Segment Policy Optimization: Effective Segment-Level Credit Assignment in RL for Large Language Models

Yiran Guo[†], **Lijie Xu**[†*], **Jie Liu**[†*], **Dan Ye**[†], **Shuang Qiu**[‡]
[†]Institute of Software, Chinese Academy of Sciences
[†]University of Chinese Academy of Sciences
[‡]City University of Hong Kong
`{guoyiran23, xulijie, ljie, yedan}@otcaix.iscas.ac.cn`
`shuanqiu@cityu.edu.hk`

Abstract

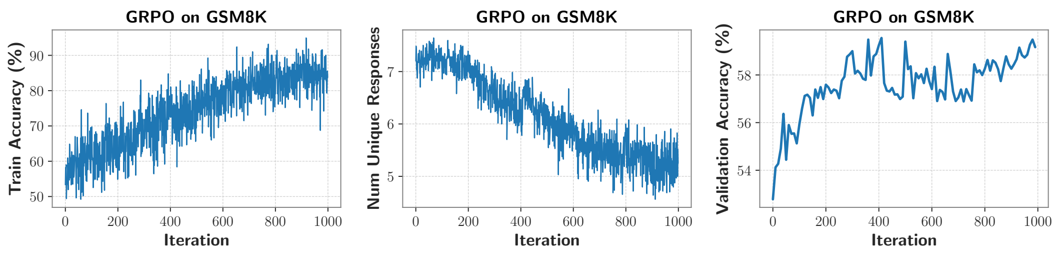
Enhancing the reasoning capabilities of large language models effectively using reinforcement learning (RL) remains a crucial challenge. Existing approaches primarily adopt two contrasting advantage estimation granularities: Token-level methods (e.g., PPO) aim to provide the fine-grained advantage signals but suffer from inaccurate estimation due to difficulties in training an accurate critic model. On the other extreme, trajectory-level methods (e.g., GRPO) solely rely on a coarse-grained advantage signal from the final reward, leading to imprecise credit assignment. To address these limitations, we propose *Segment Policy Optimization (SPO)*, a novel RL framework that leverages segment-level advantage estimation at an intermediate granularity, achieving a better balance by offering more precise credit assignment than trajectory-level methods and requiring fewer estimation points than token-level methods, enabling accurate advantage estimation based on Monte Carlo (MC) without a critic model. SPO features three components with novel strategies: **(1)** flexible segment partition; **(2)** accurate segment advantage estimation; and **(3)** policy optimization using segment advantages, including a novel probability-mask strategy. We further instantiate SPO for two specific scenarios: **(1)** *SPO-chain* for short chain-of-thought (CoT), featuring novel cutpoint-based partition and chain-based advantage estimation, achieving 6-12 percentage point improvements in accuracy over PPO and GRPO on GSM8K. **(2)** *SPO-tree* for long CoT, featuring novel tree-based advantage estimation, which significantly reduces the cost of MC estimation, achieving 7-11 percentage point improvements over GRPO on MATH500 under 2K and 4K context evaluation. We make our code publicly available at `https://github.com/AIFrameResearch/SPO`.

1 Introduction

Reinforcement learning (RL) has become the cornerstone of training state-of-the-art reasoning large language models (LLMs) like OpenAI o1 [Jaech et al., 2024], DeepSeek R1 [Guo et al., 2025], Kimi K1.5 [Team et al., 2025], and Qwen3 [Yang et al., 2025]. These models demonstrate RL's unique ability to cultivate advanced reasoning capabilities, particularly in complex (STEM-related) tasks. However, achieving both effectiveness and efficiency in RL training hinges on overcoming a fundamental challenge: *the credit assignment, i.e., accurately attributing success or failure to individual actions within a sequence* [Sutton et al., 1998]. In the context of LLMs, the challenge is even greater due to sparse and delayed rewards. Advantage estimation is commonly employed to

[*] Corresponding Authors. Lijie Xu, Jie Liu, and Dan Ye are also affiliated with Key Laboratory of System Software (Chinese Academy of Sciences) and University of Chinese Academy of Sciences, Nanjing.

perform credit assignment in RL. Different existing RL methods primarily vary in the granularity of their advantage estimation, typically operating at two extremes, each of which has its own limitations.



(a) Train Accuracy. (b) Train Num Unique Responses. (c) Validation Accuracy.

Figure 1: GRPO on GSM8K exhibits rapid overfitting: while training accuracy improves steadily (Fig. 1a), the unique response number constantly drops (Fig. 1b), and validation accuracy saturates early (Fig. 1c).

Fine-grained token-level methods like Proximal Policy Optimization (PPO) [Schulman et al., 2017] use a critic model to estimate advantages for each token. However, accurately predicting state values poses a particular challenge in LLM training, due to the significant variability among states conditioned on different prompts and the limited per-prompt data to effectively train the critic model. Empirical findings by Kazemnejad et al. [2024] provide extensive evidence that this difficulty causes the critic model to produce unreliable value predictions, leading to suboptimal credit assignment in practice. Additionally, PPO requires maintaining a separate critic model, typically as large as the actor, which doubles the memory footprint and computational cost, making it less efficient for large-scale training. At the other extreme, coarse-grained trajectory-level methods such as Group Relative Policy Optimization (GRPO) [Shao et al., 2024] bypass the critic model and compute a single advantage for the entire generated sequence based solely on the final outcome. While this approach is computationally efficient and unbiased, it leads to imprecise credit assignment over long sequences [Kazemnejad et al., 2024]. Applying a single advantage signal to a large number of tokens makes it challenging for the model to identify which specific actions contribute positively or negatively, resulting in the model failing to reward partial progress or learning inefficient solution paths [Qu et al., 2025]. Moreover, our experimental results, consistent with a concurrent work [Yu et al., 2025], find that GRPO can rapidly overfit on a fixed training set, with the number of unique responses decreasing and the performance on the validation set saturating early (see Figure 1).

To overcome the limitations of both token-level and trajectory-level methods, we propose ***S**egment **P**olicy **O**ptimization (SPO)*, a novel RL framework focusing on mid-grained, segment-level advantage estimation. Instead of assigning credit for each token or only at the end of a trajectory, SPO partitions the generated sequence into contiguous segments and estimates advantages at this intermediate granularity. This segment-level estimation offers several key benefits: **(1) Improved credit assignment:** Segment-level feedback provides more localized information than trajectory-level methods, allowing credit assignment to shorter segments. This finer granularity enables the model to reward partial progress even for ultimately unsuccessful responses and penalize redundancy or unnecessary portions within successful responses. **(2) More accurate advantage estimation:** Compared to token-level advantages, segment-level advantages involve fewer estimation points. This enables SPO to leverage effective Monte Carlo (MC) sampling, yielding accurate and unbiased advantage estimation directly from the policy, thus eliminating the need for an additional, unstable critic model. **(3) Flexibility and adaptability:** Our segment partition method can be arbitrarily defined without requiring semantic completeness, offering flexible adjustment of granularity from token-level to trajectory-level, also making it adaptable to a wide range of tasks.

Our SPO framework contains three key components: **(1) Flexible segment partition**, **(2) Segment advantage estimation via MC**, and **(3) Policy optimization using segment advantages**. This modular design allows for various strategies to be implemented within each component, making the framework highly adaptable to different tasks and scenarios. We further instantiate SPO with two specialized instances tailored for different reasoning scenarios. For short chain-of-thought (CoT), we introduce SPO-chain, which employs a cutpoint-based segment partition strategy and chain-based segment advantage estimation. For long CoT, we introduce SPO-tree, featuring a novel tree-based segment advantage estimation strategy specifically designed to significantly improve MC sampling efficiency. Additionally, we propose a novel probability-mask optimization strategy that selectively compute the loss for key tokens instead of all tokens within a segment, which can be applied to either SPO-chain or SPO-tree to further enhance credit assignment. Our experimental evaluations demonstrate the effectiveness of the SPO framework and its specialized instantiation. For short CoT, SPO-chain achieves 6–12 percentage point accuracy improvements over PPO and GRPO on GSM8K. For long CoT, SPO-tree achieves 7–11 percentage point accuracy improvements over GRPO on MATH500 under 2K and 4K context evaluation. Our major contributions are summarized as follows:

1. We propose Segment Policy Optimization (SPO), a novel segment-level RL training framework for LLMs. SPO introduces mid-grained advantage estimation to overcome the respective limitations of token-level and trajectory-level methods, and features a modular architecture.
2. We contribute to introducing several key techniques integrated within the SPO framework, including cutpoint-based segment partition, tree-based segment advantage estimation, and a policy optimization strategy utilizing probability masks for both chain and tree structures.
3. Building on the proposed SPO framework, we introduce two specialized instantiations: SPO-chain and SPO-tree, for short and long CoT scenarios, respectively, and demonstrate their effectiveness through extensive experiments on mathematical reasoning benchmarks, GSM8K and MATH.

2 Related Work

Boosting LLM's Reasoning Capacity. Various approaches have been explored to strengthen the reasoning abilities of LLMs. Several methods focus on using high-quality data for training. For instance, RFT [Yuan et al., 2024, 2023] fine-tunes the pretrained base model using correct samples generated by a supervised fine-tuned model. Similarly, RestEM [Singh et al., 2023] adopts an iterative self-training strategy, repeatedly retraining the base model on high-quality reasoning traces produced by previously obtained checkpoints. Preference-based methods like DPO [Rafailov et al., 2023, 2024] optimize models by contrasting correct and incorrect reasoning outputs. Search-guided approaches, like Monte Carlo Tree Search (MCTS) [Zhang et al., 2024, Chen et al., 2024], help models discover improved reasoning paths during inference. Another significant direction involves RL frameworks, which typically adopt policy gradient-based methods and differ mainly in advantage estimation granularity. For example, GRPO [Shao et al., 2024] and RLOO [Ahmadian et al., 2024] estimate trajectory-level advantages, while PPO [Schulman et al., 2017, Ouyang et al., 2022] estimates token-level advantages using a dedicated critic model. Recent work [Kazemnejad et al., 2024] highlights that accurately training the critic in PPO is challenging and proposes VinePPO, a critic-free approach that partitions the reasoning trajectory into discrete steps using heuristic rules (e.g., line breaks) and estimates step-level advantages through MC sampling. Compared to VinePPO, our segment-level framework adopts a more general concept of segmentation, allowing arbitrary partitions without enforcing semantic coherence. This enables us to freely adjust the granularity between token-level and trajectory-level, and facilitates adaptation to broader tasks. Moreover, we introduce a novel tree-based segment advantage estimation method within our framework, significantly reducing the computational cost associated with MC-based advantage estimation, allowing our method to scale effortlessly to the long CoT scenario. Notably, VinePPO can be viewed as a special case within our more general SPO framework. PRIME [Cui et al., 2025] simultaneously trains a process reward model using outcome rewards during policy training, providing process rewards to better guide policy optimization, while our work does not rely on reward models. Most recently, several works have proposed improvements upon the original GRPO method, like DAPO [Yu et al., 2025] and Dr. GRPO [Liu et al., 2025], based on the trajectory-level advantage estimation.

