policy-induced trajectory distribution, which is precisely why the preceding ReST-GRPO is crucial. We present the pseudocode for this process in Algorithm 2. Once sufficient value data are acquired, a VM V_ϕ is trained on these data to predict the value target of different states. The loss function used by VM-MCTS for training is demonstrated in Eq. (5).

$$\mathcal{L}_\phi = \mathbb{E}_{(S,v)\sim D_{value}}(V_\phi(S) - v)^2 \tag{5}$$

2.3.3 Assisted Decoding

Previous work such as PPO-MCTS [16] demonstrates the utility of VMs for token-level MCTS decoding but relies on coupled policy–value training. Here, a separately trained VM predicts expected reward for partial states under a static policy and known rewards for terminal states. As shown in Algorithm 3, the adapted search uses VM estimates for single-step MC rollouts and for final Best-of-N-style selection [13]. Search allocation and final verification therefore use a common state-value scale, rather than combining independently trained guidance and reranking models. Under the assumptions stated in Appendix A.12, the value-based rollout remains unbiased and has no greater variance than complete-trace rollout.

3 Experiments

As an important branch of LLM reasoning tasks, code writing has received much attention in recent years [41; 11; 10]. In this research, we also focus on coding problems, considering that they have a more comprehensive test of LLM's reasoning, planning, and invocation abilities. In this section, we present various experiments to validate ReST-RL. Since ReST-RL jointly involves policy optimization and value-guided decoding, we also include separate evaluation of ReST-GRPO and VM-MCTS against their related baselines in order to identify the contribution of each design under controlled settings. These evaluations are used to establish a fair comparison for each stage aside from testing the complete pipeline. As a whole, we further show that the two stages form a coherent dependency chain and that their unity leads to the best end-to-end performance. In addition, to assess practical feasibility from an end-to-end perspective, we provide supplementary analyses of training-time and inference-time cost, latency, and efficiency for ReST-RL in Appendix A.8.

Table 1: Main results for policy training. LLM policies are evaluated after each training iteration ends. "Iter." refers to sequential training iterations on Q_{train} (0th iter. means not trained). The complete DAPO comparison is conducted under the main Qwen3-8B configuration; the remaining model blocks retain the original baseline set.

Model	Training Method	HumanEval	HumanEval+	MBPP	MBPP+	APPS-500	BCB	Average
Qwen3-8B	Base (0th iter.)	0.829	0.780	0.717	0.622	0.118	0.418	0.503
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.854	0.799	0.730	0.627	0.152	0.434	0.523
	GRPO (1st iter.)	0.799	0.750	0.754	0.651	0.346	0.439	0.566
	DAPO (1st iter.)	0.805	0.768	0.770	0.667	0.351	0.425	0.570
	ReST-GRPO (1st iter.)	**0.872**	**0.817**	**0.780**	**0.672**	**0.377**	**0.469**	**0.604**
	ReST-DPO (2nd iter.)	0.841	**0.811**	0.741	0.653	0.153	0.429	0.526
	GRPO (2nd iter.)	0.829	0.787	0.757	0.667	0.403	0.436	0.590
	DAPO (2nd iter.)	0.811	0.773	0.770	0.676	0.428	0.444	0.597
	ReST-GRPO (2nd iter.)	**0.860**	0.805	**0.802**	**0.690**	**0.565**	**0.476**	**0.655**
Qwen2.5-Coder-7B-Instruct	Base (0th iter.)	0.872	0.799	0.817	0.683	0.287	0.381	0.563
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.884	0.823	0.820	0.688	0.321	0.386	0.579
	GRPO (1st iter.)	**0.896**	**0.835**	0.812	0.693	0.350	0.405	0.593
	ReST-GRPO (1st iter.)	0.872	0.817	0.823	0.701	0.401	0.441	0.612
	ReST-DPO (2nd iter.)	0.872	0.823	0.796	0.669	0.330	0.400	0.578
	GRPO (2nd iter.)	0.872	0.799	0.815	0.685	0.356	0.403	0.586
	ReST-GRPO (2nd iter.)	0.890	**0.835**	**0.849**	**0.712**	**0.415**	**0.462**	**0.630**
DS-Coder-6.7b-Instruct	Base (0th iter.)	0.744	0.677	0.741	0.646	0.230	0.338	0.493
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.756	0.683	0.725	0.635	0.251	0.329	0.495
	GRPO (1st iter.)	0.726	0.646	0.735	0.632	0.262	0.342	0.493
	ReST-GRPO (1st iter.)	0.756	0.683	0.743	0.635	0.282	0.355	0.511
	ReST-DPO (2nd iter.)	0.756	0.689	0.720	0.632	0.242	0.329	0.492
	GRPO (2nd iter.)	0.707	0.634	0.730	0.624	0.283	0.357	0.497
	ReST-GRPO (2nd iter.)	**0.793**	**0.707**	**0.749**	**0.646**	**0.300**	**0.368**	**0.529**
OpenCI-DS-6.7B	Base (0th iter.)	0.756	0.707	0.722	0.630	0.204	0.331	0.486
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.756	0.677	0.714	0.622	0.212	0.351	0.487
	GRPO (1st iter.)	0.726	0.677	0.733	0.632	0.277	0.363	0.506
	ReST-GRPO (1st iter.)	0.726	0.677	0.725	0.635	0.281	0.374	0.509
	ReST-DPO (2nd iter.)	0.744	0.671	0.725	0.630	0.232	0.336	0.488
	GRPO (2nd iter.)	0.750	0.689	**0.743**	**0.648**	0.279	0.352	0.512
	ReST-GRPO (2nd iter.)	**0.774**	**0.713**	0.725	0.630	**0.325**	**0.377**	**0.531**

Table 2: Average results of ReST-RL and different reward verification methods on all benchmarks. All verification is based on 100 samples. ReST-RL is globally optimal when combining the Table 1.

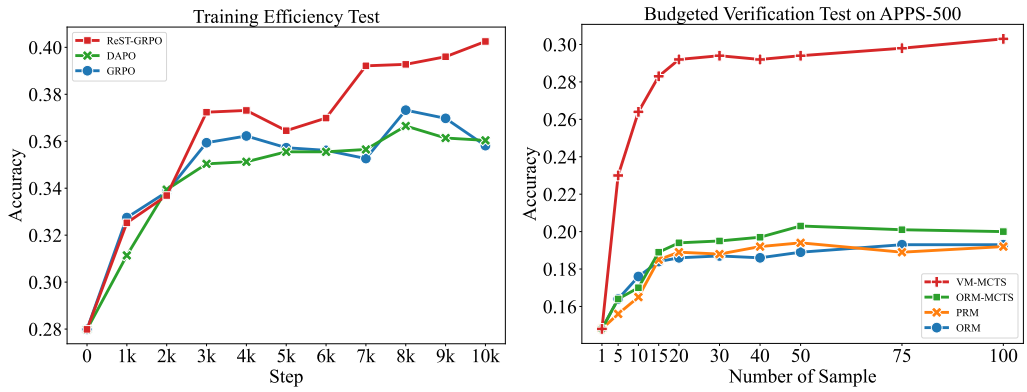
Method	Qwen3-8B	Qwen2.5-Coder-7B-Instruct	DS-Coder-6.7b-Instruct	OpenCI-DS-6.7B
Base	0.503	0.563	0.493	0.486
ORM	0.531	0.592	0.542	0.537
PRM	0.516	0.591	0.539	0.532
ORM-MCTS	0.538	0.588	0.547	0.535
VM-MCTS	0.615	0.652	0.576	0.569
ReST-RL	**0.689**	**0.673**	**0.584**	**0.583**

3.1 Experiment Settings

In this section, we present the main experiment settings, including selected LLMs, datasets used for training and base reward function. Additionally, more experimental details for benchmark evaluation are displayed in Appendix A.3.

Base LLM Policies. We primarily test recent code LLMs: Qwen2.5-Coder-7B-Instruct [8], CodeQwen1.5-7B-Chat [27], Deepseek-Coder-6.7B-Instruct [5], and OpenCodeInterpreter-DS-6.7B [43]. We additionally include Qwen3-8B [28], Llama-3-8B, and Llama-3.1-8B-Instruct [19] to test whether the coding improvements extend across both specialized and general base policies.

Training Datasets. In terms of source datasets for training, we select three well-known open-source coding datasets: BigCodeBench (BCB) [45], DS-1000 [9] and APPS [6]. After filtering out data in the source train splits (or the randomly sampled train split for BCB) without test cases, we obtain a final dataset Q_{train} of $6,945$ coding prompts. For experiments mentioned in Section 3.2 and Section 3.3, training of policies and VMs with ReST-RL and other baselines are all based on this dataset. All APPS-500 results use one immutable random subset, and we will release its exact problem IDs



(a) Training efficiency of ReST-GRPO, DAPO, and GRPO on average benchmark score. Llama-3-8B is trained for $10k$ steps and evaluated every $1k$ steps.

(b) Budgeted verification on APPS-500 with CodeQwen. Sampling temperature is set to 0.7 for all methods.

Figure 3: Test of training efficiency and budgeted verification.

and evaluation script. Appendix A.3 specifies the metric and protocol and records that the historical subset-construction seed is unavailable.

Base Reward Function. According to previous research, the use of model-based learned reward functions in reinforcement learning scenarios may increase the risk of reward hacking [33], causing undesirable effects on our experimental results. Therefore, we primarily adopt a rule-based reward function that uses test cases to calculate reward, as shown in Eq. (6). It evaluates generated codes with the ratio of passed test cases to total test cases, which naturally reflects the functional correctness of these codes. Somewhat differently, two additional rewards have been added to improve stability and effectiveness for ReST-GRPO and naive GRPO training, which is illustrated in Appendix A.5.

$$R_{base}(S_{end}) = R_{base}(q, A) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{I}(\text{eval}(A, t_i) = y_i), \tag{6}$$

where t_i are test cases, y_i are desired test outputs.

3.2 Evaluation of LLM Policy Training Methods

To assess ReST-GRPO, we compare it with ReST-DPO (essentially ReST^{EM} [26] with DPO optimization), naive GRPO, and DAPO [36]. These baselines cover offline preference optimization, ordinary online group-relative optimization, and full dynamic sampling. We evaluate this stage separately because its immediate role is to improve the policy and reshape the trajectory distribution before any VM is trained. All policy methods use a learning rate of 1e-7. Appendix A.5 provides their remaining configurations. Table 3(a) reports downstream component controls, while Appendix A.1 separates reward-signal diagnostics from final task performance.

Performance comparison for iterative policy training. We perform two sequential training iterations on Q_{train}. Table 1 reports four LLM policies, with additional Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat results in Appendix A.7. ReST-GRPO improves the average over ReST-DPO and naive GRPO for every model and iteration. Under the main Qwen3-8B configuration, it also outperforms DAPO on all six benchmarks, with average margins of 3.4 points after the first iteration and 5.8 after the second. After two iterations, ReST-GRPO improves the selected models' average scores by 15.2%, 6.7%, 3.6%, and 4.5%, respectively, without degrading any benchmark. Table 9 shows that the default setting remains competitive under reasonable hyperparameter changes.

Assessment of training efficiency for ReST-GRPO, DAPO and naive GRPO. By evaluating policy performance after same number of training steps, we get a glimpse of the difference in training efficiency between ReST-GRPO, DAPO [36] and naive GRPO. Figure 3(a) compares average accuracy for Llama-3-8B over $10k$ steps. We use this general base model because it leaves substantial room for coding improvement and makes differences in optimization efficiency easier to observe than with a heavily code-specialized checkpoint. The three methods are comparable during the first $2k$ steps, after which ReST-GRPO's lead grows. At $10k$ steps its improvement is 12.3%, versus 7.8%

for naive GRPO and 8.1% for DAPO. Figure 4 isolates online training throughput, whereas Table 4 includes the one-time offline sampling cost. Under this end-to-end accounting, ReST-GRPO uses 1,752 GPU-hours versus GRPO's 2,080 and reaches a 9% improvement in 71 versus 207 hours. Thus, the offline stage is amortized by fewer online hours to reach the same performance level.

Component and curriculum controls. Table 3(a) replaces reward variance as the sole attribution evidence with final downstream results. Filtering-only and partial-only each improve over naive GRPO, but their integration performs best (0.207/0.440 on APPS-500/BCB). E2H-C changes which original prompts are sampled over time and yields 0.189/0.418, whereas ReST-GRPO additionally changes where online RL resumes. Exponential prefix sampling also outperforms uniform sampling in this setting; relative to uniform it favors earlier prefixes and therefore leaves a larger suffix action space. Full settings and end-to-end compute are reported in Appendix A.2.

3.3 Comparison with Decoding and Verification Methods

Here we compare VM-MCTS with representative ORM-, PRM-, and MCTS-based decoding methods. These assessments test whether the learned value signal guides search and verification better once a policy is fixed. We include ORM+BoN (ORM), PRM+BoN (PRM), and ORM-MCTS. Prior work finds MCTS more effective than BFS/DFS tree search in comparable reasoning settings [39; 34], thus we do not include BFS/DFS based variants as baselines. Appendix A.6 provides detailed configurations. **Comparison of method accuracy on all benchmarks.** Table 2 reports average results with verification limited to 100 candidates. VM-MCTS improves over the strongest listed baseline by 11.2%, 8.9%, 8.3%, and 8.3% for Qwen3-8B, Qwen2.5-Coder, DS-Coder, and OpenCI, respectively. Its lead over ORM-MCTS supports using a state-localized value estimate during decoding rather than only scoring completed outputs. Since all methods use the same candidate budget in this comparison, the result measures sample efficiency. Appendix Table 10 further shows smooth gains as the branch factor increases.

Table 3: Controlled Qwen3-8B comparisons. Panel (a) matches optimizer, reward, $G = 8$, and 2,000 updates; applicable variants reuse the same offline trajectories. Panel (b) matches each VM to its generating policy. Panel (c) fixes the policy and VM while approximately matching latency.

(a) Policy components

Method	APPS-500	BCB
Naive GRPO	0.180	0.416
Filtering-only	0.193	0.429
E2H-C	0.189	0.418
Partial-only	0.190	0.425
Uniform-prefix	0.187	0.418
Full ReST-GRPO	**0.207**	**0.440**

(b) Matched policy–VM pipelines

Pipeline	APPS-500	BCB
GRPO (policy only)	0.403	0.436
GRPO + matched VM-MCTS	0.538	0.449
ReST-GRPO (policy only)	0.565	0.476
ReST-GRPO + matched VM-MCTS	**0.642**	**0.506**

(c) Approximately matched latency

Method	APPS-500	Time
VM-BoN ($N = 260$)	0.619	$\approx$41.5 s†
VM-MCTS ($n = 5, T = 20$)	**0.642**	41.5 s

† VM-BoN latency is the calibration target.

Verification results based on controlled budget. Figure 3(b) compares APPS-500 accuracy across sample budgets with CodeQwen. VM-MCTS remains ahead across the tested range, and Appendix Table 8 reports the corresponding token usage. Equal candidate counts, however, do not imply equal latency. We therefore compare the same ReST-GRPO policy and VM at approximately matched mean wall-clock in Table 3(c): VM-MCTS obtains 0.642 versus 0.619 for VM-BoN calibrated at $N = 260$ (both about 41.5 s per prompt). Because policy, verifier, and latency are controlled, the remaining difference isolates whether the VM is used during tree allocation or only for final reranking. Appendix Table 18 retains the complementary equal-sample latency trade-off.

Table 4: End-to-end training time (hours) assessment for ReST-GRPO and naive GRPO using Llama-3-8B. We conduct training for a total of 10k steps (same number of samples) with each method. Improvement is evaluated based on average score across all benchmarks, and pre-training sampling time for ReST-GRPO is always included.

Method	Pre-sampling Time	Total Training Time	Total GPU Hours	6% Improvement	9% Improvement	12% Improvement
GRPO	0	260	2080	52	207	>260
ReST-GRPO	10	219	1752	51	71	199

3.4 Validation of the Overall Approach

Based on the final policies trained with ReST-GRPO, we evaluate the full ReST-RL pipeline. Table 2 reports the best average result for each base model. Appendix Table 6 provides mechanism-level evidence: identical MCTS collection yields mean trajectory rewards of 0.37/0.56/0.68 for Base/GRPO/ReST-GRPO. The matched downstream test in Table 3(b) holds VM initialization, architecture, 1.1M targets, collection, optimization, and search fixed. A GRPO-trained VM improves its policy to 0.538/0.449 on APPS-500/BCB, while the matched ReST-GRPO pipeline reaches 0.642/0.506. Thus a learned state-localized VM benefits either policy, and the first-stage policy-induced advantage persists through matched value learning and search.

Table 5: Preliminary cross-domain transfer results with Qwen3-8B. The VM is trained only on code trajectories and receives no target-domain tuning. We also pair the initial VM (0th) with ReST-GRPO (2nd) to assess robustness to policy shift.

Model	Method	APPS-500	BCB	MATH	Omni-MATH	GPQA-Diamond
Qwen3-8B	Base (0th)	0.118	0.418	0.780	0.234	0.449
	Base (0th) + VM (0th)	0.415	0.471	0.828	0.238	0.460
	ReST-GRPO (2nd) + VM (0th)	0.630	0.496	0.862	0.246	0.480
	ReST-RL (2nd)	**0.642**	**0.506**	**0.872**	**0.256**	**0.490**

Preliminary cross-domain transfer. We investigate whether a code-trained ReST-RL policy and VM remain useful on out-of-domain tasks and after policy updates. Note that coding remains the principal empirical domain, and this experiment tests transfer without target-domain tuning rather than claiming a universal verifier. Table 5 reports Qwen3-8B based results on MATH [7], Omni-MATH [3], and GPQA-Diamond [23]. Without target-domain tuning, adding the code-trained VM to the base policy improves the three scores from 0.780/0.234/0.449 to 0.828/0.238/0.460. The initial VM also remains useful after two policy-training rounds (0.862/0.246/0.480), while updating it to the final shifted policy yields 0.872/0.256/0.490. These controlled observations provide evidence that the domain-agnostic objective $V^{\pi}(s) = \mathbb{E}_{\pi}[R \mid s]$ can transfer across task and policy shifts. The initial VM's retained gains also indicate robustness to moderate policy shift, while the updated matched VM performs best.

4 Conclusion

In this paper, we introduce **ReST-RL**, a unified, state-localized policy–value framework for Reinforced Self-Training. ReST-GRPO combines reward-based filtering with partial-state online RL: selected prefixes determine new starting contexts, while their suffixes are freshly sampled and optimized rather than imitated. VM-MCTS subsequently learns $V^{\pi}(s)$ under the resulting static policy and uses this signal for both MCTS allocation and final output selection. The two stages therefore reconnect policy improvement and value-guided reasoning without requiring a jointly trained online actor–critic.

Across multiple coding benchmarks and model families, ReST-GRPO improves over ReST-DPO, naive GRPO, and DAPO, while VM-MCTS improves over representative verification and search baselines. Controlled component, matched policy–VM, and approximately latency-matched comparisons further support the distinct contribution of state-localized training and value-guided allocation. End-to-end accounting shows that the offline sampling cost can be amortized by more efficient online training. Finally, math and science results without target-domain tuning provide transfer evidence beyond our primary coding domain.

Acknowledgements

This work was supported by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM101), NSFC for Distinguished Young Scholar (62425601), New Cornerstone Science Foundation through the XPLORER PRIZE and CIPS-SMP-Zhipu Large Model Fund (No. CIPS-SMP20250319).