Fine-Grained Reward Signals. A common approach in prior work assigns a single binary reward to the final output of LLMs by comparing it against the ground truth. However, this sparse reward provides limited guidance, resulting in the difficulty of credit assignment during training. To address this challenge, Lightman et al. [2023] initially proposed "Process Reward", which involves manually judging correctness at every intermediate reasoning step. Subsequent works, such as Wang et al. [2023], Luo et al. [2024], and Ma et al. [2023], automated the process reward generation without manual labeling through rollouts. These methods focus on training an external Process Reward Model (PRM) to provide intermediate rewards for policy optimization and enable Best-of-N (BoN) selection during inference. In practice, when applying PRM for policy optimization, existing approaches typically combine step-level rewards with the final binary correctness rewards and still employ standard RL algorithms such as PPO or GRPO for training. In contrast, our approach differs fundamentally by improving the RL optimization algorithm itself, rather than relying on external reward models. We introduce an MC-based estimation method to directly compute segment-level advantages from the current policy, aiming at potentially reducing gradient estimation variance and enabling finer-grained credit assignment during optimization. This strategy helps stabilize training and leads to more effective policy updates compared to using sparse, trajectory-level rewards alone. Notably, our MC advantage estimation framework remains fully compatible with the PRM approach: intermediate process reward signals generated via PRMs could be integrated with our trajectory evaluations. Such integration may combine the strengths of both methodologies, providing further potential improvements in overall model performance. In Appendix B, we provide an analysis of integrating the process reward into our framework.

3 Preliminary

Language Model as an MDP. Language generation tasks can be modeled as a Markov Decision Process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, \mathcal{R})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathbb{P}$ represents transition dynamics, and R is the reward function. Specifically, at each time step t, the state $s_t \in \mathcal{S}$ consists of the prompt tokens $\boldsymbol{x}$ along with previously generated tokens $\boldsymbol{y}_{<t} = [y_1, \dots, y_{t-1}]$, that is, $s_t = [\boldsymbol{x}, \boldsymbol{y}_{<t}]$. The action $a_t \in \mathcal{A}$ corresponds to selecting the next token y_t. The decision-making process follows a policy $\pi_\theta(a_t|s_t)$, parameterized by θ, which defines the probability of the next token conditioned on the current state. The transition dynamics $\mathbb{P}$ are deterministic: given state s_t and action (token) a_t, the next state s_{t+1} is obtained by concatenating the selected token to the current state: $s_{t+1} = \mathbb{P}(s_t, a_t) = [s_t, a_t]$. The reward function assigns 0 intermediate rewards, providing only a sparse binary reward $\mathcal{R}(\boldsymbol{x}, \boldsymbol{y}) = 1$ or 0 at episode termination, where $\boldsymbol{y} = [y_1, y_2, \dots, y_T]$ is the full generated response, indicating whether the generated response matches the ground truth or not. The value function $V(s)$ represents the expected cumulative reward from state s. The state-action value function $Q(s, a)$ corresponds to the expected cumulative reward from choosing action a at state s. The advantage function $A(s, a)$, defined as $Q(s, a) - V(s)$, measures the improvement in expected reward realized by taking an action a at state s. Under deterministic transition dynamics and sparse binary rewards, the advantage function simplifies to $A(s, a) = V(s') - V(s)$, where $s' = \mathbb{P}(s, a)$ is the next state reached deterministically from state s after taking action a.

Proximal Policy Optimization. Proximal policy optimization (PPO) introduces a clipped surrogate objective to enhance training stability and efficiency by constraining policy updates around the previously established policy. PPO optimizes the following objective

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}, \boldsymbol{y}\sim\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{x})} \left[\frac{1}{|\boldsymbol{y}|} \sum_{t=1}^{|\boldsymbol{y}|} \min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{1}$$

where $r_t(\theta)$ is the policy ratio defined as $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, and ϵ is a small hyperparameter that limits excessively large updates. PPO uses GAE [Schulman et al., 2015] to estimate the token-level advantages $\hat{A}_t$, which requires a critic model providing token-level value predictions $\hat{V}_t$.

Group Relative Policy Optimization. Compared to PPO, group relative policy optimization (GRPO) eliminates the need for the value function and instead estimates advantage in a group-relative manner. Given an input prompt x, the policy $\pi_{\theta_{\text{old}}}$ samples a group of G responses $\{\boldsymbol{y}^{(i)}\}_{i=1}^{G}$. The estimated advantage for the i-th response is calculated by normalizing these group-level final rewards as $\hat{A}_i = \frac{\mathcal{R}(\boldsymbol{x},\boldsymbol{y}^{(i)}) - \texttt{mean}(\{\mathcal{R}(\boldsymbol{x},\boldsymbol{y}^{(j)})\}_{j=1}^{G})}{\texttt{std}(\{\mathcal{R}(\boldsymbol{x},\boldsymbol{y}^{(j)})\}_{j=1}^{G})}$. This trajectory-level advantage is assigned to all tokens within the corresponding trajectory as $\hat{A}_{i,t} = \hat{A}_i, \forall t \in \{1, \dots, |\boldsymbol{y}^{(i)}|\}$. Similar to PPO, GRPO adopts a clipped objective together with a KL penalty term in the following form:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},\, \{\boldsymbol{y}^{(i)}\}_{i=1}^{G}\sim\pi_\theta(\cdot|x)}$$
$$\left\{ \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\boldsymbol{y}^{(i)}|} \sum_{t=1}^{|\boldsymbol{y}^{(i)}|} \left[\min\left(r_{i,t}(\theta)\hat{A}_{i,t},\ \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{i,t} \right) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right] \right\}, \tag{2}$$

where the policy ratio $r_{i,t}(\theta)$ is defined as $\frac{\pi_\theta(a_{i,t}|s_{i,t})}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_{i,t})}$.

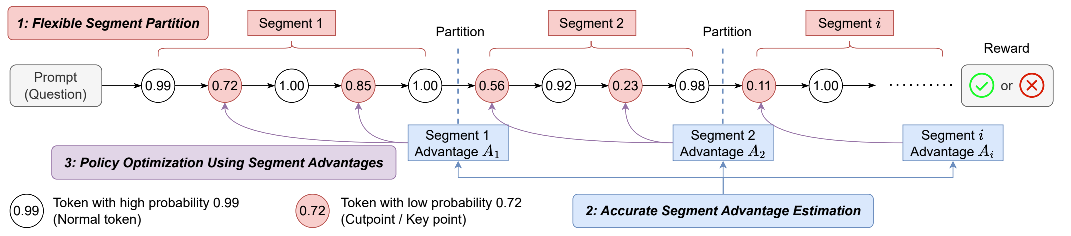

1: Flexible Segment Partition
Segment 1
Partition
Segment 2
Partition
Segment i
Reward
Prompt (Question)
0.99
0.72
1.00
0.85
1.00
0.56
0.92
0.23
0.98
0.11
1.00
or
Segment 1 Advantage A_1
Segment 2 Advantage A_2
Segment i Advantage A_i
3: Policy Optimization Using Segment Advantages
Token with high probability 0.99 (Normal token)
Token with low probability 0.72 (Cutpoint / Key point)
2: Accurate Segment Advantage Estimation

Figure 2: Overview of SPO framework. Our framework consists of three components: segment partition, segment advantage estimation, and policy optimization, each of which can be implemented in different ways. This figure illustrates the cutpoint-based partition strategy used in SPO-chain, where partitioning occurs after a predetermined number of cutpoints. It also illustrates our probability-mask policy optimization method, which assigns the corresponding segment advantages specifically to the cutpoints instead of all tokens within a segment.

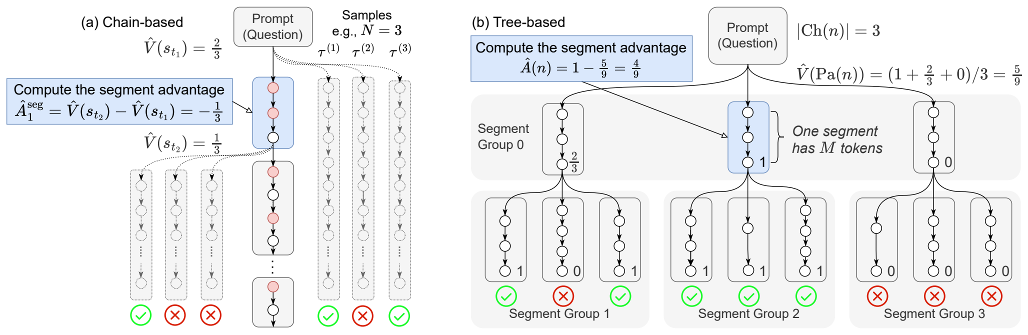

(a) Chain-based
Prompt (Question)
Samples e.g., $N=3$
$\tau^{(1)}$ $\tau^{(2)}$ $\tau^{(3)}$
$\hat{V}(s_{t_1}) = \frac{2}{3}$
Compute the segment advantage
$\hat{A}_1^{\text{seg}} = \hat{V}(s_{t_2}) - \hat{V}(s_{t_1}) = -\frac{1}{3}$
$\hat{V}(s_{t_2}) = \frac{1}{3}$
(b) Tree-based
Compute the segment advantage
$\hat{A}(n) = 1 - \frac{5}{9} = \frac{4}{9}$
Prompt (Question)
$|\text{Ch}(n)| = 3$
$\hat{V}(\text{Pa}(n)) = (1 + \frac{2}{3} + 0)/3 = \frac{5}{9}$
Segment Group 0
One segment has M tokens
Segment Group 1
Segment Group 2
Segment Group 3

Figure 3: **(a)** Chain-based advantage estimation method. For each segment, we independently sample N trajectories to estimate its value V. The advantage for segment k is estimated as $\hat{V}(s_{t_{k+1}}) - \hat{V}(s_{t_k})$. **(b)** Tree-based advantage estimation method. Trajectories are organized in a tree structure, where nodes sharing the same parent form a group with identical prompts and token counts (except for leaf nodes, whose token lengths may vary). This hierarchical organization facilitates the calculation of advantages within each group.

4 Segment Policy Optimization

In this section, we introduce our segment policy optimization (SPO) framework. Effective credit assignment is crucial for training LLMs in reasoning tasks. Trajectory-level methods such as GRPO rely solely on sparse final rewards, making credit assignment challenging. Token-level methods like PPO heavily depend on the critic model, whose value estimation is often inaccurate. The SPO framework aims to balance these extremes by operating at the segment granularity, enabling richer feedback signals compared to GRPO and allowing more accurate Monte Carlo estimates of segment-level advantages, thereby bypassing the need for a critic model.

We develop the SPO framework guided by the following three challenging problems: **(1)** *How to partition the generated sequence into multiple segments?* **(2)** *How to accurately and efficiently estimate the advantage for each segments?* **(3)** *How to update the policy by using the segment-level advantage?* The proposed SPO framework, as illustrated in Figure 2, adopts a modular architecture, where each of its components can be implemented using various strategies, allowing the framework to be tailored to diverse tasks and scenarios. In what follows, we introduce three key components of SPO to answer the above questions and further instantiate SPO with two instances, tailored for short and long CoT scenarios.

4.1 Flexible Segment Partition

A segment is defined as a contiguous sequence of generated tokens, denoted as $\text{seg}_k = [y_{t_k}, y_{t_k+1}, \ldots, y_{t_{k+1}-1}]$, where t_k is the starting token index of the k-th segment. Formally, given a full generated trajectory $\boldsymbol{y} = [y_1, y_2, \ldots, y_T]$, the partition can be expressed as

$$\boldsymbol{y} = [y_1, y_2, \ldots, y_T] = [\text{seg}_1, \text{seg}_2, \cdots, \text{seg}_K].$$

The SPO framework supports arbitrary partition strategies, allowing flexible definition of segment boundaries, without requiring semantic completeness. This flexibility enables us to choose partition granularity between token-level and trajectory-level, allowing a trade-off between computational cost and credit assignment. In this work, we consider two main partition strategies, designed for different Chain-of-Thought (CoT) scenarios:

- **Fixed Token Count Partition.** A straightforward strategy that divides the sequence into segments of a predetermined fixed number of tokens.
- **Adaptive Cutpoint-based Partition.** An advanced strategy (see Section 5.1) that defines segments by accumulating a fixed number of low-probability tokens (i.e., tokens whose probabilities $\pi_\theta(y_t|s_t)$ are less than a threshold ρ). This strategy adaptively places segment boundaries at positions where V values are more likely to change, avoiding the issue in the fixed token count partition strategy where values may remain unchanged across segment boundaries when each segment is short.

4.2 Segment Advantage Estimation via Monte Carlo

After obtaining each segment $\text{seg}_k = [y_{t_k}, y_{t_k+1}, \ldots, y_{t_{k+1}-1}]$, we define the segment advantage in the following way:

$$A_k^{\text{seg}} := V(s_{t_{k+1}}) - V(s_{t_k}) = V([\boldsymbol{x}, \text{seg}_1, \ldots, \text{seg}_k]) - V([\boldsymbol{x}, \text{seg}_1, \ldots, \text{seg}_{k-1}]), \tag{3}$$

which measures the incremental improvement in expected reward resulting from generating the tokens in seg_k. The core challenge lies in accurately estimating the value function $V(s)$. A common approach is to train a critic model. However, in the LLM scenario, accurately predicting token-level state values presents particular challenges. Each prompt can be viewed as a distinct "environment" with its own unique distribution of trajectories. Due to significant variations in trajectories generated by different prompts, a critic model trained on one set of prompts may struggle to generalize effectively to previously unseen ones. Moreover, the limited number of trajectory samples per prompt (e.g. 8 trajectories per prompt) further restricts the critic's access to sufficient prompt-specific data, making accurate state-value estimation even more difficult. Indeed, as empirically demonstrated by Kazemnejad et al. [2024], the critic model struggles to accurately estimate values in the LLM scenario. Given these considerations, we propose to bypass the critic entirely and instead adopt Monte Carlo (MC) estimation to compute unbiased estimates of segment values directly from sampled trajectories. While high variance is often a concern with MC estimation, in the LLM setting, rewards are sparse and binary, significantly mitigating the variance issue. Specifically, we can view V as a Bernoulli variable, the variance is at most 0.25. We find that the accuracy obtained with a small number of samples ($N = 4$ or $N = 9$) is sufficient for effective policy optimization. Our SPO framework supports various MC sampling strategies:

- **Chain-based Advantage Estimation.** We can independently roll out N trajectories from state s, and estimate $V(s)$ as the average reward of these trajectories, further yielding the estimation of the advantage A_k^{seg} following Equation (3).
- **Tree-based Advantage Estimation.** An advanced strategy of advantage estimation for improved sample efficiency (see details in Section 6.2). Unlike the chain-based advantage estimation, which discards MC samples after estimating the V values, this strategy organizes MC samples into a tree structure and computes the values via bottom-up aggregation of rewards along the tree. Nodes sharing the same parent have identical prompts and token counts, forming segment groups, enabling advantage computations within each group. This tree structure allows the reuse of MC samples for policy optimization, significantly enhancing sample efficiency in a long CoT scenario.

4.3 Policy Optimization Using Segment Advantages

Given a set of generated segments $\{\text{seg}_k\}$ and their corresponding advantages $\{A_k^{\text{seg}}\}$, we now focus on effectively leveraging training samples to update the policy π_θ. Specifically, we can adapt various policy optimization approaches into our SPO framework:

- **Policy Gradient.** We can assign A_k^{seg} to all tokens contained within the k-th segment seg_k and then optimize using the PPO loss.
- **Policy Gradient with Probability Masks.** An improved version of policy gradient (see details in Section 5.3), incorporating probability masks. Instead of uniformly assigning A_k^{seg} to all tokens within the segment, this strategy exclusively assigns A_k^{seg} only to low-probability tokens, based on the intuition that these tokens primarily contribute to the segment's advantage. This refined approach further enhances the credit assignment to critical tokens.
- **Other Strategies.** Our framework can also be adapted to other policy optimization methods such as policy iteration-based approach (see discussions in Appendix A) and potentially GFlowNet-based optimization [Bartoldson et al., 2025].

5 SPO-chain for Short CoT

For short Chain-of-Thought (CoT) scenarios, where the computational overhead of MC sampling is low and segments typically involve a small number of tokens, we designed a tailored instance of SPO by taking into account these characteristics, which we refer to as **SPO-chain**. The core features of SPO-chain lie in its cutpoint-based segment partition, chain-based segment advantage estimation, and policy optimization via policy gradient with probability masks.

5.1 Adaptive Cutpoint-based Partition

The fixed token count partition strategy faces a critical issue in short CoT scenarios that the number of tokens in each segment is not very large. If the token probabilities $\pi_\theta(y_t|s_t)$ within a segment are high (close to 1), the V values estimated at the beginning and end of the segment will not differ significantly. This can lead to unnecessary partitioning, ultimately resulting in a waste of sampling budget. To address this, we propose an adaptive cutpoint-based partition strategy. Specifically, we first identify the *cutpoints*, defined as positions where token probabilities drop below a pre-defined threshold ρ, so that we have the set of all cutpoints defined as

$$\mathcal{U}_\theta = \{t < T \mid \pi_\theta(y_t|s_t) < \rho\}.$$

These cutpoints represent positions where the model's reasoning trajectory could diverge, thus potentially inducing changes in values. For better credit assignment, we prefer each segment to contain fewer cutpoints. Then, given a fixed segment count K, we find a partition that reflects this principle by solving the following optimization problem:

$$\min_{\{t_k\}_{k=1}^K} \sum_{k=1}^{K} |\mathcal{U}_\theta \cap [t_k, t_{k+1})|^2.$$

It can be shown that the optimal solution evenly distributes cutpoints across segments (assuming $|\mathcal{U}_\theta|$ is divisible by K for simplicity), that is,

$$|\mathcal{U}_\theta \cap [t_k, t_{k+1})| = \frac{|\mathcal{U}_\theta|}{K}, \qquad \forall k \le K,$$

which corresponds to partitioning the trajectory such that each segment contains the same number of cutpoints, as shown in Figure 2. A practical example is provided in Appendix F. Our experiments further show that this partition strategy would lead to a superior performance (see our comparison of different segment partition strategies in Section 7.1).

5.2 Chain-based Segment Advantage Estimation

In the short CoT scenario, the computational overhead of MC estimation is generally manageable. Thus, we adopt a simple, chain-based MC sampling approach as shown in Figure 3(a). Specifically, at each segment boundary state $s_{t_k} = [\boldsymbol{x}, \boldsymbol{y}_{<t_k}]$, we independently sample N trajectories from the policy π_θ, i.e. $\boldsymbol{\tau}_{t_k}^{(j)} \sim \pi_\theta(\cdot|s_{t_k}), j = 1, \dots, N$ and then estimate the value of such a state by averaging the returns from these sampled trajectories, i.e.,

$$\hat{V}(s_{t_k}) = \frac{1}{N} \sum_{j=1}^{N} \mathcal{R}(\boldsymbol{x}, [\boldsymbol{y}_{<t_k}, \boldsymbol{\tau}_{t_k}^{(j)}]).$$

The estimated advantage $\hat{A}_k^{\text{seg}}$ of each segment k is computed by taking the difference between the state values at consecutive segment boundaries $\hat{A}_k^{\text{seg}} = \hat{V}(s_{t_{k+1}}) - \hat{V}(s_{t_k})$ following Equation (3).

5.3 Policy Gradient with Probability Masks

Policy gradient methods have become mainstream for training LLMs using RL. Accordingly, we adopt a policy gradient-based approach to optimize the policy. For example, once we have computed the segment advantage A_k^{seg}, we can assign it to all tokens within the segment and obtain token-level advantage A_t, then adopt the PPO loss in Equation (1) to optimize the policy. However, since the changes in V values between segments are primarily caused by tokens at cutpoints, we assign the segment advantage A_k^{seg} only to these critical tokens. Formally, our policy is trained by minimizing the following loss:

$$\mathcal{J}_{\text{SPO}}^{\text{chain}}(\theta) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},\, [\text{seg}_1,\ldots,\text{seg}_K]\sim\pi_{\theta_{old}}} \left\{ \frac{1}{Z}\sum_{k=1}^{K}\sum_{t=t_k}^{t_{k+1}-1} \left[M_t \cdot \min\Big(r_t(\theta)\hat{A}_k^{\text{seg}},\, \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_k^{\text{seg}}\Big) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right] \right\}, \tag{4}$$

where M_t is the mask whose value is 1 if $\pi_{\theta_{old}}(a_t|s_t) < \rho$ and 0 otherwise, and Z is a normalization term that equals the total number of tokens for when the mask M_t is 1: $Z = \sum_{t=1}^{T} M_t$. This approach further enhances credit assignment within each segment by concentrating the advantage signals on fewer critical tokens. Our experiments demonstrate that this method improves the accuracy (refer to our ablation study on the probability-mask optimization strategy in Section 7.1).

6 SPO-tree for Long CoT

For long Chain-of-Thought (CoT) scenarios, where the sampling cost of chain-based MC estimation in SPO-chain becomes prohibitively high, we propose a novel tree-based segment advantage estimation method. This approach drastically reduces sampling overhead, making it feasible to apply our framework effectively in long CoT settings. We refer to this instantiation of our SPO framework as **SPO-tree**. In particular, SPO-tree features the fixed token count partition, tree-based segment advantage estimation, and policy gradient with probability masks.

6.1 Fixed Token Count Partition

In long CoT scenarios, each segment typically contains a large number of tokens. As a result, it is unlikely that all token transitions within an entire segment will have probabilities close to 1. Thus, we employ a partition strategy where segment boundaries are made at fixed token intervals, which allows us to generate segments with equal token count, supporting our tree-based segment advantage estimation method proposed in the following subsection.

6.2 Tree-based Segment Advantage Estimation

The core drawback of the chain-based segment advantage estimation strategy studied in Section 5.2 is that it discards samples used for estimating the values (e.g., $\boldsymbol{\tau}^{(1)}$, $\boldsymbol{\tau}^{(2)}$, $\boldsymbol{\tau}^{(3)}$ in Figure 3(a)), leading to substantial waste of samples in scenarios involving long reasoning trajectories. To address this issue, we propose a tree-based segment advantage estimation strategy, which reuses the samples employed for value estimation in policy optimization, significantly improving the sample efficiency.