References

[1] Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[2] Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[3] Gao, B., Song, F., Yang, Z., Cai, Z., Miao, Y., Dong, Q., Li, L., Ma, C., Chen, L., Xu, R., Tang, Z., Wang, B., Zan, D., Quan, S., Zhang, G., Sha, L., Zhang, Y., Ren, X., Liu, T., and Chang, B. Omni-math: A universal olympiad level mathematic benchmark for large language models, 2024. URL https://arxiv.org/abs/2410.07985.

[4] Gulcehre, C., Paine, T. L., Srinivasan, S., Konyushkova, K., Weerts, L., Sharma, A., Siddhant, A., Ahern, A., Wang, M., Gu, C., Macherey, W., Doucet, A., Firat, O., and de Freitas, N. Reinforced self-training (rest) for language modeling, 2023. URL https://arxiv.org/abs/2308.08998.

[5] Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y. K., Luo, F., Xiong, Y., and Liang, W. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024. URL https://arxiv.org/abs/2401.14196.

[6] Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., and Steinhardt, J. Measuring coding challenge competence with apps, 2021. URL https://arxiv.org/abs/2105.09938.

[7] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

[8] Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Dang, K., et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

[9] Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., tau Yih, S. W., Fried, D., Wang, S., and Yu, T. Ds-1000: A natural and reliable benchmark for data science code generation, 2022. URL https://arxiv.org/abs/2211.11501.

[10] Le, H., Wang, Y., Gotmare, A. D., Savarese, S., and Hoi, S. C. H. Coderl: Mastering code generation through pretrained models and deep reinforcement learning, 2022. URL https://arxiv.org/abs/2207.01780.

[11] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Sutherland Robson, E., Kohli, P., de Freitas, N., Kavukcuoglu, K., and Vinyals, O. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158.

[12] Light, J., Cheng, W., Riviere, B., Yue, W., Oyamada, M., Wang, M., Yue, Y., Paternain, S., and Chen, H. Disc: Dynamic decomposition improves llm inference scaling, 2025. URL https://arxiv.org/abs/2502.16706.

[13] Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

[14] Liu, C. Y., Zeng, L., Liu, J., Yan, R., He, J., Wang, C., Yan, S., Liu, Y., and Zhou, Y. Skywork-reward: Bag of tricks for reward modeling in llms. *arXiv preprint arXiv:2410.18451*, 2024.

[15] Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation, 2023. URL https://arxiv.org/abs/2305.01210.

[16] Liu, J., Cohen, A., Pasunuru, R., Choi, Y., Hajishirzi, H., and Celikyilmaz, A. Don't throw away your value model! generating more preferable text with value-guided monte-carlo tree search decoding, 2024. URL https://arxiv.org/abs/2309.15028.

[17] Lu, T., Li, Y., Wang, L., Lin, B., Lv, Q., Xu, Z., Zheng, H.-T., Li, Y., and Kim, H.-G. From token to line: Enhancing code generation with a long-term perspective, 2026. URL https://arxiv.org/abs/2504.07433.

[18] Luo, H., Sun, Q., Xu, C., Zhao, P., Lou, J., Tao, C., Geng, X., Lin, Q., Chen, S., Tang, Y., and Zhang, D. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct, 2025. URL https://arxiv.org/abs/2308.09583.

[19] Meta. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[20] OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

[21] Parashar, S., Gui, S., Li, X., Ling, H., Vemuri, S., Olson, B., Li, E., Zhang, Y., Caverlee, J., Kalathil, D., and Ji, S. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning, 2025. URL https://arxiv.org/abs/2506.06632.

[22] Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[23] Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[24] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[25] Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

[26] Singh, A., Co-Reyes, J. D., Agarwal, R., Anand, A., Patil, P., Liu, P. J., Harrison, J., Lee, J., Xu, K., Parisi, A., et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

[27] Team, Q. Code with codeqwen1.5, April 2024. URL https://qwenlm.github.io/blog/codeqwen1.5/.

[28] Team, Q. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[29] Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

[30] Ugadiarov, L., Kuratov, Y., Panov, A., and Skrynnik, A. Revisiting tree search for llms: Gumbel and sequential halving for budget-scalable reasoning, 2026. URL https://arxiv.org/abs/2603.21162.

[31] Wang, P., Li, L., Shao, Z., Xu, R. X., Dai, D., Li, Y., Chen, D., Wu, Y., and Sui, Z. Math-shepherd: Verify and reinforce llms step-by-step without human annotations, 2024. URL https://arxiv.org/abs/2312.08935.

[32] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[33] Weng, L. Reward hacking in reinforcement learning. *lilianweng.github.io*, Nov 2024. URL https://lilianweng.github.io/posts/2024-11-28-reward-hacking/.

[34] Xu, B., Lin, Y., Li, Y., and Gao, Y. Sra-mcts: Self-driven reasoning augmentation with monte carlo tree search for code generation, 2025. URL https://arxiv.org/abs/2411.11053.

[35] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

[36] Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Dai, W., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Yan, L., Qiao, M., Wu, Y., and Wang, M. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

[37] Zelikman, E., Wu, Y., Mu, J., and Goodman, N. D. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

[38] Zhang, D., Huang, X., Zhou, D., Li, Y., and Ouyang, W. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b, 2024. URL https://arxiv.org/abs/2406.07394.

[39] Zhang, D., Zhoubian, S., Hu, Z., Yue, Y., Dong, Y., and Tang, J. Rest-mcts*: Llm self-training via process reward guided tree search, 2024. URL https://arxiv.org/abs/2406.03816.

[40] Zhang, J., Huang, J., Yao, H., Liu, S., Zhang, X., Lu, S., and Tao, D. R1-vl: Learning to reason with multimodal large language models via step-wise group relative policy optimization, 2025. URL https://arxiv.org/abs/2503.12937.

[41] Zhang, S., Chen, Z., Shen, Y., Ding, M., Tenenbaum, J. B., and Gan, C. Planning with large language models for code generation, 2023. URL https://arxiv.org/abs/2303.05510.

[42] Zhang, Z., Zheng, C., Wu, Y., Zhang, B., Lin, R., Yu, B., Liu, D., Zhou, J., and Lin, J. The lessons of developing process reward models in mathematical reasoning, 2025. URL https://arxiv.org/abs/2501.07301.

[43] Zheng, T., Zhang, G., Shen, T., Liu, X., Lin, B. Y., Fu, J., Chen, W., and Yue, X. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024.

[44] Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning acting and planning in language models, 2024. URL https://arxiv.org/abs/2310.04406.

[45] Zhuo, T. Y., Vu, M. C., Chim, J., Hu, H., Yu, W., Widyasari, R., Yusuf, I. N. B., Zhan, H., He, J., Paul, I., Brunner, S., Gong, C., Hoang, T., Zebaze, A. R., Hong, X., Li, W.-D., Kaddour, J., Xu, M., Zhang, Z., Yadav, P., Jain, N., Gu, A., Cheng, Z., Liu, J., Liu, Q., Wang, Z., Hui, B., Muennighoff, N., Lo, D., Fried, D., Du, X., de Vries, H., and Werra, L. V. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions, 2025. URL https://arxiv.org/abs/2406.15877.

A Appendix

A.1 Ablation and Sensitivity Analysis

We analyze ReST-RL from both training and inference perspectives. The results address two questions: whether performance is robust to reasonable hyperparameter changes, and whether the policy-induced trajectory shift remains consequential after matched value learning. Downstream component controls are separated from the reward-variance proxy to avoid treating an optimization statistic as final task performance.

ReST-GRPO: reshaping distribution for an appropriate search policy. Table 7 is a reward-standard-deviation proxy analysis: filtering increases mean SD from 0.104 to 0.148, while high-reward trajectory selection and partial-state sampling further increase it to 0.168. Final downstream attribution is reported separately in Table 11. Table 9 varies σ_0, α, and r_0 and shows a relatively narrow performance range, with the default configuration strongest overall. These results indicate robustness to reasonable perturbations without implying that reward variance alone establishes task-level gains.

VM-MCTS: unlock the potential of search under controlled budgets. Table 10 studies the branch factor n in VM-MCTS. APPS-500 accuracy increases from the policy-only score of 0.565 to 0.652 as the search becomes more expressive. Gains are already substantial at the main setting $n = 5$, indicating a smooth compute–performance trade-off rather than dependence on one branch factor.

Policy-induced trajectories and matched value learning. Table 6 uses the same MCTS collection procedure as VM training. Relative to Base and GRPO, ReST-GRPO increases mean trajectory reward from 0.37/0.56 to 0.68 and median reward from 0.01/0.60 to 0.78. This establishes a distributional shift, while Table 3(b) tests its downstream consequence. With VM initialization, architecture, 1.1M targets, collection parameters, optimization steps, seed, and VM-MCTS fixed, the matched GRPO and ReST-GRPO pipelines obtain 0.538/0.449 and 0.642/0.506 on APPS-500/BCB, respectively. Because a VM estimates V^π, each VM is evaluated with the policy that generated its trajectories; a mismatched VM would mix data quality with policy–VM distribution shift.

Practical suggestions for hyperparameter selection. The ablations also suggest practical defaults. With rewards in $[0, 1]$, σ_0 can start in a small range such as $(0, 0.1)$ to avoid discarding informative prompts. In $p(j) \propto \alpha^{j-1}$, lower α concentrates mass on shorter prefixes, while α near one approaches a uniform distribution over prefix lengths; $\alpha = 0.95$ provides a mild early-prefix bias in our setting. For r_0, stricter thresholds favor more reliable high-reward anchors. For VM-MCTS, n and T should be selected according to the available latency budget.

Overall implication. Together, the sensitivity, trajectory, and matched-pipeline results show that the method is stable under reasonable hyperparameter changes and that its two stages remain complementary under controlled value training and search.