As illustrated in Figure 3(b), we model the trajectory sampling process as a tree structure. We define $\text{hist}(n)$ as the full history sequence corresponding to node n, including the initial prompt $\boldsymbol{x}$ and all tokens generated along the path from the root node to node n. Each node n represents a segment $\text{seg}(n)$, which is generated by extending the sequence of its parent node, $\text{Pa}(n)$, with M newly sampled tokens, i.e.,

$$\text{seg}(n) = \left[y_1^{(n)}, \ldots, y_M^{(n)}\right], \quad y_t^{(n)} \sim \pi\left(\cdot \,\middle|\, \left[\text{hist}(\text{Pa}(n)), \boldsymbol{y}_{<t}^{(n)}\right]\right).$$

Each node n generates a set of child nodes, denoted as $\text{Ch}(n)$, and has a set of siblings, denoted as $\text{Sib}(n)$. Nodes among siblings share the same prompts and sequence lengths (except for leaf nodes), ensuring fair comparison under the same token budget. The value of the state represented by node n is denoted as $V(n)$, which can be estimated recursively from leaves to root as follows:

$$\hat{V}(n) = \begin{cases} \mathcal{R}(\boldsymbol{x}, \text{hist}(n)), & \text{if } n \text{ is a leaf node} \\ \frac{1}{|\text{Ch}(n)|} \sum_{n' \in \text{Ch}(n)} \hat{V}(n'), & \text{otherwise} \end{cases}.$$

We denote the advantage of the segment represented by node n as $A(n)$. The estimated advantage $\hat{A}(n)$ for node n, relative to its siblings, is computed either in unnormalized or normalized form as:

$$\hat{A}(n) = \hat{V}(n) - \texttt{mean}_{n' \in \text{Sib}(n)} \hat{V}(n') \quad \text{or} \quad \frac{\hat{V}(n) - \texttt{mean}_{n' \in \text{Sib}(n)} \hat{V}(n')}{\texttt{std}_{n' \in \text{Sib}(n)} \hat{V}(n')}.$$

More details about the tree construction are presented in Appendix H. In contrast to the chain-based segment advantage estimation strategy, each node (segment) in the tree can be used as a training example, substantially improving sample efficiency. In addition, due to extensive node-sharing among trajectories, the actual number of tokens we need to sample is significantly less than the sum of tokens across all trajectories, greatly reducing sampling overhead. Within each training iteration, sampling a large number of trajectories from a single question can lead to the model overfitting to specific samples. To address this, we introduce a replay buffer mechanism that distributes sampled trajectories across multiple iterations in Appendix C.

The width of the tree determines the number of nodes among siblings. A larger tree width provides more sibling nodes for comparison, resulting in more nuanced advantage estimations. The depth of the tree determines the granularity of segment partition: deeper trees lead to more segments, offering finer-grained reward signals. Additionally, a deeper and wider tree structure allows for more accurate value estimates for each node, as the estimation benefits from a greater number of sampled trajectories. Furthermore, this tree structure aligns with the intuition that the initial parts of a reasoning trajectory are more uncertain and therefore require more samples to accurately estimate V values, whereas the later parts of the trajectory are more certain and require fewer samples for estimation. Correspondingly, in the tree structure, nodes closer to the root (upper levels) aggregate a large number of trajectories for estimating their values, while lower-level nodes (closer to the leaves) rely on fewer samples. This hierarchical allocation of samples naturally matches the varying levels of uncertainty throughout the reasoning process.

6.3 Policy Gradient with Probability Masks

To focus model training on segments with discriminative signals, we extract segments with non-zero advantages from the tree for training. We also note that such filtering of non-zero advantage samples aligns with practices in several concurrent works [Yu et al., 2025, Lin et al., 2025]. Thus, we propose to minimize $\mathcal{J}_{\text{SPO}}^{\text{tree}}$ below for policy optimization:

$$\mathcal{J}_{\text{SPO}}^{\text{tree}}(\theta) = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D},\, \text{tree} \sim \pi_{\theta_{\text{old}}}} \Bigg\{ \frac{1}{Z} \sum_{n \in \text{tree}: \hat{A}(n) \neq 0} \sum_{t=1}^{|\text{seg}(n)|} \Big[M_{n,t} \cdot \min \Big(r_{n,t}(\theta) \hat{A}(n),\, \text{clip}(r_{n,t}(\theta), 1-\epsilon, 1+\epsilon) \hat{A}(n) \Big) - \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \Big] \Bigg\}, \tag{5}$$

where $M_{n,t}$ is the mask whose value is 1 if $\pi_\theta(y_t^{(n)} \mid [\text{hist}(\text{Pa}(n)), \boldsymbol{y}_{<t}^{(n)}]) < \rho$ and 0 otherwise, the ratio term $r_{n,t}(\theta)$ is then defined as $\frac{\pi_\theta(y_t^{(n)} | [\text{hist}(\text{Pa}(n)), \boldsymbol{y}_{<t}^{(n)}])}{\pi_{\theta_{\text{old}}}(y_t^{(n)} | [\text{hist}(\text{Pa}(n)), \boldsymbol{y}_{<t}^{(n)}])}$, and the normalization term Z is given by $Z = \sum_{n \in \text{Tree}:\, \hat{A}(n) \neq 0} \sum_{t=1}^{|\text{seg}(n)|} M_{n,t}$.

7 Experiments

7.1 Experiments on SPO-chain

Experimental Setups. We evaluate SPO-chain with the RhoMath 1.1B model [Lin et al., 2024] on the GSM8K dataset [Cobbe et al., 2021] using Pass@1 (accuracy) as our primary evaluation metric.

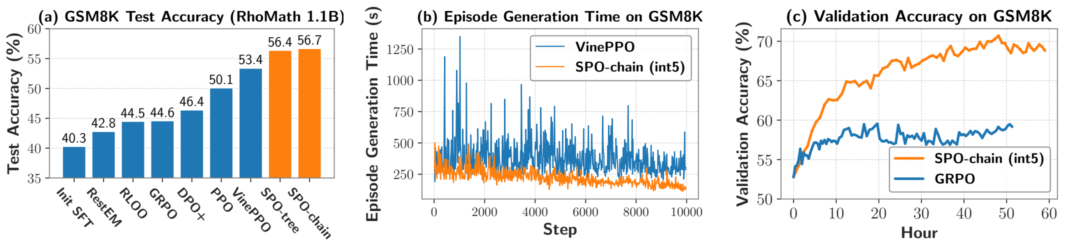


Figure 4: **(a)** Test accuracy comparison of different methods on GSM8K. Baseline results are from Kazemnejad et al., 2024. **(b)** Episode generation time comparison between SPO-chain (int5) and VinePPO during training. **(c)** Validation accuracy of SPO-chain (int5) and GRPO during training.

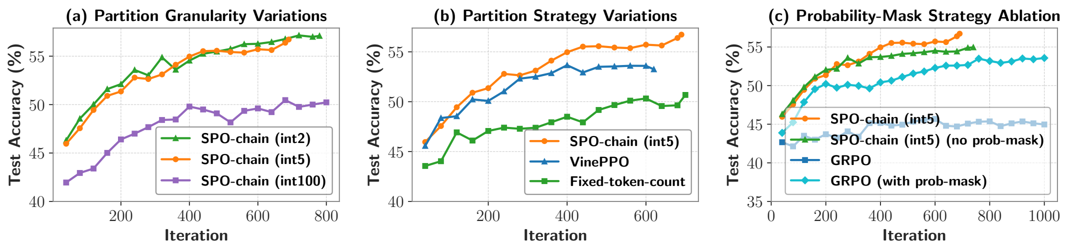


Figure 5: **(a)** Variations of segment partition granularity (different cutpoint intervals). **(b)** Variations of segment partition strategies. **(c)** Ablation on probability-mask policy optimization strategy.

Following Kazemnejad et al. [2024], we first finetune the base model on the GSM8K training set, and then use the obtained SFT model for subsequent RL training. We compare against baseline methods including RestEM, DPO, PPO, GRPO, RLOO, and VinePPO. The experimental hyperparameters largely follow Kazemnejad et al. [2024], with detailed settings provided in Appendix E. For SPO-chain, we set the number of MC samples to $N = 9$ and partitioned the model output at intervals of every 5 cutpoints, which is referred to as SPO-chain (int5). We also evaluate our SPO-tree method in short CoT scenario. We adopt a 6-6-6 tree structure, meaning the tree consists of three layers and each internal node expands into six child nodes, which is referred to as SPO-tree (6-6-6). For the first two layers, we segment the model output every 30 tokens, while for the final layer, we let the model complete the entire reasoning trajectory.

Comparison with Baseline Methods. As shown in Figure 4(a), our method, SPO-chain (int5), achieves the highest accuracy on the GSM8K test set, outperforming PPO and GRPO by 6-12 percentage points. Furthermore, compared to VinePPO, our method not only delivers higher accuracy but also requires less time for episode generation, as demonstrated in Figure 4(b). This is due to our SPO-chain (int5) method uses fewer advantage estimation points per trajectory compared to VinePPO. Additionally, our SPO-tree (6-6-6) method achieves the second-highest accuracy on the GSM8K test set, validating its effectiveness and making it an efficient alternative. We also compared our method with GRPO in terms of validation performance under the same wall-clock time[2] (Figure 4(c)). The results show that our algorithm significantly outperforms GRPO and ultimately converges to a much better solution. We further conduct experiments with the DeepSeekMath 7B model [Shao et al., 2024], with detailed results provided in Appendix D.

Impact of Segmentation Granularity. To investigate how the segment granularity affects final model performance, we conducted experiments using various segment intervals, as shown in Figure 5(a). The results indicate that an interval of 2 performs slightly better than an interval of 5, while an interval of 100 performs significantly worse than 5. This suggests that using an excessively fine-grained interval provides limited benefits, whereas an overly coarse interval can severely degrade accuracy. These findings support the design of our segment-level advantage approach. Specifically, the token-level advantage offers only marginal improvement over the segment-level advantage, suggesting that such fine granularity may not be necessary in practice. In contrast, the trajectory-level advantage performs considerably worse, highlighting the drawbacks of overly coarse granularity. Overall, our

[2] Evaluation time is also included in the reported wall-clock time. The model is evaluated every 10 iterations.

segment-level advantage method achieves an effective and efficient balance, achieving significantly better accuracy than the trajectory-level approach while avoiding the high computational complexity of token-level advantage estimation.

Comparison of Different Segment Partition Strategies. We compare different trajectory partition methods, including the naive fixed token count partition, the heuristic rules (e.g. line breaks) for partitioning as in VinePPO, and the cutpoint-based segment partition in SPO-chain. The results are presented in Figure 5(b). For the naive fixed token count partitioning method, we explicitly set each trajectory's segment count at 3, making it higher than the total sampling budget of the SPO-chain (int5). Remarkably, despite having the smallest sampling budget, SPO-chain (int5) achieves the best test accuracy, validating the effectiveness of our cutpoint-based segment partition strategy.

Ablation Study on Probability-Mask Optimization Strategy. We conduct an ablation study on our proposed probability-mask optimization technique. As shown in Figure 5(c), removing the probability-mask technique leads to a decreased accuracy for SPO-chain (int5), from 56.7% to 55.0%. Surprisingly, applying the probability-mask technique to GRPO results in a significant improvement, boosting its accuracy from 45.7%[3] to 53.6%. We hypothesize that employing the probability-mask technique enables more precise credit assignment to tokens that are most likely to influence the model's reasoning trajectory. Meanwhile, the losses associated with tokens whose probabilities are close to 1 are masked out and thus excluded from the overall loss function, which potentially helps reduce overfitting.