Table 6: Trajectory reward statistics for Qwen3-8B and its trained variants under the same MCTS collection procedure used for VM training. ReST-GRPO shifts the sampled distribution toward higher-reward regions; the downstream consequence is evaluated directly in Table 3(b).

Method	Base	GRPO	ReST-GRPO
Mean Reward	0.37	0.56	0.68
Median Reward	0.01	0.60	0.78

Table 7: Reward-standard-deviation proxy analysis for train-data sampling variants. This table characterizes the training signal; downstream component performance is reported separately in Table 11.

Sampling Method	$\sigma_0 = 0, \beta = 0$	$\alpha \to 0 (0.01)$	$r_0 = 0.1$	$r_0 = 0.5$	$r_0 = 0.9$
Mean SD	0.104	0.148	0.157	0.163	**0.168**
Median SD	0.000	0.120	0.111	0.124	**0.130**

Table 8: Average token usage per prompt for CodeQwen. We compare ORM + Best-of-N with our VM-MCTS method, showing the number of decoded tokens under various sampling sizes.

Model	Method	N=5	N=10	N=20	N=50	N=100
CodeQwen	Best-of-N	1290	2566	5137	12800	25564
	VM-MCTS	1287	2544	4962	12055	22427

Table 9: Sensitivity analysis for the main hyperparameters of ReST-GRPO. Qwen3-8B is trained with each setting for 1k steps, and the influence of each hyperparameter is assessed individually.

Hyperparameter	Value 1	Value 2	Value 3
σ_0	0	0.05	0.2
Acc (APPS)	0.139	0.140	0.133
α	0.01	0.5	0.95
Acc (APPS)	0.131	0.137	0.140
r_0	0.1	0.5	0.9
Acc (APPS)	0.134	0.138	0.140

Table 10: Sensitivity to the VM-MCTS branch factor on APPS-500. The policy is Qwen3-8B after two ReST-GRPO iterations; evaluation uses $T = 20$ and otherwise follows the main decoding configuration.

n (branch factor)	None (Policy only)	2	3	5	8
APPS-500 ($T = 20$)	0.565	0.613	0.625	0.642	0.652

Table 11: Downstream component ablations from the same Qwen3-8B checkpoint. Each variant uses the same reward, optimizer, rollout group $G = 8$, and 2,000 updates; variants requiring offline statistics reuse the same $N = 30$ trajectories. GPU-hours in this table cover online training only; end-to-end accounting is in Table 12.

Method	Prompt selection	Partial starts	Prefix rule	APPS-500 / BCB	Online GPU-h
Naive GRPO	None	No	–	0.180 / 0.416	410
Filtering-only	Static filter	No	–	0.193 / 0.429	434
E2H-C	Dynamic E2H	No	–	0.189 / 0.418	406
Partial-only	None	Yes	Exponential	0.190 / 0.425	370
Uniform-prefix	Static filter	Yes	Uniform	0.187 / 0.418	294
Full ReST-GRPO	Static filter	Yes	Exponential	**0.207 / 0.440**	309

A.2 Controlled Component and Compute Ablations

Table 11 reports final downstream performance from the same Qwen3-8B checkpoint. Filtering-only and Partial-only improve APPS-500 over naive GRPO by 1.3 and 1.0 points, while Full ReST-GRPO improves by 2.7 points and also performs best on BCB. Uniform-prefix changes only the prefix-position distribution and falls below the exponential rule. The latter favors earlier prefixes relative to uniform sampling, leaving more of the suffix available for online exploration. Relative to ordinary GRPO, both rules still introduce nonempty partial-state starts.

E2H-C follows the cosine easy-to-hard schedule of Parashar et al. [21]. It forms four quantile bins using empirical pass rate from the same stored trajectories and gradually shifts sampling from easier to harder original prompts. It never uses partial-state starts. Its 0.189/0.418 APPS-500/BCB result shows that prompt-level scheduling alone does not match state-localized training in this controlled setting.

Table 12 distinguishes online training cost from the shared offline trajectory cost. The 309 GPU-hours reported for Full ReST-GRPO's controlled run are online GPU-hours, including 96 offline GPU-hours gives 405 total, compared with 410 for naive GRPO. We likewise separate offline and online tokens and reward calls rather than presenting the online portion as end-to-end cost.

Table 12: End-to-end compute accounting for the 2,000-update Qwen3-8B controlled runs. Offline trajectories are generated once and reused by all applicable variants; totals nevertheless include their attributable offline cost. Reward calls count solution-level executions.

Method	GPU-hours			Sampled tokens (M)			Reward calls			Updates
	Offline	Online	Total	Offline	Online	Total	Offline	Online	Total	
Naive GRPO	0	410	410	0	22.4	22.4	0	32k	32k	2,000
Filtering-only	96	434	530	56.3	22.3	78.6	208k	32k	240k	2,000
E2H-C	96	406	502	56.3	22.4	78.7	208k	32k	240k	2,000
Partial-only	96	370	466	56.3	29.2	85.5	208k	32k	240k	2,000
Uniform-prefix	96	294	390	56.3	23.8	80.1	208k	32k	240k	2,000
Full ReST-GRPO	96	309	**405**	56.3	24.1	80.4	208k	32k	240k	2,000

A.3 Benchmark Evaluation Details

We evaluate on nine benchmarks: HumanEval [2], HumanEval+ [15], MBPP [1], MBPP+ [15], BigCodeBench (BCB) [45], APPS [6], MATH [7], Omni-MATH [3], and GPQA-Diamond [23]. We report pass@1 except on APPS-500, where we follow APPS and average the fraction of passed test cases. All evaluations are zero-shot.

APPS-500 reproducibility. All experiments use the same immutable 500-problem subset stored as `data/APPS/test_500.jsonl` and no response experiment resamples it. We will release this file together with problem IDs/indices, evaluation scripts and commands, prompt templates, metric definitions, temperature, and maximum generation length. The historical random seed used to construct the subset was not logged, so we do not claim that the sampling process can be reconstructed from a seed. Releasing the exact subset instead fully determines the evaluated examples.

A.4 Main Algorithms

We present the self-training ReST-GRPO algorithm in Algorithm 1, the automated training data collection process of VM in Algorithm 2 and the assisted decoding algorithm of VM-MCTS in Algorithm 3.

Algorithm 1 ReST-GRPO algorithm for policy training.

Input: base LLM policy π_θ, reward function (model) R, original question dataset Q, original prompt dataset P_Q, number of solution samples N, standard deviation threshold σ_0, reward threshold r_0, ratio of data to sample β, sampling exponent factor α, number of iterations T.

for $i = 1$ to T **do**
  $P_Q^+ \leftarrow \varnothing$ # initialize train set
  for prompt p in P_Q **do**
    Sample N solutions $\{A_i\}_{i=1}^N \sim \pi_\theta(\cdot\,|p)$ # generate solutions samples
    Get reward $\{r_i = R(p, A_i)\}_{i=1}^N$ # obtain rewards for samples
    $\sigma \leftarrow std(\{r_i\}_{i=1}^N)$ # calculate standard deviation for reward
    if $\sigma \geq \sigma_0$ **then**
      $P_Q^+ \leftarrow p$
      if $max_i(r_i) \geq r_0$ **then**
        $A \leftarrow argmax_{A_i}(r_i)$ # get the solution with highest reward
        $D_A \leftarrow \{a_{1,2,\ldots,j}\}_{j=1}^{|A|}$ # extract partial solutions
        $D_A^* \leftarrow do_sample(D_A, \beta, \alpha)$ # sample partial solutions for train data assembly
        for partial solution p^* in D_A^* **do**
          $P_Q^+ \leftarrow p + p^*$
        end for
      end if
    end if
  end for
  $\pi_\theta \leftarrow GRPO(P_Q^+, \pi_\theta, R)$ # train the policy with the GRPO objective
end for

Output: π_θ

Algorithm 2 The value model's train data collection process.

Input: LLM policy π_θ, reward function (model) R, question (instruction) dataset Q, prompt dataset P_Q, max search iterations T, number of samples for each expansion n, exploration constant c, small constant that avoids zero-division ϵ, initial value v_0,

$D_{value} \leftarrow \varnothing$ # initialize value train set
for prompt p in P_Q **do**
  Initialize the search tree T_p
  for $t = 1, 2, \ldots, T$ **do**
    $S \leftarrow S_{init}$
    ———————***Node Selection***———————
    while S is expanded **do**
      $S \leftarrow argmax_{S' \in \text{children}(S)}(v_{S'} + c\sqrt{\frac{\ln N_S + 1}{N_{S'} + \epsilon}})$ # select child node based on UCT
    end while
    $v, m \leftarrow 0$ # for value calculation in backpropagation process
    if $S = (p, a_{1,2,\ldots,j})$ is not an end state **then**
      ———————***Node Expansion***———————
      Sample n traces $\{o_i = (a^{(i)}_{j+1,j+2,\ldots})\}_{i=1}^n \sim \pi_\theta(O|S)$ starting from S
      Build child nodes by taking actions $\{a^{(i)}_{j+1}\}_{i=1}^n$ # use the first action of sample traces to expand
      for node $S^{'}$ in children(S) **do**
        $v_{S'} \leftarrow v_0$
        $N_{S'}, U_{S'} \leftarrow 0$
      end for
      ———————***MC Rollout***———————
      Get rewards with reward function $\{r_i = R(p, a_{1,2,\ldots,j}, o_i)\}_{i=1}^n$
      $D_{value} \leftarrow \{((p, a_{1,2,\ldots,j}, o_i), r_i)\}_{i=1}^n$ # add value targets for end states to train set
      $v, m \leftarrow sum(\{r_i\}_{i=1}^n), n$
    else
      $r \leftarrow R(S)$ # get the reward for current end state
      $v, m \leftarrow r, 1$
    end if
    ———————***Value Backpropagation***———————
    for node $S^{'}$ from S to root **do**
      $N_{S'} \leftarrow N_{S'} + m$ # update visit count
      $U_{S'} \leftarrow U_{S'} + v$ # update sum of reward
      $v_{S'} \leftarrow \frac{U_{S'}}{N_{S'}}$ # update value estimation
    end for
  end for
  for node S in T_p **do**
    if S is expanded **then**
      $D_{value} \leftarrow (S, v_S)$ # add value targets for partial states to train set
    end if
  end for
end for
Return D_{value}

Output: D_{value}

A.5 Details for Compared Policy Training Methods

- **ReST-GRPO:** In addition to test-case pass rate, the training reward checks required output substrings and penalizes redundant characters, as shown in Eq. (7). We set $\omega_1 = 1e-3$, $\omega_2 = 1e-6$, $N = 30$, $\sigma_0 = 0.05$, $r_0 = 0.9$, $\beta = 0.5$, $\alpha = 0.95$, and rollout group $G = 8$. The original prompt is retained when it passes filtering; sampled nonempty prefixes are added as further online-GRPO starts.