7.2 Experiments on SPO-tree

Experimental Setups. To evaluate our method in the long CoT scenario, we utilize DeepSeek-R1-Distill-Qwen-1.5B model as our base model, which has been fine-tuned on huge amount of long CoT data, demonstrating complex reasoning strategies and typically generating long outputs. Given the strong capability of these models, we select the competition-level MATH [Hendrycks et al., 2021] dataset for our training and evaluation. We mainly focus on comparing our proposed SPO-tree method against GRPO, as GRPO is the mainstream training approach adopted for reasoning models, particularly favored for long CoT scenarios due to its efficiency. We do not compare with VinePPO because its efficiency issue, making it impractical to apply in long CoT settings. For our SPO-tree method, the structure remains identical to the SPO-tree used in the short CoT scenario, as described in Section 7.1. Detailed hyper-parameters can be found in Appendix E.

Comparison with Baseline Methods. We initially limit the training context window size to 2K and evaluate model accuracy (Pass@1) on MATH500 every 10 iterations. For evaluation, we follow the implementation provided by Kazemnejad et al. [2024], setting the evaluation context window also to 2K and using greedy decoding (temperature = 0). Figure 6(a) shows the training curves of SPO-tree and GRPO, demonstrating that our method consistently achieves higher accuracy throughout the training process under the same wall-clock time. This validates the effectiveness of SPO-tree and highlights the benefit of leveraging finer-grained signals.

To further assess our method, we continue training from the 2K checkpoint and expand the model's context length to 4K. The corresponding model performance is presented in Table 1. We adopt the evaluation script provided by Face [2025] to conduct evaluation on the obtained model checkpoints. As shown in the results, the SPO-tree consistently outperforms GRPO across all context sizes and achieves superior performance under 2K and 4K contexts, which are the context lengths it was trained on. We hypothesize that our segment-level advantage method significantly increases token efficiency due to more precise credit assignment, enabling the model to arrive at correct answers more straightforwardly (see Appendix G). This improvement leads to considerably better performance under limited context sizes. Interestingly, even though our model is trained under context windows up to 4K, it still achieves improved performance under 32K context evaluation, surpassing the base model. We also compared our approach against the most recent works, including DeepScaleR [Luo et al., 2025] and STILL-3 [Chen et al., 2025], both of which were trained on DeepSeek-R1-Distill-Qwen-1.5B using GRPO. Notably, DeepScaleR and STILL-3 adopted larger training datasets and extended context windows for training. Specifically, DeepScaleR progressively increased the context length from 8K to 16K and finally to 24K, whereas our training scales only from 2K to 4K. While

[3]Our GRPO implementation achieves a slightly higher test accuracy (45.7%) compared to the 44.6% reported in Kazemnejad et al. [2024].

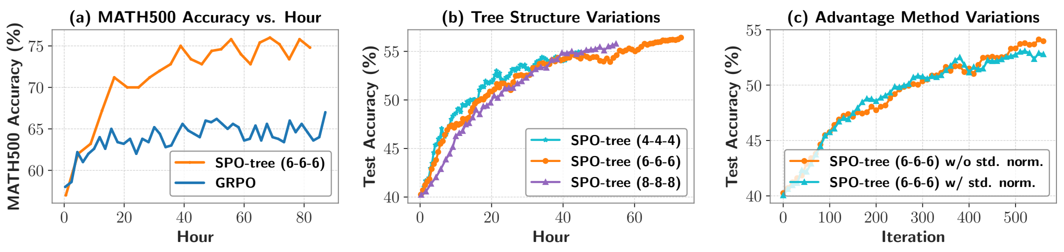


Figure 6: **(a)** Comparison of SPO-tree (6-6-6) and GRPO on MATH500 with a context size of 2K. **(b)** Variations of tree structures on GSM8K. **(c)** SPO-tree with different advantage methods on GSM8K.

Dataset	Eval Context Size	Base	GRPO	SPO-tree	DeepScaleR*	STILL-3*
MATH500	2K	0.566	0.62	**0.736**	0.538	0.662
	4K	0.74	0.752	**0.828**	0.744	0.794
	32K	0.838	0.84	0.848	**0.878**	0.846
AIME24	2K	0.067	0.033	**0.1**	0	0.067
	4K	0.167	**0.2**	**0.2**	0.167	0.133
	32K	0.267	**0.333**	**0.333**	**0.333**	0.233

* These models were trained using larger datasets and longer context windows. Specifically, DeepScaleR uses 8K→16K→24K context and STILL-3 sets 'generate_max_len' to 29000 during training, while our GRPO and SPO-tree use a maximum context of 4K during training.

Table 1: Accuracy comparison among methods with various context sizes on MATH500 and AIME24.

DeepScaleR performs best at 32K, we observe the opposite trend in lower-context settings, where DeepScaleR exhibits the worst performance at 2K and 4K. This finding suggests that GRPO-based training might not optimize token efficiency, potentially leading to redundancy in the generated outputs, and thus negatively affecting the model's performance when evaluated under limited context size. We will extend our experiments to longer context scenarios in our later studies.

Comparison of Different Tree Structures and Advantage Computation Methods. To investigate the impact of different tree structures, we compared the performance[4] of various tree structures (4-4-4, 6-6-6, 8-8-8) on the GSM8K test set (Figure 6(b)). When using a larger tree structure, each iteration generates a greater number of segments with non-zero advantage. Therefore, we set different values of the hyperparameter "num_episodes_per_iteration" for different tree structures: specifically, we set it to 1024 for SPO-tree (8-8-8), 512 for SPO-tree (6-6-6), and 384 for SPO-tree (4-4-4). We found that the performance differences among these tree structures were not substantial under the same wall-clock time, indicating the robustness of the tree structure. Smaller tree structures achieve higher accuracy initially, possibly because they can process more data examples within the same time frame. However, larger tree structures eventually outperform smaller ones at later training stages, as they enable more accurate value estimation and benefit from having more segments within each group, leading to more reliable and nuanced advantage estimates. We also compared the performance of different advantage calculation methods with and without standard deviation normalization. As shown in Figure 6(c)), both methods perform similarly, while the variant without normalization performs slightly better.

8 Conclusion and Future Work

In this work, we propose Segment Policy Optimization (SPO), a novel RL framework that leverages segment-level advantage estimation at an intermediate granularity, achieving a better balance by offering more precise credit assignment than trajectory-level methods and requiring fewer estimation points than token-level methods, enabling accurate advantage estimation based on MC without a critic model. We further instantiate SPO for short CoT and long CoT scenarios. Extensive experiments demonstrate the effectiveness and efficiency of our proposed SPO framework in both scenarios.

[4]Evaluation time is also included in the reported wall-clock time. The model is evaluated every 10 iterations.

Due to limited computational resources, our current experiments in the long CoT scenario are limited to a maximum context size of 4K tokens. In the future, we plan to conduct additional experiments with larger context sizes. Additionally, our current experiments focus primarily on mathematical tasks. We will test the effectiveness of SPO in broader application scenarios such as code generation and RLHF.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Brian R Bartoldson, Siddarth Venkatraman, James Diffenderfer, Moksh Jain, Tal Ben-Nun, Seanie Lee, Minsu Kim, Johan Obando-Ceron, Yoshua Bengio, and Bhavya Kailkhura. Trajectory balance with asynchrony: Decoupling exploration and learning for fast, scalable llm post-training. *arXiv preprint arXiv:2503.18929*, 2025.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*, 2024.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, et al. An empirical study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL https://github.com/huggingface/open-r1.

Thomas Foster and Jakob Foerster. Learning to reason at the frontier of learnability. *arXiv preprint arXiv:2502.12272*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Unlocking rl potential for llm reasoning through refined credit assignment. *arXiv preprint arXiv:2410.01679*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Zhenghao Lin, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, Yujiu Yang, Jian Jiao, Nan Duan, et al. Rho-1: Not all tokens are what you need. *arXiv preprint arXiv:2404.07965*, 2024.

Zhihang Lin, Mingbao Lin, Yuan Xie, and Rongrong Ji. Cppo: Accelerating the training of group relative policy optimization-based reasoning models. *arXiv preprint arXiv:2503.22342*, 2025.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. `https://github.com/PraMamba/DeepScaleR`, 2025.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*, 2023.

Michael Noukhovitch, Shengyi Huang, Sophie Xhonneux, Arian Hosseini, Rishabh Agarwal, and Aaron Courville. Asynchronous rlhf: Faster and more efficient off-policy rl for language models. *arXiv preprint arXiv:2410.18252*, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Yuxiao Qu, Matthew YR Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement fine-tuning. *arXiv preprint arXiv:2503.07572*, 2025.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. From r to q^*: Your language model is secretly a q-function. *arXiv preprint arXiv:2404.12358*, 2024.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, et al. Advancing llm reasoning generalists with preference trees. *arXiv preprint arXiv:2404.02078*, 2024.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024.

A Policy Learning via Policy Iteration

We can alternatively adopt a policy iteration-based approach formulated in the following way for policy learning. The objective of training LLMs via RL can be written as a KL-constrained policy optimization problem:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{h=1}^{H} \left(r(s_h, a_h) - \beta \log \frac{\pi(a_h|s_h)}{\pi_{\text{ref}}(a_h|s_h)} \right) \right].$$

By defining the modified reward $\bar{r}(s,a) = r(s,a) + \beta \log \pi_{\text{ref}}(a|s)$, we arrive at an equivalent maximum-entropy RL formulation:

$$J(\pi) = \mathbb{E}_{\pi, s_0 \sim d_0} \left[\sum_{h=1}^{H} \left(\bar{r}(s_h, a_h) + \beta \mathcal{H}(\pi(\cdot|s_h)) \right) \right].$$

Under this formulation, the Q-function and value function satisfy the soft Bellman equation:

$$Q(s_h, a_h) = \bar{r}(s_h, a_h) + V(s_{h+1}).$$

Given the above relationships, we can optimize the Q-function via the loss:

$$L_Q(\theta) = \mathbb{E}_{(s_h, a_h, s_{h+1}) \sim \mathcal{D}} \left[\left(Q_\theta(s_h, a_h) - \bar{r}(s_h, a_h) - V_\phi(s_{h+1}) \right)^2 \right].$$

Given the estimates of Q-function and value function, the optimal policy satisfies:

$$Q_\theta(s_h, a_h) = V_\phi(s_h) + \beta \log \pi_\theta(a_h|s_h).$$

Substituting this optimal policy relationship back into the Bellman equation results in the following policy optimization objective:

$$L_\pi(\theta) = \mathbb{E}_{(s_h, a_h, s_{h+1}) \sim \mathcal{D}} \left[\left(V_\phi(s_h) + \beta \log \pi_\theta(a_h|s_h) - \bar{r}(s_h, a_h) - V_\phi(s_{h+1}) \right)^2 \right].$$

Since we have estimated the advantages $A(s_h, a_h) = V(s_{h+1}) - V(s_h)$, explicit estimation of value function V is not necessary. Thus, the policy optimization objective further simplifies to:

$$L_\pi(\theta) = \mathbb{E}_{(s_h, a_h) \sim \mathcal{D}} \left[\left(\beta \log \frac{\pi_\theta(a_h|s_h)}{\pi_{\text{ref}}(a_h|s_h)} - A(s_h, a_h) \right)^2 \right].$$

Each iteration of this procedure corresponds to a policy improvement step, and after updating the policy, newly sampled trajectories can be employed to update the Q-function estimation.