Algorithm 3 The assisted decoding algorithm of VM-MCTS.

Input: LLM policy π_θ, Value model V_ϕ, original question (instruction) q, question prompt p_q, max search iterations T, number of samples for each expansion n, exploration constant c, small constant that avoids zero-division ϵ,

Initialize the search tree T_{p_q}
$D_{sol} \leftarrow \varnothing$ # initialize solution set
for $t = 1, 2, \ldots, T$ **do**
  $S \leftarrow S_{init}$
  ———————***Node Selection***———————
  while S is expanded **do**
    $S \leftarrow argmax_{S' \in \text{children}(S)}(v_{S'} + c\sqrt{\frac{\ln N_S + 1}{N_{S'} + \epsilon}})$ # select child node based on UCT
  end while
  $v, m \leftarrow 0$ # for value calculation in backpropagation process
  if $S = (p, a_{1,2,\ldots,j})$ is not an end state **then**
    ———————***Node Expansion***———————
    Sample n traces $\{o_i = (a^{(i)}_{j+1,j+2,\ldots})\}_{i=1}^n \sim \pi_\theta(O|S)$ starting from S
    $D_{sol} \leftarrow \{(a_{1,2,\ldots,j}, o_i)\}_{i=1}^n$ # add sample solutions to solution set
    Build child nodes by taking actions $\{a^{(i)}_{j+1}\}_{i=1}^n$ # use the first action of sample traces to expand
    ———————***Value Based MC Rollout***———————
    for node $S^{'}$ in children(S) **do**
      $v_{S'} \leftarrow V_\phi(S^{'})$ # get value estimation
      Set $N_{S'}$ to the number of times $S^{'}$ was visited during expansion phase
      $U_{S'} \leftarrow N_{S'} v_{S'}$
      $v \leftarrow v + U_{S'}$
    end for
    $m \leftarrow n$
  else
    $v, m \leftarrow v_S, 1$
  end if
  ———————***Value Backpropagation***———————
  for node $S^{'}$ from S to root **do**
    $N_{S'} \leftarrow N_{S'} + m$ # update visit count
    $U_{S'} \leftarrow U_{S'} + v$ # update sum of reward
    $v_{S'} \leftarrow \frac{U_{S'}}{N_{S'}}$ # update value estimation
  end for
end for
Get value (reward) estimation of every solution $A_i \in D_{sol}$: $\{r_i = V_\phi(p_q, A_i)\}_{i=1}^{|D_{sol}|}$
$A^* = argmax_{A_i}(r_i)$ # Best-of-N
Return A^*

Output: A^*

- **GRPO:** To validate the effectiveness of the proposed partial sampling methodology on GRPO, we involve naive GRPO as a baseline. For this baseline, the same reward function R_{GRPO} is adopted for training. The number of generations per group is also set to 8. In contrast, different from ReST-GRPO, questions in Q_{train} are directly used for training. This means that each question in Q_{train} will be used once within a single training iteration.
- **DAPO:** We use the same Qwen3-8B initialization and training data as the main policy comparison and implement DAPO dynamic sampling, which resamples and removes groups with zero reward variance. Following the DAPO configuration, we use $\epsilon_{\text{high}} = 0.28$, the decoupled GRPO loss, and mask truncated completions. Other shared optimization and rollout settings follow the main comparison.
- **E2H-C:** This controlled curriculum baseline follows the cosine schedule of E2H Reasoner [21]. Empirical pass rate from the shared $N = 30$ trajectories defines four quantile difficulty bins.

Sampling gradually shifts from easy to hard original prompts over 2,000 updates; no partial state is used, while reward, optimizer, and $G = 8$ match the component runs.

- **ReST-DPO:** We compare another ReST baseline ReST-DPO, which is similar to the ReST^{EM} [26] method, except that the policy update algorithm is replaced from SFT to DPO [22]. For ReST-DPO, we also perform solution sampling with an initial LLM policy on Q_{train} within each self-train iteration, with the number of samples per question set to 30. Subsequently, the base reward function R_{base} is deployed to obtain rewards for generated solutions, which are then used for the construction of preference pairs. Finally, we update the policy with the DPO objective, based on generated preference data.

$$R_{\text{GRPO}}(q, A) = R_{\text{base}}(q, A) + \omega_1 * \mathbb{I}(s \subseteq A) - \omega_2 * n_redundant_char(A), \\ \text{where } s \text{ is an essential substring, and } \omega_1, \omega_2 \text{ are weight parameters.} \tag{7}$$

A.6 Details for Compared Decoding and Verification Methods

We provide detailed descriptions and configurations of compared decoding and verification methods, i.e., VM-MCTS, ORM, PRM, and ORM-MCTS.

- **VM-MCTS:** For each policy, we train a corresponding VM with Algorithm 2 on Q_{train}. Value-data collection uses $T = 30$, $n = 5$, $c = 0.4$, $\epsilon = 0.1$, and $v_0 = 0$; the VM is initialized by adding a classifier head to the policy network. Test-time decoding uses $c = 0.1$. Main ReST-RL results and the matched controls use $T = 20, n = 5$; other budget experiments vary T as stated.
- **ORM:** We use the Skywork-Reward-Llama-3.1-8B-v0.2 ORM, which is trained on high-quality preference pairs sourced from publicly available data [14]. The output with the highest reward is returned.
- **PRM:** A Qwen2.5 PRM trained with an improved M-S method [42] is adopted to provide action-level reward. We use the minimum action-level reward for a single output as the final verification score. The output with the highest score is returned.
- **ORM-MCTS:** To assess the influence of MCTS on decoding accuracy, we include this baseline that also uses the Skywork ORM. We deploy a MCTS decoding algorithm similar to Algorithm 3, except that value estimation and final verification is based on the ORM.

A.7 Additional Experimental Results

Additional results for policy training. Different from training results in Table 1, we provide policy training results when using greedy decoding in test in Table 13. In addition, we also provide the training results of Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat in Table 14. To ensure that the improvements achieved are beyond noise, we conduct four independent replicate self-train experiments under identical conditions mentioned in Section 3.2. In Table 15, we report self-train results with std-dev for Qwen3-8B on every benchmark. It can be observed that ReST-GRPO maintains clear margins ($> 5\%$ Avg., clearly not within random noise) over all other baselines and across iterations, indicating its significant improvement.

Additional results for decoding and verification. Figure 5 provides per-benchmark results for the compared decoding and verification methods, and Table 16 reports averages for Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat. To control domain mismatch, we also train a Qwen3-8B ORM on Q_{train} with the same unit-test reward as the VM. Table 17 shows that this ORM obtains 54.9% with BoN and 55.0% with MCTS, versus 68.9% for the complete ReST-RL pipeline.

A.8 Discussions

Practical feasibility of the framework. Although ReST-RL improves reasoning performance through a two-stage pipeline, its practical feasibility should be assessed from an end-to-end perspective. For the self-training stage, Table 4 shows that, even when the pre-sampling cost is included, ReST-GRPO remains more efficient than naive GRPO in terms of wall-clock time and total GPU hours in the long run. This is because the main bottleneck of RL training lies in repeated online sampling and policy optimization, whereas the offline pre-sampling step is highly parallelizable and incurs only limited additional latency. More importantly, the filtered and partially reassembled training data

Table 13: Policy training results when using greedy decoding in test.

Model	Training Method	HumanEval	HumanEval+	MBPP	MBPP+	APPS-500	BCB	Avg.
Qwen3-8B	Base (0th iter.)	0.848	0.787	0.717	0.619	0.116	0.430	0.508
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.817	0.762	0.730	0.638	0.140	0.407	0.505
	GRPO (1st iter.)	0.829	0.793	0.743	0.640	0.347	0.445	0.574
	ReST-GRPO (1st iter.)	0.841	0.799	0.775	0.664	0.405	0.468	0.603
	ReST-DPO (2nd iter.)	0.823	0.774	0.741	0.656	0.153	0.416	0.517
	GRPO (2nd iter.)	0.829	0.787	0.757	0.656	0.412	0.448	0.594
	ReST-GRPO (2nd iter.)	0.878	0.829	0.794	0.698	0.552	0.482	**0.658**
Qwen2.5-Coder-7B-Instruct	Base (0th iter.)	0.909	0.841	0.828	0.714	0.302	0.419	0.592
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.909	0.841	0.831	0.714	0.340	0.416	0.601
	GRPO (1st iter.)	0.896	0.835	0.833	0.714	0.384	0.425	0.612
	ReST-GRPO (1st iter.)	0.896	0.848	0.860	0.730	0.436	0.468	0.643
	ReST-DPO (2nd iter.)	0.909	0.848	0.825	0.714	0.356	0.422	0.607
	GRPO (2nd iter.)	0.890	0.829	0.844	0.728	0.392	0.428	0.616
	ReST-GRPO (2nd iter.)	0.909	0.860	0.854	0.725	0.416	0.492	**0.646**
DS-Coder-6.7b-Instruct	Base (0th iter.)	0.768	0.695	0.751	0.659	0.242	0.346	0.506
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.780	0.707	0.746	0.661	0.245	0.348	0.510
	GRPO (1st iter.)	0.780	0.713	0.765	0.677	0.277	0.350	0.524
	ReST-GRPO (1st iter.)	0.787	0.707	0.767	0.685	0.287	0.366	0.532
	ReST-DPO (2nd iter.)	0.780	0.707	0.746	0.659	0.254	0.350	0.513
	GRPO (2nd iter.)	0.762	0.683	0.759	0.675	0.286	0.356	0.520
	ReST-GRPO (2nd iter.)	0.780	0.701	0.767	0.672	0.323	0.390	**0.543**
OpenCI-DS-6.7B	Base (0th iter.)	0.787	0.713	0.738	0.648	0.210	0.365	0.505
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.774	0.713	0.733	0.643	0.207	0.373	0.503
	GRPO (1st iter.)	0.780	0.713	0.735	0.656	0.312	0.365	0.530
	ReST-GRPO (1st iter.)	0.780	0.713	0.738	0.656	0.324	0.373	0.535
	ReST-DPO (2nd iter.)	0.768	0.701	0.738	0.646	0.218	0.360	0.501
	GRPO (2nd iter.)	0.780	0.720	0.728	0.643	0.308	0.361	0.526
	ReST-GRPO (2nd iter.)	0.774	0.713	0.749	0.664	0.324	0.387	**0.540**

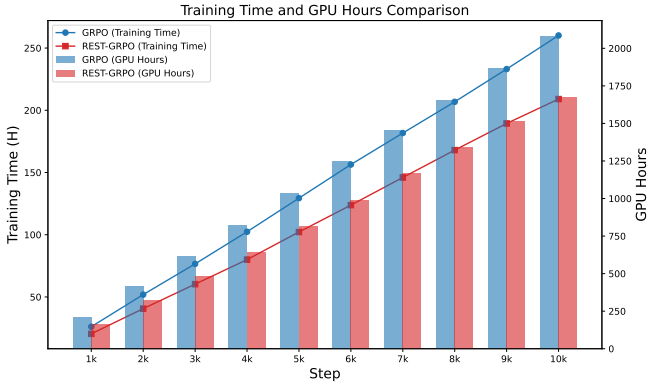


Figure 4: Training time and corresponding GPU-hour accounting for Llama-3-8B, comparing ReST-GRPO with naive GRPO. The training time is recorded every 1000 steps. Note that the time for pre-train sampling of ReST-GRPO is not included. See Table 4 for a full end-to-end training time comparison.

allow the policy to reach the same or higher performance levels in fewer effective optimization steps, which explains why the end-to-end cost is still reduced despite the extra pre-sampling stage. For inference, Table 18 gives the equal-sample view on the base policy: VM-MCTS is 1.8–2.6 times slower than ORM-BoN, depending on tree shape, while yielding higher APPS-500 accuracy. This comparison uses different verifiers and therefore measures the practical pipeline rather than isolating tree allocation. The causal control in Table 3(c) instead holds the ReST-GRPO policy and VM fixed and approximately matches mean wall-clock: VM-MCTS obtains 0.642 versus 0.619 for VM-BoN at

Table 14: Policy training results for Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat.

Model	Training Method	HumanEval	HumanEval+	MBPP	MBPP+	APPS-500	BCB	Avg.
Llama-3.1-8B-Instruct	Base	0.671	0.591	0.693	0.569	0.175	0.301	0.435
	Base (greedy)	0.689	0.610	0.664	0.545	0.213	0.325	0.448
	Below are results for the first training iteration							
	ReST-DPO	0.628	0.579	0.669	0.569	0.209	0.313	0.436
	GRPO	0.652	0.573	0.675	0.566	0.271	0.324	0.457
	ReST-GRPO	0.677	0.610	0.735	0.616	0.248	0.383	**0.488**
	ReST-DPO (greedy)	0.677	0.616	0.701	0.574	0.259	0.321	0.466
	GRPO (greedy)	0.683	0.634	0.706	0.579	0.329	0.329	0.490
	ReST-GRPO (greedy)	0.707	0.646	0.725	0.608	0.312	0.407	**0.516**
	Below are results for the second training iteration							
	ReST-DPO	0.671	0.598	0.672	0.556	0.249	0.299	0.449
	GRPO	0.671	0.598	0.701	0.585	0.284	0.341	0.476
	ReST-GRPO	0.665	0.598	0.693	0.587	0.235	0.418	**0.481**
	ReST-DPO (greedy)	0.677	0.616	0.698	0.574	0.272	0.325	0.470
	GRPO (greedy)	0.701	0.634	0.696	0.563	0.307	0.343	0.487
	ReST-GRPO (greedy)	0.720	0.659	0.741	0.624	0.342	0.425	**0.535**
CodeQwen1.5-7B-Chat	Base	0.805	0.732	0.810	0.677	0.148	0.348	0.502
	Base (greedy)	0.854	0.787	0.831	0.706	0.163	0.374	0.532
	Below are results for the first training iteration							
	ReST-DPO	0.823	0.774	0.759	0.659	0.167	0.353	0.507
	GRPO	0.829	0.750	0.788	0.653	0.244	0.365	0.530
	ReST-GRPO	0.854	0.793	0.804	0.690	0.270	0.389	**0.557**
	ReST-DPO (greedy)	0.854	0.787	0.817	0.696	0.185	0.373	0.534
	GRPO (greedy)	0.866	0.799	0.817	0.698	0.272	0.401	0.566
	ReST-GRPO (greedy)	0.890	0.835	0.831	0.714	0.294	0.447	**0.594**
	Below are results for the second training iteration							
	ReST-DPO	0.841	0.780	0.799	0.672	0.176	0.339	0.515
	GRPO	0.878	0.823	0.815	0.672	0.255	0.375	0.556
	ReST-GRPO	0.878	0.816	0.828	0.706	0.278	0.425	**0.579**
	ReST-DPO (greedy)	0.860	0.787	0.836	0.709	0.211	0.362	0.542
	GRPO (greedy)	0.866	0.799	0.828	0.701	0.296	0.401	0.574
	ReST-GRPO (greedy)	0.902	0.835	0.836	0.725	0.308	0.479	**0.609**

Table 15: Reinforce self training results for Qwen3-8B with standard deviation.

Model	Training Method	HumanEval	HumanEval+	MBPP	MBPP+	APPS-500	BCB	Avg.
Qwen3-8B	Base (0th iter.)	0.829±0.018	0.780±0.019	0.717±0.003	0.622±0.002	0.118±0.003	0.418±0.007	0.503±0.008
	Below are results for sequential training iterations							
	ReST-DPO (1st iter.)	0.854±0.013	0.799±0.006	0.730±0.005	0.627±0.011	0.152±0.015	0.434±0.010	0.523±0.011
	GRPO (1st iter.)	0.799±0.018	0.750±0.030	0.754±0.016	0.651±0.013	0.346±0.006	0.439±0.004	0.566±0.012
	ReST-GRPO (1st iter.)	0.872±0.012	0.817±0.006	0.780±0.005	0.672±0.003	0.377±0.008	0.469±0.013	0.604±0.009
	ReST-DPO (2nd iter.)	0.841±0.006	0.811±0.012	0.741±0.003	0.653±0.007	0.153±0.011	0.429±0.007	0.526±0.008
	GRPO (2nd iter.)	0.829±0.006	0.787±0.006	0.757±0.006	0.667±0.014	0.403±0.007	0.436±0.004	0.590±0.007
	ReST-GRPO (2nd iter.)	0.860±0.012	0.805±0.012	0.802±0.006	0.690±0.010	0.565±0.022	0.476±0.007	**0.655**±0.012

Table 16: Average results of ReST-RL and different verification methods on all benchmarks for Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat.

Method	Llama-3.1-8B-Instruct	CodeQwen1.5-7B-Chat
Base	0.435	0.502
ORM	0.480	0.542
PRM	0.466	0.526
ORM-MCTS	0.481	0.545
VM-MCTS	0.519	0.599
ReST-RL	**0.556**	**0.616**

about 41.5 seconds per prompt. VM-MCTS is thus most appropriate when quality warrants additional latency, rather than strict real-time settings.

Output diversity after training. A natural concern for methods that bias training toward high-reward trajectories is whether they may reduce exploration and lead to diversity collapse. The diversity

Table 17: Detailed inference-time verification results for Qwen3-8B. To ensure domain parity, we include two extra baselines that adopt an ORM trained on the same dataset Q_{train}.