We conduct preliminary experiments using this objective function, where we adopt our SPO-tree (6-6-6) method and introduce an importance weight $\frac{\pi_\theta}{\pi_{old}}$ to alleviate the off-policy influence. The results, shown in Figure 7, demonstrate that our approach enables steady policy improvement. However, it does not reach the performance achieved by policy gradient with probability masks described in Section 6.3), which achieves a validation accuracy slightly above 0.7.

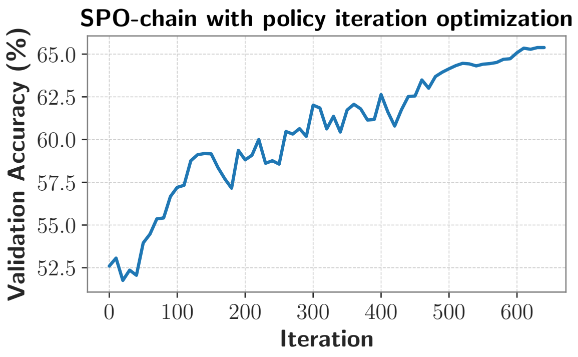


Figure 7: Validation accuracy on GSM8K using policy iteration optimizing method.

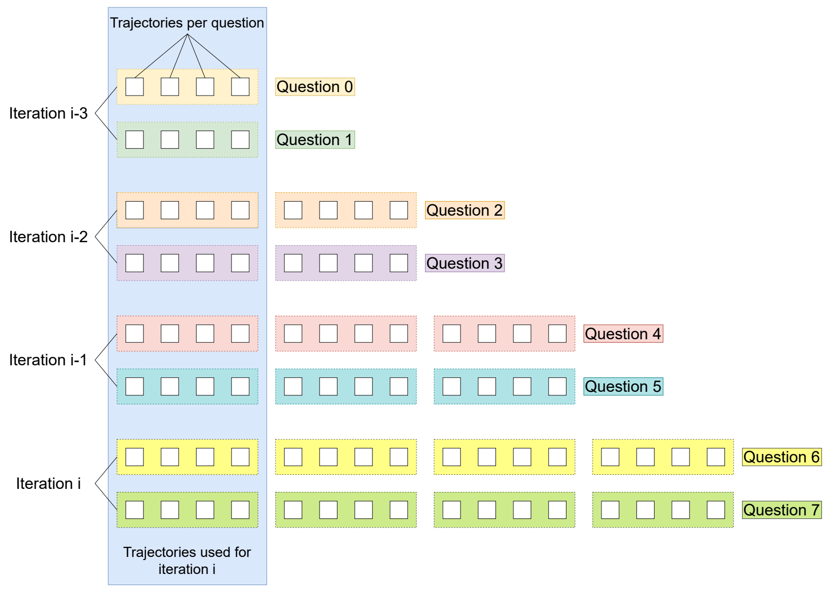


Figure 8: Illustration of the replay buffer. In this example, in each iteration, we sample two questions, with each question generating four trajectories. These trajectories are distributed across four iterations for optimization. The trajectories used in the i-th iteration are selected in the blue box. In total, there are 8 different questions, each with 4 trajectories, ensuring a balanced distribution of samples.

B Incorporate Process Reward

Our framework is compatible with the process reward. Inspired by the work in Setlur et al. [2024], we adopt the current policy's BoN as the prover policy μ and utilize its advantage as the process reward. Specifically, the gradient is computed as follows:

$$\nabla_\pi \ell(\pi) = \sum_{h=1}^{H} \nabla_\pi \log \pi(a_h \mid s_h) \cdot (Q^\pi(s_h, a_h) + \alpha \cdot A^\mu(s_h, a_h))$$

Unlike the aforementioned work, our method does not require training a separate reward model (PAV). Given that the adopted prover policy μ is the current policy's BoN, we can compute its value function $V^\mu(s_h)$ from the value function of the current policy $V^\pi(s_h)$:

$$V^\mu(s_h) = 1 - (1 - V^\pi(s_h))^N,$$

where N is the number of MC samples. Then, the prover policy's advantage $A^\mu(s_h, a_h)$ can be easily computed as follows:

$$A^\mu(s_h, a_h) = V^\mu(s_{h+1}) - V^\mu(s_h).$$

Thus, we can directly integrate process rewards into our framework using MC samples generated from the current policy, simplifying the overall algorithm and effectively circumventing potential issues that arise from introducing a separate reward model, such as fitting errors and reward hacking.

C Replay Buffer

Within each iteration, sampling a large number of trajectories from a single question can lead to imbalanced training, causing the model to overfit to specific samples. To address this issue, we introduce a replay buffer mechanism that distributes sampled trajectories across multiple iterations, ensuring a more balanced question distribution, as shown in Figure 8. Additionally, PPO's clip mechanism helps maintain stable optimization when using these trajectories from previous iterations. Specifically, in our experiments, we use a *6-6-6* tree structure for sampling, generating $6 \times 6 \times 6 = 216$ trajectories per question. In each iteration, we sample 16 distinct questions to generate trajectories, and these trajectories are then distributed across the following 8 iterations, with at most 32 trajectories per question processed in each iteration. As a result, each iteration covers 128 distinct questions (8 × 16), ensuring a well-balanced sample distribution. We observe that the clip ratio remains around 0.1, validating the effectiveness of our replay buffer strategy.

Our replay buffer mechanism leads to a certain degree of off-policy learning, enabling our framework to be compatible with the asynchronous RL setting, which was first explored by Noukhovitch et al. [2024]. Leveraging asynchronous RL can further parallelize data collection and policy optimization, potentially leading to additional efficiency improvements.

D Additional Experiment Results

We also evaluate our SPO-chain method using DeepSeekMath 7B on the MATH dataset, as shown in Figure 9. Our approach achieves performance comparable to VinePPO and surpasses all other baselines. Due to computational constraints, we initially train with an interval of 10 and later switch to 5. After switching from the interval of 10 to 5, the model performance continues to improve, suggesting that using a finer-grained advantage is beneficial for model training. Notably, as shown in Figure 10, our method uses significantly fewer segments than VinePPO while attaining similar performance.

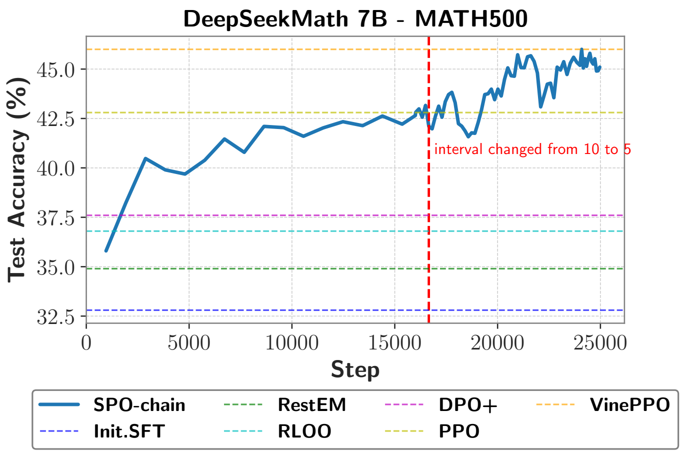


Figure 9: Performance of SPO-chain on DeepSeekMath 7B evaluated on the MATH500. Baseline results are adopted from Kazemnejad et al., 2024.

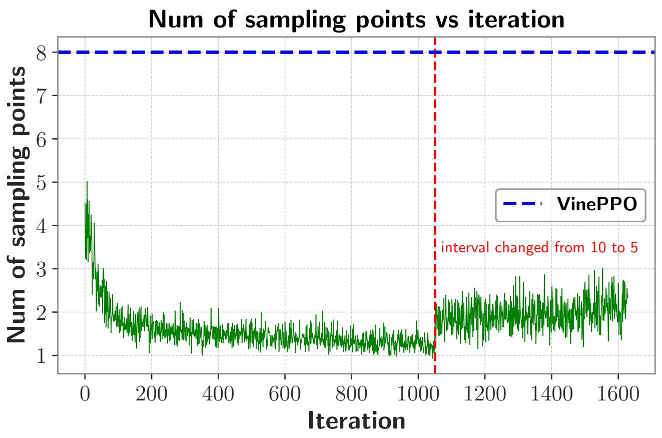


Figure 10: Comparison of the number of sampling points between SPO-chain and VinePPO. According to Foster and Foerster, 2025, there are approximately 8 reasoning steps on average for MATH.

E Hyperparameters and Compute Resources

Our code base is based on Kazemnejad et al. [2024]. Most hyperparameters are the same as theirs. Here we list important hyperparameters in the following five tables. We perform SPO-chain and SPO-tree experiments with RhoMath 1.1B using a single A100 GPU (40GB). For the long-CoT experiments (2K and 4K context) using DeepSeek-R1-Distill-Qwen-1.5B, we use a single A100 GPU (80GB).

SPO-chain (int5) Rho 1.1B on GSM8K

Hyperparameter	Value
Target train batch size	64
Num episodes per iteration	512
Dataset samples per iteration	64
Samples (per prompt)	8
Cutpoint interval	5
K (MC samples)	9
M (Tokens per level)	-
Branch factors	-
Train policy temperature	0.6
Train policy top-p	0.9
Train context size	2047
Initial KL coef	0.0001
Learning rate	1×10^{-6}
Epochs per iteration	2
Prob mask threshold	0.9
Eval. context size	2047
Eval. temperature	0.35
Eval. top-p	0.9
Eval. samples	16

SPO-tree (6-6-6) Rho 1.1B on GSM8K

Hyperparameter	Value
Target train batch size	128
Num episodes per iteration	512
Dataset samples per iteration	16
Samples (per prompt)	-
Cutpoint interval	-
K (MC samples)	-
M (Tokens per level)	30
Branch factors	[6,6,6]
Train policy temperature	0.6
Train policy top-p	0.9
Train context size	2047
Initial KL coef	0.0001
Learning rate	1×10^{-6}
Epochs per iteration	1
Prob mask threshold	0.9
Eval. context size	2047
Eval. temperature	0.35
Eval. top-p	0.9
Eval. samples	16

GRPO DeepSeek-R1-Distill-Qwen 1.5B on MATH

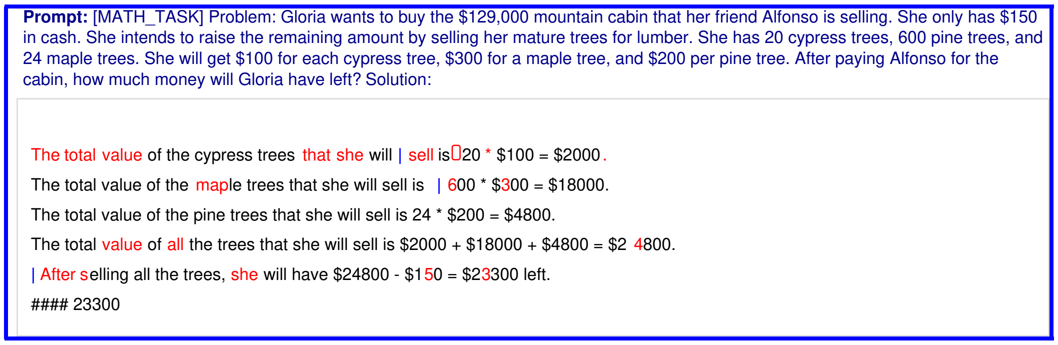
Hyperparameter	Value
Target train batch size	128
Num episodes per iteration	512
Dataset samples per iteration	64
Samples (per prompt)	8
Cutpoint interval	-
K (MC samples)	-
M (Tokens per level)	-
Branch factors	-
Train policy temperature	0.6
Train policy top-p	1
Train context size	$2048 \to 4096$
Initial KL coef	0.0001
Learning rate	1×10^{-6}
Epochs per iteration	1
Prob mask threshold	No prob mask
Eval. context size	$2048 \to 4096$
Eval. temperature	0
Eval. top-p	0.9
Eval. samples	1

SPO-tree (6-6-6) DeepSeek-R1-Distill-Qwen 1.5B on MATH

Hyperparameter	Value
Target train batch size	128
Num episodes per iteration	1024
Dataset samples per iteration	16
Samples (per prompt)	-
Cutpoint interval	-
K (MC samples)	-
M (Tokens per level)	600
Branch factors	[6,6,6]
Train policy temperature	0.6
Train policy top-p	1
Train context size	$2048 \to 4096$
Initial KL coef	0.0001
Learning rate	1×10^{-6}
Epochs per iteration	1
Prob mask threshold	0.9
Eval. context size	$2048 \to 4096$
Eval. temperature	0
Eval. top-p	0.9
Eval. samples	1

SPO-chain (int5→10) DeepSeekMath 7B on MATH

Hyperparameter	Value
Target train batch size	64
Num episodes per iteration	512
Dataset samples per iteration	64
Samples (per prompt)	8
Cutpoint interval	$10 \to 5$
K (MC samples)	9
M (Tokens per level)	-
Branch factors	-
Train policy temperature	0.6
Train policy top-p	0.9
Train context size	4095
Initial KL coef	0.0001
Learning rate	1×10^{-6}
Epochs per iteration	2
Prob mask threshold	0.9
Eval. context size	4095
Eval. temperature	0.35
Eval. top-p	0.9
Eval. samples	16


Prompt: [MATH_TASK] Problem: Gloria wants to buy the $129,000 mountain cabin that her friend Alfonso is selling. She only has $150 in cash. She intends to raise the remaining amount by selling her mature trees for lumber. She has 20 cypress trees, 600 pine trees, and 24 maple trees. She will get $100 for each cypress tree, $300 for a maple tree, and $200 per pine tree. After paying Alfonso for the cabin, how much money will Gloria have left? Solution:

The total value of the cypress trees that she will | sell is 20 * $100 = $2000.

The total value of the maple trees that she will sell is | 600 * $300 = $18000.

The total value of the pine trees that she will sell is 24 * $200 = $4800.

The total value of all the trees that she will sell is $2000 + $18000 + $4800 = $2 4800.

| After selling all the trees, she will have $24800 - $150 = $23300 left.

23300

Figure 11: Segment partition based on fixed *cutpoint* interval. Each row corresponds to a single step. Tokens marked in red indicate cutpoints (tokens with a probability lower than 0.9). The blue vertical lines represent SPO segment boundaries.

F Segment Partition Example

Figure 11 shows a practical segment partition example. As we can see, many of the cutpoints (tokens whose probability is below a threshold) correspond to the places where the model makes mistakes. For instance, in the second step, the total value of the maple trees should be 24 * $300, but the model outputs 600 * 300 and shows low confidence at the digit 6. If it had output 2 in this position, this step might have been correct. Similarly, in the final step, the calculation should be 24800 - 12900, but the model instead outputs 24800 - 150 and is uncertain at the digit 5. Had it produced a 2 at this point, the equation might have been correct. This case shows that cutpoints are the locations where the model's reasoning trajectory can shift, and they are the main drivers behind "segment advantage". Therefore, identifying and accurately assigning credit to these cutpoints is essential, which is consistent with our segment partition method 5.1 and probability mask technique 5.3.

It is also worth noting that partitioning by line breaks, as in VinePPO, is not the most effective approach. For example, in step three, none of the tokens have probabilities below 0.9, indicating that the transition probabilities for all tokens in this step are very high. As a result, the estimated V value at the beginning and end of this step will not differ significantly, thus the advantage for this segment will be close to zero. Allocating sampling budget to this step is likely to result in wasted samples.

Moreover, partitioning by step boundaries can lead to an uneven distribution of cutpoints, i.e., some segments may contain too many, while others include too few. Our goal in segment partition is to have each segment contain roughly the same number of potential "turning points", so that we can assign credit as efficiently as possible within a limited sampling budget. Step-based partition does not achieve this balance. This may help explain why our SPO-chain method can outperform VinePPO, even with fewer sampling points.

G Examples of Model Outputs

Compared to DeepSeek-R1-Distill-Qwen-1.5B, our optimized model demonstrates greater token efficiency, requiring fewer tokens to arrive at the correct answer (both models use greedy decoding). When the model's context length is limited, increased token efficiency significantly enhances performance.

Problem

What is the smallest positive integer n such that $17n \equiv 1234 \pmod{7}$?

SPO-tree

Okay, so I need to find the smallest positive integer n such that $17n \equiv 1234 \pmod{7}$. Hmm, let me think about how to approach this. I remember that when dealing with congruences modulo a number, it's often helpful to simplify the coefficients modulo that number first. So, maybe I should start by reducing 17 and 1234 modulo 7.
Let me recall that to find a number modulo 7, I can divide by 7 and find the remainder. So, let's start with 17. Dividing 17 by 7, 7 goes into 17 twice with a remainder. 2 times 7 is 14, and 17 minus 14 is 3. So, 17 is congruent to 3 modulo 7. That means $17 \equiv 3 \pmod{7}$. So, the equation $17n \equiv 1234 \pmod{7}$ simplifies to $3n \equiv 1234 \pmod{7}$.
Now, I need to find 1234 modulo 7. Let me do that. 1234 divided by 7. Hmm, 7 times 176 is 1232 because 176 times 7 is 1232. So, 1234 minus 1232 is 2. Therefore, $1234 \equiv 2 \pmod{7}$. So, now the equation is $3n \equiv 2 \pmod{7}$.
Alright, so I have $3n \equiv 2 \pmod{7}$. I need to solve for n. To do that, I think I need to find the multiplicative inverse of 3 modulo 7. The multiplicative inverse of 3 is a number x such that $3x \equiv 1 \pmod{7}$. Once I have that, I can multiply both sides of the equation by x to get $n \equiv 2x \pmod{7}$.
So, let me find the inverse of 3 modulo 7. Let's try numbers from 1 upwards:
- $3 \times 1 = 3 \equiv 3 \pmod{7} \rightarrow$ Not 1. - $3 \times 2 = 6 \equiv 6 \pmod{7} \rightarrow$ Not 1. - $3 \times 3 = 9 \equiv 2 \pmod{7} \rightarrow$ Not 1. - $3 \times 4 = 12 \equiv 5 \pmod{7} \rightarrow$ Not 1. - $3 \times 5 = 15 \equiv 1 \pmod{7} \rightarrow$ Yes!
So, 5 is the inverse because 3*5=15, which is 1 mod 7.
Great, so the inverse of 3 mod 7 is 5. Therefore, multiplying both sides of $3n \equiv 2 \pmod{7}$ by 5 gives:
$5 \times 3n \equiv 5 \times 2 \pmod{7}$
Simplifying, $15n \equiv 10 \pmod{7}$. But 15 mod 7 is 1, because 7*2=14, so 15-14=1. Similarly, 10 mod 7 is 3, because 7*1=7, 10-7=3. So, this simplifies to:
$1n \equiv 3 \pmod{7}$
Which means $n \equiv 3 \pmod{7}$. So, the smallest positive integer n that satisfies this is 3. Let me just verify that.
Calculating $17 * 3 = 51$. Then, 51 mod 7: 7*7=49, so 51-49=2. And 1234 mod 7 is 2, as we found earlier. So, 51 mod 7 is 2, which matches 1234 mod 7. So, yes, 3 is the correct answer.
Final Answer The smallest positive integer n is $\boxed{3}$.
To find the smallest positive integer n such that $17n \equiv 1234 \pmod{7}$, we start by simplifying the coefficients modulo 7.

First, we reduce 17 modulo 7:

$$17 \div 7 = 2 \text{ remainder } 3 \implies 17 \equiv 3 \pmod{7}$$

Thus, the equation $17n \equiv 1234 \pmod{7}$ simplifies to:

$$3n \equiv 1234 \pmod{7}$$

Next, we reduce 1234 modulo 7:

$$1234 \div 7 = 176 \text{ remainder } 2 \implies 1234 \equiv 2 \pmod{7}$$

So, the equation becomes:

$$3n \equiv 2 \pmod{7}$$

To solve for n, we need the multiplicative inverse of 3 modulo 7. Testing values, we find that:

$$3 \times 5 = 15 \equiv 1 \pmod{7}$$

Thus, the inverse of 3 modulo 7 is 5. Multiplying both sides of $3n \equiv 2 \pmod{7}$ by 5:

$$5 \times 3n \equiv 5 \times 2 \pmod{7} \implies 15n \equiv 10 \pmod{7}$$

Simplifying, since $15 \equiv 1 \pmod{7}$ and $10 \equiv 3 \pmod{7}$:

$$n \equiv 3 \pmod{7}$$

The smallest positive integer n is 3. Verifying, $17 \times 3 = 51 \equiv 2 \pmod{7}$ and $1234 \equiv 2 \pmod{7}$, confirming the solution.

$$\boxed{3}$$

DeepSeek-R1-Distill-Qwen-1.5B

Okay, so I have this problem here: find the smallest positive integer n such that $17n \equiv 1234 \pmod{7}$. Hmm, modular arithmetic, right? I remember that congruences modulo 7 mean that when you divide $17n - 1234$ by 7, there's no remainder. So, basically, $17n$ and 1234 leave the same remainder when divided by 7.
First, maybe I should simplify both sides of the congruence modulo 7. That is, find what 17 is equivalent to modulo 7, and what 1234 is equivalent to modulo 7. That should make the equation easier to handle.
Starting with 17 modulo 7. Let me divide 17 by 7. 7 goes into 17 twice, which is 14, and the remainder is 3. So, $17 \equiv 3 \pmod{7}$. That simplifies the left side of the congruence to $3n$.
Now, the right side is 1234 modulo 7. Hmm, 1234 is a big number, so I need a systematic way to find its remainder when divided by 7. Maybe I can break it down digit by digit or use some divisibility rules. Alternatively, I can use the fact that 10 is congruent to 3 modulo 7, so powers of 10 can be simplified.
Wait, another approach: I can divide 1234 by 7 and find the remainder. Let me do that. 7 goes into 12 once, with a remainder of 5. Bring down the 3 to make 53. 7 goes into 53 seven times, which is 49, leaving a remainder of 4. Bring down the 4 to make 44. 7 goes into 44 six times, which is 42, leaving a remainder of 2. So, 1234 divided by 7 is 176 with a remainder of 2. Therefore, $1234 \equiv 2 \pmod{7}$.
So now, the congruence simplifies to $3n \equiv 2p \mod 7$. Okay, so I need to solve for n in this equation. That is, find the smallest positive integer n such that when multiplied by 3 and then divided by 7, the remainder is 2.
To solve $3n \equiv 2 \pmod{7}$, I think I need to find the multiplicative inverse of 3 modulo 7. The multiplicative inverse of 3 is a number x such that $3x \equiv 1 \pmod{7}$. Once I have that, I can multiply both sides of the congruence by x to isolate n.
So, let's find the inverse of 3 modulo 7. I can test numbers from 1 upwards to see which one satisfies $3x \equiv 1 \pmod{7}$.