Model	Method	HumanEval	HumanEval+	MBPP	MBPP+	APPS-500	BCB	Avg.
Qwen3-8B	Base	0.829	0.780	0.717	0.622	0.118	0.418	0.503
	ORM (Skywork)	0.841	0.787	0.746	0.661	0.165	0.441	0.531
	PRM (Qwen)	0.860	0.817	0.714	0.614	0.138	0.423	0.516
	ORM (Skywork) + MCTS	0.854	0.805	0.746	0.661	0.193	0.424	0.538
	ORM (Trained)	0.854	0.823	0.746	0.643	0.214	0.449	0.549
	ORM (Trained) + MCTS	0.860	0.826	0.743	0.648	0.221	0.441	0.550
	VM-MCTS	0.866	0.829	0.775	0.675	0.415	0.471	0.615
	ReST-RL	0.866	0.829	0.817	0.704	0.642	0.506	**0.689**

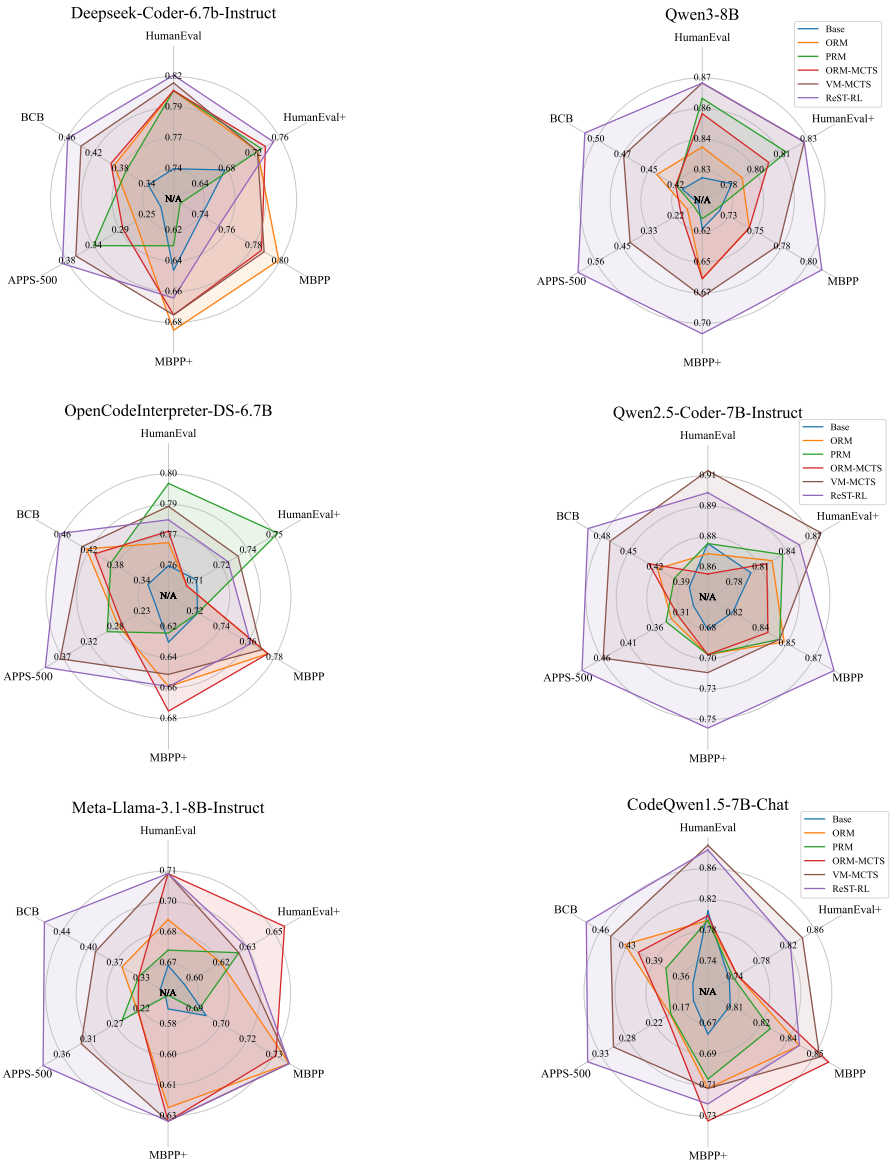


Figure 5: Performance of different base LLM policies when using ReST-RL and verification methods on all benchmarks. All verification is based on 100 samples, with the sampling temperature set to 0.7.

results in Table 19 do not support this concern. ReST-GRPO attains a Self-BLEU score close to the base model and lower than GRPO, indicating that the proposed training procedure does not simply force the policy into a narrow mode. This observation is consistent with the design of ReST-GRPO:

Table 18: Equal-sample test-time assessment on APPS-500 using the base Qwen3-8B policy. Each approach generates 100 candidates. BoN uses the coding ORM reported in Table 17, whereas VM-MCTS uses its corresponding VM; this table reports the practical sample-matched trade-off rather than isolating tree allocation.

Method	ORM-BoN ($N = 100$)	VM-MCTS ($n = 5, T = 20$)	VM-MCTS ($n = 10, T = 10$)	VM-MCTS ($n = 20, T = 5$)
Avg. Time (s) / Prompt	16.0	41.5	36.4	28.6
APPS-500	0.214	0.415	0.355	0.327

Table 19: Average Self-BLEU score on APPS-500 for models trained with different approaches. For each prompt, we sample 8 answers to calculate the score. Note that a higher Self-BLEU indicates lower output diversity.

Method	Base	GRPO	ReST-DPO	ReST-GRPO
Avg. Self-BLEU (APPS)	0.610	0.620	0.602	0.613

the method does not deterministically replay a single trajectory, but instead reshapes the training distribution by combining prompt-level filtering with partial-state sampling. As a result, optimization is biased toward promising regions of the search space without eliminating variability in subsequent rollouts. If the gains arose mainly from collapse onto a narrow set of patterns, Self-BLEU would increase sharply. Instead, diversity is preserved while the controlled coding results improve. The preliminary out-of-domain results in Table 5 provide additional transfer evidence.

Action granularity. In this work, we define an action as a single line of text. We view this choice as a practical compromise between expressiveness, generality, and search efficiency. Compared with token-level actions, line-level actions substantially reduce the branching complexity of tree search and make value estimation more stable, since each transition corresponds to a semantically more meaningful unit. This consideration is also consistent with recent long-horizon code reasoning studies such as LSR-MCTS [17], which show that salient attention spikes often emerge around line boundaries and suggest that line-level units better capture long-range dependencies in code generation. Compared with coarser alternatives such as full reasoning steps or code blocks, line-level actions are easier to apply across domains because they do not require domain-specific segmentation rules or additional annotations. Recent inference-scaling work also supports this view: DISC [12] argues that newline-delimited units often correspond to semantically coherent reasoning segments, while ReSCALE [30] shows that using delimiter-terminated spans instead of token-level actions yields better compute scaling for search-based reasoning. The same line-level formulation can be applied without domain-specific segmentation rules, and our math/science results provide preliminary evidence that it remains useful beyond code. It is also aligned with process-supervision practice: PRM800K-style steps are commonly organized line by line [13]. We do not claim that this granularity is universally optimal; adaptive or task-specific action abstractions remain an important direction.

Comparison with other tree search methods. As summarized in Table 20, VM-MCTS differs from related tree-search methods in how its guidance signal is trained and used. PPO-MCTS [16] learns policy and value jointly, whereas we train a dedicated static-policy VM after policy optimization. Math-Shepherd [31] and ReST-MCTS* [39] focus on process-oriented supervision and search-based data generation, whereas our target is expected terminal reward for a partial state. LATS [44] instead emphasizes interaction, reflection, and external feedback. Moreover, the motivation for introducing VM-MCTS is not merely to add another MCTS variant, but to complement the weakness of GRPO-style policy optimization. GRPO is efficient, yet it removes explicit reward feedback learning from the RL loop, which may weaken fine-grained guidance during both training and inference. Our framework reintroduces that missing signal in a lightweight manner. ReST-GRPO improves the policy-induced trajectory distribution, and VM-MCTS exploits this improved distribution to learn value estimates that are critical at inference. In this sense, the comparison with existing MCTS-based methods also clarifies the conceptual contribution of ReST-RL. It is an attempt to reconnect policy optimization and value-guided reasoning without returning to the full cost of online actor-critic learning.

Table 20: Design comparison with representative MCTS-based approaches.

Feature	ReST-RL (VM-MCTS)	PPO-MCTS	ReST-MCTS*	LATS
Guidance signal	State-value model trained on self-collected MCTS data	Token-level value network from PPO	Process reward model trained with search-derived targets	Environment feedback and self-reflection
Training coupling	Static-policy VM training after policy optimization	Joint policy–value optimization	Iterative self-training with PRM-guided search	Agent search without policy–value training
Action granularity	Line-level	Typically token-level	Reasoning-step level	Task/agent-action level
Primary use	Value-guided allocation and final selection	Value-guided text decoding	Search-based data generation	Interactive reasoning and planning

A.9 Related Work

LLMs for reasoning and code synthesis. Recent advancements have facilitated the growing application of LLMs to reasoning tasks like coding [11; 5; 27; 43] and math [20; 25]. To assist LLMs on code synthesis, previous approaches use test cases to provide reward signals [41]. However, though demonstrating notable performance on elementary programming tasks, LLMs continue to struggle with complex ones, such as those found in competitive programming platforms like Codeforces [6].

Selective self-training, curriculum, and policy optimization. Self-training methods such as STaR, ReST, and ReST-EM sample from the current policy and train on filtered complete responses [37; 4; 26]. Difficulty-aware methods instead control which original problems are presented: E2H Reasoner uses an easy-to-hard cosine curriculum [21], while DAPO dynamically resamples groups with nonzero reward variance [36]. ReST-GRPO shares their goal of improving weak training signals but intervenes at a different level. Selected partial states become new online-GRPO contexts and their suffixes are re-sampled, rather than using complete accepted responses as imitation targets or only rescheduling original prompts. Table 11 directly compares these levels of intervention.

Reasoning-time search. Prompting methods such as Chain-of-Thought [32] and tree search methods such as Tree-of-Thought [35] improve decoding without matched policy updates. MCTS-based methods further allocate computation using rewards, learned values, or reflection [16; 39; 34; 38]. Our controlled VM-BoN comparison holds both policy and verifier fixed, isolating the benefit of using the learned VM during allocation from its use only at final selection.

Output verification with reward models. Outcome reward models score complete outputs, whereas process reward models provide intermediate feedback [29; 13]. PRMs can improve verification but require high-quality process targets. Monte-Carlo supervision reduces annotation cost [31; 39], although the resulting targets can be noisy [42]. We use the term Value Model because our target is explicitly $V^{\pi}(s)$—expected terminal reward from a partial state under a static policy—rather than step correctness.