- $3 \times 1 = 3 \equiv 3 \pmod 7$ → Not 1. - $3 \times 2 = 6 \equiv 6 \pmod 7$ → Not 1. - $3 \times 3 = 9 \equiv 2 \pmod 7$ → Not 1. - $3 \times 4 = 12 \equiv 5 \pmod 7$ → Not 1. - $3 \times 5 = 15 \equiv 1 \pmod 7$ → Yes! That works.

So, the inverse of 3 modulo 7 is 5. Therefore, multiplying both sides of the congruence $3n \equiv 2 \pmod 7$ by 5 gives:

$5 \times 3n \equiv 5 \times 2 \pmod 7$

Simplifying the left side: $15n \equiv 2 \pmod 7$. But 15 modulo 7 is 1, since 15 divided by 7 is 2 with a remainder of 1. So, this simplifies to:

$1n \equiv 2 \pmod 7$, which is just $n \equiv 2 \pmod 7$.

Therefore, the smallest positive integer n that satisfies this congruence is 2. Let me double-check to make sure I didn't make a mistake.

Plugging $n = 2$ back into the original equation: $17 \times 2 = 34$. Now, 34 divided by 7 is 4 with a remainder of 6. Wait, but 1234 modulo 7 was 2, so 34 modulo 7 is 6, which is not equal to 2. Hmm, that's a problem. Did I do something wrong?

Wait, hold on. Let me check my steps again. Maybe I messed up the inverse or the simplification.

Starting over: $17n \equiv 1234 \pmod 7$. I found that $17 \equiv 3 \pmod 7$ and $1234 \equiv 2 \pmod 7$. So, $3n \equiv 2 \pmod 7$. Then, I found the inverse of 3 modulo 7 is 5 because $3 \times 5 = 15 \equiv 1 \pmod 7$. So, multiplying both sides by 5: $n \equiv 10 \pmod 7$. But 10 modulo 7 is 3, so $n \equiv 3 \pmod 7$. Wait, so earlier I thought it was 2, but that was a mistake.

Wait, no, let me recast that. If $n \equiv 10 \pmod 7$, then 10 divided by 7 is 1 with a remainder of 3, so $n \equiv 3 \pmod 7$. So, the smallest positive integer n is 3. Let me check that.

Plugging $n = 3$ into $17n$: $17 \times 3 = 51$. 51 divided by 7 is 7 with a remainder of 2. So, 51 modulo 7 is 2, which matches the right side of the congruence. So, that works.

Wait, so earlier I thought the inverse was 5, but that led me to $n \equiv 10 \pmod 7$, which is 3. So, the correct answer is 3, not 2. I must have made a mistake in my initial calculation.

Let me go through the inverse part again. So, to find the inverse of 3 modulo 7, I need a number x such that $3x \equiv 1 \pmod 7$. Testing:

- $3 \times 1 = 3 \equiv 3 \pmod 7$ - $3 \times 2 = 6 \equiv 6 \pmod 7$ - $3 \times 3 = 9 \equiv 2 \pmod 7$ - $3 \times 4 = 12 \equiv 5 \pmod 7$ - $3 \times 5 = 15 \equiv 1 \pmod 7$

Yes, so 5 is the inverse. So, multiplying both sides by 5: $5 \times 3n \equiv 5 \times 2 \pmod 7$, which is $15n \equiv 10 \pmod 7$. But 15 modulo 7 is 1, so $n \equiv 10 \pmod 7$. 10 divided by 7 is 1 with a remainder of 3, so $n \equiv 3 \pmod 7$. So, the smallest positive integer is 3.

Wait, so why did I get confused earlier? Because I thought 10 modulo 7 is 3, but I thought 10 was 3, but when I first thought of 10, I thought it was 3, but when I first thought of 10, I thought it was 3, but when I thought of 10, I thought it was 3, but when I thought of 10, I thought it was 3, but when I thought of 10, I thought it was 3, but when I thought of 10, I thought it was 3. Wait, no, that's not right.

Wait, 10 divided by 7 is 1 with a remainder of 3, so 10 modulo 7 is 3. So, $n \equiv 3 \pmod 7$. So, the smallest positive integer is 3.

Let me verify again: $17 \times 3 = 51$. 51 divided by 7 is 7 with a remainder of 2. So, 51 modulo 7 is 2, which is equal to 1234 modulo 7, which is 2. So, that works.

Wait, so earlier, when I thought I had 10 modulo 7, I thought it was 3, but I think I confused myself. So, the correct answer is 3.

But just to make sure, let me try $n = 3$: $17 \times 3 = 51$. 51 divided by 7 is 7*7=49, so 51-49=2. So, 51 mod7=2, which is equal to 1234 mod7=2. So, yes, that works.

Wait, but earlier, when I thought of 10 modulo7, I thought it was 3, but that was a miscalculation. So, the correct inverse is 5, leading to $n \equiv 10 \pmod 7$, which is 3.

So, the smallest positive integer is 3.

But just to make sure, let me try $n = 3$ in the original equation: $17 \times 3 = 51$. 51 mod7 is 2, which is equal to 1234 mod7, which is 2. So, yes, that works.

Wait, but let me also check $n = 3 - 7 = -4$. But since we're looking for positive integers, 3 is the smallest.

Alternatively, if I had a different inverse, but no, 5 is the inverse.

Wait, but just to make sure, let me try $n = 3 + 7 = 10$. $17 \times 10 = 170$. 170 divided by 7 is 24*7=168, so 170-168=2. So, 170 mod7=2, which is equal to 1234 mod7=2. So, that also works, but 10 is larger than 3, so 3 is the smallest.

Therefore, the answer is 3.
Wait, but hold on, in my initial calculation, I thought I had 10 modulo7=3, but I thought 10 was 3, but that was a confusion. So, the correct answer is 3.
But just to make sure, let me try another approach. Maybe I can solve $3n \equiv 2 \pmod 7$ by testing n=1,2,3,... until I find the smallest n that satisfies the equation.
n=1: $3 * 1 = 3 \equiv 3 \neq 2 \pmod 7$ n=2: $3 * 2 = 6 \equiv 6 \neq 2 \pmod 7$ n=3: $3 * 3 = 9 \equiv 2 \pmod 7$. Yes! So, n=3 is the smallest positive integer.
So, that confirms it. So, the answer is 3.
Wait, but in my initial thought process, I thought I had 10 modulo7=3, but that was a miscalculation. So, the correct answer is 3.
Therefore, the smallest positive integer n is 3.
Final Answer The smallest positive integer n is $\boxed{3}$.
To find the smallest positive integer n such that $17n \equiv 1234 \pmod 7$, we start by simplifying both sides of the congruence modulo 7.
First, we simplify $17 \mod 7$:

$$17 \div 7 = 2 \text{ remainder } 3$$

Thus, $17 \equiv 3 \pmod 7$.
Next, we simplify $1234 \mod 7$. We can do this by dividing 1234 by 7 and finding the remainder:

$$1234 \div 7 = 176 \text{ remainder } 2$$

Thus, $1234 \equiv 2 \pmod 7$.
This reduces our original congruence to:

$$3n \equiv 2 \pmod 7$$

To solve for n, we need the multiplicative inverse of 3 modulo7 . We find this by testing values:

$$3 \times 5 = 15 \equiv 1 \pmod 7$$

Thus, the inverse of 3 modulo 7 is 5.
Multiplying both sides of the congruence $3n \equiv 2 \pmod 7$ by 5:

$$5 \times 3n \equiv 5 \times 2 \pmod 7$$

$$15n \equiv 10 \pmod 7$$

Since $15 \equiv 1 \pmod 7$, this simplifies to:

$$n \equiv 10 \pmod 7$$

$$10 \div 7 = 1 \text{ remainder } 3$$

Thus, $n \equiv 3 \pmod 7$.
The smallest positive integer n is 3. Verifying, we find:

$$17 \times 3 = 51$$

$$51 \div 7 = 7 \text{ remainder } 2$$

Thus, $51 \equiv 2 \pmod 7$, confirming our solution.
The smallest positive integer n is $\boxed{3}$.

H Pseudo-code of Tree-Structured Advantage Estimation Method

We efficiently construct the tree using Python's asyncio library. For the detailed algorithm procedure, please refer to Algorithm 1 below.

Algorithm 1 Tree-structured Advantage Estimation Method

```
1: procedure ConstructTree($prompt$, $max\_depth$, $structure$, $M$, $instance$, $adv\_method$)
2:     create root node:
3:         $root.text \leftarrow prompt$
4:         $root.full\_text \leftarrow prompt$
5:         $root.depth \leftarrow 0$
6:     await Build($root$, $prompt$, $0$, $max\_depth$, $structure$, $M$, $instance$)
7:     ComputeAdvantage($root$, None, $adv\_method$)
8:     return $root$
9: end procedure
10: procedure Build($node$, $prefix$, $depth$, $max\_depth$, $structure$, $M$, $instance$)
11:     if $depth \geq max\_depth$ then
12:         $node.reward \leftarrow$ REWARD_FUNCTION($prefix$, $node.text$, $instance$)
13:         return
14:     end if
15:     $K \leftarrow structure[depth]$
16:     if $depth < max\_depth - 1$ then
17:         $max\_tokens \leftarrow M$
18:     else
19:         $max\_tokens \leftarrow$ None
20:     end if
21:     $children \leftarrow$ await Expand($prefix$, $depth$, $K$, $max\_tokens$)
22:     $node.children \leftarrow children$
23:     initialize empty list $expansion\_tasks$
24:     for each $child$ in $children$ do
25:         if $child.finish\_reason \neq$ "length" then
26:             $child.reward \leftarrow$ REWARD_FUNCTION($prefix$, $child.text$, $instance$)
27:         else
28:             create async task $t \leftarrow$ Build($child$, $child.full\_text$, $depth + 1$, $max\_depth$,
                                        $structure$, $M$, $instance$)
29:             append $t$ to $expansion\_tasks$
30:         end if
31:     end for
32:     await GATHER($expansion\_tasks$)
33:     $node.reward \leftarrow$ mean(rewards of $node.children$)
34:     $node.reward\_std \leftarrow$ std(rewards of $node.children$)
35: end procedure
36: procedure Expand($prefix$, $depth$, $K$, $max\_tokens$)
37:     construct LLM prompt from template using $prefix$
38:     create async task calling LLM API requesting $K$ responses, each limited to $max\_tokens$
39:     $responses \leftarrow$ await $task$
40:     initialize empty list $children$
41:     for each $response$ in $responses$ do
42:         create new node $child$:
43:             $child.text \leftarrow response.text$
44:             $child.finish\_reason \leftarrow response.finish\_reason$
45:             $child.full\_text \leftarrow$ concatenate_texts($prefix$, $child.text$)
46:         append $child$ to $children$
47:     end for
48:     return $children$
49: end procedure
50: procedure ComputeAdvantage($node$, $parent$, $adv\_method$)
51:     if $parent$ is not $None$ then
52:         if $adv\_method =$ "RLOO" then
53:             $node.advantage \leftarrow node.reward - parent.reward$
54:         else if $adv\_method =$ "GRPO" then
55:             $node.advantage \leftarrow (node.reward - parent.reward)/parent.reward\_std$
56:         end if
57:     end if
58: end procedure
```