A.10 Limitations

First, ReST-RL does not study repeated multi-round coupling of an updated policy and VM beyond the reported two-stage pipeline. Second, VM-MCTS has higher latency than BoN at equal candidate counts. The approximately latency-matched control is reported on Qwen3-8B and APPS-500 and does not cover every deployment regime. Third, the controlled component and matched-VM runs use one training seed and one base model, although the main ReST-GRPO result is supported by four independent runs and multiple model families. Finally, the math/science experiments use Qwen3-8B

and show transfer without target-domain tuning, but they do not establish a universal reasoning value model.

A.11 Table of Hyperparameters

Table 21: Main hyperparameters and default values for ReST-GRPO.

Hyperparameter	N	σ_0	r_0	α	β
Meaning	Number of solution samples	SD threshold	Reward threshold	Sampling exponent factor	Sample ratio
Default Value	30	0.05	0.9	0.95	0.5

Table 22: Main hyperparameters and default values for VM-MCTS.

Hyperparameter	T	n	c	ϵ
Meaning	Search iterations	Number of branches expanded	Exploration constant	Constant avoiding zero-division in UCT
Default Value	30	5	0.4	0.1

A.12 Mathematical Proof for Value Estimation

In this part we prove that our proposed value-based MC rollout tends to have more precise estimation of partial state's value, in terms of variance. Suppose we are estimating the value of partial state $S_i = (q, a_{1,2,\dots,i})$ for a fixed policy π. We are allowed to use two verifier models, a value model V_ϕ and a reward model R_λ (which is allowed to be identical to V_ϕ), while the actual reward function R and value function V are unknown (since it's test time). As a reasonable approximation, we assume that the two models can be regarded as two unbiased estimator variables that have the same variance when predicting, along with some conditions of independence:

$$\begin{aligned} & V_\phi(S) = V(S) + \epsilon_V, \\ & R_\lambda(S_{end}) = R(S_{end}) + \epsilon_R, \\ s.t.\ & \mathbb{D}[\epsilon_V] = \mathbb{D}[\epsilon_R], \mathbb{E}[\epsilon_V] = \mathbb{E}[\epsilon_R] = 0, \\ & \epsilon_V \text{ is independent of } S, \\ & \epsilon_R \text{ is independent of } S_{end}. \end{aligned} \tag{8}$$

Now, we consider two approaches for value estimation, with a limit of n simulations. For value-based MC rollout, it does simulation with the random variable S_{i+1}, i.e. the state reached by π performing a single action. It estimates $V(S_i)$ by:

$$\begin{aligned} & \hat{V}_v(S_i) = \frac{1}{n}\sum_{j=1}^{n} V_\phi(S_{i+1}^{(j)}), \\ s.t.\ & S_{i+1}^{(1)}, S_{i+1}^{(2)}, \dots, S_{i+1}^{(n)} \overset{\text{i.i.d.}}{\sim} \pi(S_{i+1}|S_i). \end{aligned} \tag{9}$$

On the other hand, regular complete rollout does simulation with the random variable S_{end}, i.e. the end state reached after a complete trace is generated. It estimates $V(S_i)$ by:

$$\begin{aligned} & \hat{V}_r(S_i) = \frac{1}{n}\sum_{j=1}^{n} R_\lambda(S_{end}^{(j)}), \\ s.t.\ & S_{end}^{(1)}, S_{end}^{(2)}, \dots, S_{end}^{(n)} \overset{\text{i.i.d.}}{\sim} \pi(S_{end}|S_i). \end{aligned} \tag{10}$$

We now state the proposition to be proved:

$$\mathbb{E}[\hat{V}_r(S_i)] = \mathbb{E}[\hat{V}_v(S_i)] = V(S_i), \tag{11}$$

and

$$\mathbb{D}[\hat{V}_r(S_i)] \geq \mathbb{D}[\hat{V}_v(S_i)]. \tag{12}$$

If the statements are true, we can conclude that $\hat{V}_v(S_i)$ is a better estimator than $\hat{V}_r(S_i)$ due to its smaller variance, which justifies our original claim. Now, we provide the proof for Eq. (11).

Proof. First, we notice that by Eq. (8):

$$\begin{aligned} \hat{V}_r(S_i) &= \frac{1}{n}\sum_{j=1}^{n} R_\lambda(S_{end}^{(j)}) \\ &= \frac{1}{n}\sum_{j=1}^{n} (R(S_{end}^{(j)}) + \epsilon_R) \\ &= \epsilon_R + \frac{1}{n}\sum_{j=1}^{n} R(S_{end}^{(j)}) \end{aligned}$$

Substituting the above equation into the expression of expectation, we derive that:

$$\begin{aligned}
\mathbb{E}[\hat{V}_r(S_i)] &= \mathbb{E}[\epsilon_R + \frac{1}{n}\sum_{j=1}^{n} R(S_{end}^{(j)})] \\
&= \mathbb{E}[\epsilon_R] + \frac{1}{n}\sum_{j=1}^{n} \mathbb{E}[R(S_{end}^{(j)})] \\
&= \frac{1}{n}\sum_{j=1}^{n} \mathbb{E}[R(S_{end})|S_i] \\
&= \mathbb{E}[R(S_{end})|S_i] \\
&= V(S_i)
\end{aligned}$$

Similarly, we can know that:

$$\begin{aligned}
\hat{V}_v(S_i) &= \frac{1}{n}\sum_{j=1}^{n} V_\phi(S_{i+1}^{(j)}) \\
&= \frac{1}{n}\sum_{j=1}^{n} (V(S_{i+1}^{(j)}) + \epsilon_V) \\
&= \epsilon_V + \frac{1}{n}\sum_{j=1}^{n} V(S_{i+1}^{(j)})
\end{aligned}$$

Thus, we can finally derive the desired properties:

$$\begin{aligned}
\mathbb{E}[\hat{V}_v(S_i)] &= \mathbb{E}[\epsilon_V + \frac{1}{n}\sum_{j=1}^{n} V(S_{i+1}^{(j)})] \\
&= \mathbb{E}[\epsilon_V] + \frac{1}{n}\sum_{j=1}^{n} \mathbb{E}[V(S_{i+1}^{(j)})] \\
&= \frac{1}{n}\sum_{j=1}^{n} \mathbb{E}[V(S_{i+1})|S_i] \\
&= \mathbb{E}[V(S_{i+1})|S_i] \\
&= V(S_i)
\end{aligned}$$

□

Next, we provide proof for Eq. (12) as follows.

Proof. Using the conclusions made by the proof above, we can derive the required property:

$$
\begin{aligned}
&\mathbb{D}[\hat{V}_r(S_i)] \geq \mathbb{D}[\hat{V}_v(S_i)] \\
\Longleftarrow\ &\mathbb{D}[\epsilon_R + \frac{1}{n}\sum_{j=1}^{n} R(S_{end}^{(j)})] \geq \mathbb{D}[\epsilon_V + \frac{1}{n}\sum_{j=1}^{n} V(S_{i+1}^{(j)})] \\
\Longleftarrow\ &\mathbb{D}[\epsilon_R] + \mathbb{D}[\frac{1}{n}\sum_{j=1}^{n} R(S_{end}^{(j)})] \geq \mathbb{D}[\epsilon_V] + \mathbb{D}[\frac{1}{n}\sum_{j=1}^{n} V(S_{i+1}^{(j)})] \\
\Longleftarrow\ &\frac{1}{n^2}\mathbb{D}[\sum_{j=1}^{n} R(S_{end}^{(j)})] \geq \frac{1}{n^2}\mathbb{D}[\sum_{j=1}^{n} V(S_{i+1}^{(j)})] \\
\Longleftarrow\ &\sum_{j=1}^{n} \mathbb{D}[R(S_{end}^{(j)})] \geq \sum_{j=1}^{n} \mathbb{D}[V(S_{i+1}^{(j)})] \\
\Longleftarrow\ &\mathbb{D}[R(S_{end})|S_i] \geq \mathbb{D}[V(S_{i+1})|S_i] \\
\Longleftarrow\ &\mathbb{E}[(R(S_{end}))^2|S_i] - (\mathbb{E}[R(S_{end})|S_i])^2 \geq \mathbb{E}[(V(S_{i+1}))^2|S_i] - (\mathbb{E}[V(S_{i+1})|S_i])^2 \\
\Longleftarrow\ &\mathbb{E}[(R(S_{end}))^2|S_i] \geq \mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[(V(S_{i+1}))^2|S_i] \\
\Longleftarrow\ &\mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[\mathbb{E}[(R(S_{end}))^2|S_{i+1}]|S_i] \geq \mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[(\mathbb{E}[R(S_{end})|S_{i+1}])^2|S_i] \\
\Longleftarrow\ &\mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[\mathbb{E}[(R(S_{end}))^2|S_{i+1}]] \geq \mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[(\mathbb{E}[R(S_{end})|S_{i+1}])^2] \\
\Longleftarrow\ &\mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[\mathbb{E}[(R(S_{end}))^2|S_{i+1}] - (\mathbb{E}[R(S_{end})|S_{i+1}])^2] \geq 0 \\
\Longleftarrow\ &\mathbb{E}_{S_{i+1}\sim\pi(S_{i+1}|S_i)}[\mathbb{D}[R(S_{end})|S_{i+1}]] \geq 0 \\
\Longleftarrow\ &\mathbb{D}[R(S_{end})|S_{i+1}] \geq 0
\end{aligned}
$$

□

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: We have included discussion of limitations in Appendix A.10.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: We have provided complete proof.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Our results are fully reproducible.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We have provided sufficient instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).

- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We have specified the details in Appendix A.3 and A.11.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We have reported experimental results with standard deviation in Table 15.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We have disclosed sufficient information about resources such as GPU-hours for experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our work proposes a general framework for LLM reasoning optimization and does not directly relate to negative applications.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All are properly credited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: All are well documented.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.