

TreePO: Bridging the Gap of Policy Optimization and Efficacy and Inference Efficiency with Heuristic Tree-based Modeling

ByteDance Seed, M-A-P, UoM

Full author list in Contributions

Abstract

Recent advancements in aligning large language models via reinforcement learning have achieved remarkable gains in solving complex reasoning problems, but at the cost of expensive on-policy rollouts and limited exploration of diverse reasoning paths. In this work, we introduce TreePO, involving a self-guided rollout algorithm that views sequence generation as a tree-structured searching process. Composed of dynamic tree sampling policy and fixed-length segment decoding, TreePO leverages local uncertainty to warrant additional branches. By amortizing computation across common prefixes and pruning low-value paths early, TreePO essentially reduces the per-update compute burden while preserving or enhancing exploration diversity. Key contributions include: (1) a segment-wise sampling algorithm that alleviates the KV cache burden through contiguous segments and spawns new branches along with an early-stop mechanism; (2) a tree-based segment-level advantage estimation that considers both global and local proximal policy optimization. and (3) analysis on the effectiveness of probability and quality-driven dynamic divergence and fallback strategy. We empirically validate the performance gain of TreePO on a set reasoning benchmarks and the efficiency saving of GPU hours from 22% up to 43% of the sampling design for the trained models, meanwhile showing up to 40% reduction at trajectory-level and 35% at token-level sampling compute for the existing models. While offering a free lunch of inference efficiency, TreePO reveals a practical path toward scaling RL-based post-training with fewer samples and less compute. Home page locates at <https://m-a-p.ai/TreePO>.

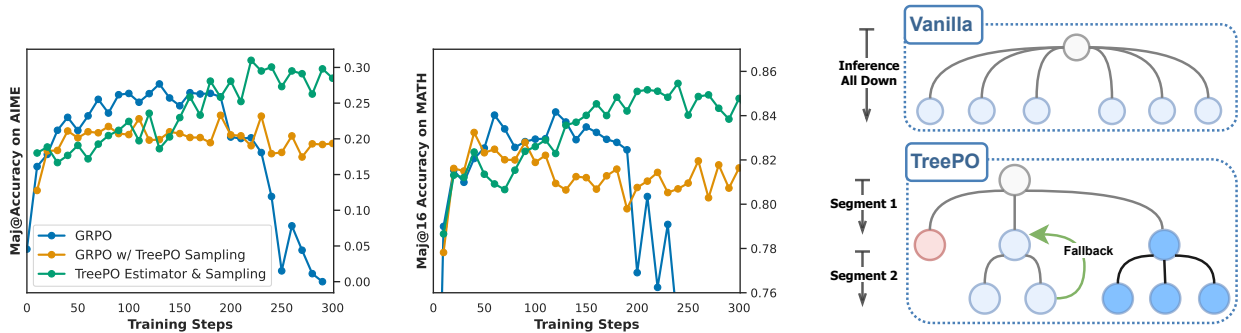


Figure 1 Demonstration of the Validation Performance Curves along Training based on Qwen2.5-7B (*Left, Mid*) and Demonstration of TreePO Sampling (*Right*). *Left, Mid*: Compared to the GRPO setting, although replaced additional treed-based sampling causes a slower convergence, it could stabilize the training. When cooperate the health.

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for enhancing the complex reasoning abilities of Large Language Models (LLMs) [1–3]. However, the efficacy and scalability of RL face fundamental constraints from two long-standing challenges: exploration (generating diverse responses) and exploitation (obtaining guidance from external feedback). In the context of LLMs, these challenges become even more pronounced, as models must generate sequences spanning thousands of tokens before receiving a single reward signal—which is typically sparse and delayed [4, 5]. This constraint creates two critical research challenges: (1) How can we enable LLMs to explore potentially correct reasoning paths while maintaining or reducing computational costs? and (2) How can we accurately attribute sparse outcome rewards to the specific tokens that contributed to correct answers?

We present key observations that inspire our approach to addressing these challenges: standard RL approaches typically generate multiple independent trajectories for a single query—a strategy that is both computationally inefficient and conceptually sub-optimal. From a computational perspective, this approach creates paths with separate Key-Value (KV) caches, failing to utilize shared KV caching mechanisms that could significantly accelerate inference. Conceptually, continuing to explore paths already known to be impossible or incorrect, without early termination, represents a critical limitation in adaptability. That is, while this sampling strategy may appear simple to implement, its lack of structural design ultimately limits its effectiveness.

A promising sampling strategy is Monte Carlo Tree Search (MCTS) [6] or its variants [7, 8], which enables agents to leverage tree structures to achieve functions like early termination and roll back. Despite its promise, MCTS is often inefficient for LLM inference, requiring numerous sequential rollouts that are poorly suited for parallelized engines. Recent efforts have moved toward better utilization of LLM inference engines, recognizing that optimizing the data generation process itself is a critical frontier [9, 10]. We believe this is the correct direction and accordingly propose a heuristic, self-guided, tree-based sampling mechanism designed to fully leverage the Key-Value (KV) cache mechanism. By structuring the rollout process as a tree, we maximize the reuse of shared prefixes as demonstrated in the Figure 1 (Right). Our findings show this approach can averagely reduce 40% of trajectory-level inference time for the baselines (see §4.1), thereby improving computational efficiency without sacrificing performance.

To address the second question of credit assignment, our tree-based sampling structure naturally facilitates a more granular advantage estimation. This allows us to propose a new advantage function that is distinct from recent related works like TreeRL [11] and SPO [12]. While these methods also leverage tree or segment-based structures, their advantage calculations are primarily MCTS-like, focusing on the value difference between a parent and its child node to assign credit. Our approach, in contrast, models entire sub-trees as coherent sub-groups, enabling a more robust relative advantage calculation based on the collective outcomes of all descendants. More critically, our design is proven to be feasible for training directly from a base model, aligning with the "RL-zero" paradigm where reasoning capabilities are elicited without prior supervised fine-tuning (SFT). This stands in contrast to the mentioned peers, which are demonstrated on models that have already undergone SFT.

In this paper, we introduce Tree-based Policy Optimization (TreePO), a framework that integrates these solutions into a unified RL pipeline. TreePO replaces inefficient independent rollouts with a computationally efficient and algorithmically flexible tree search. This structure not only improves sampling efficiency but also enables principled credit assignment and controllable exploration. We introduce novel heuristic sampling strategies, including dynamic divergence and probability-based fallback, which strategically allocate the generation budget to explore more diverse and promising reasoning paths. This transforms the rollout phase into a transparent and controllable search process, providing a powerful tool for analyzing the training dynamics of RL models. In summary, our contributions are:

- We introduce TreePO, a novel RL training scheme that replaces standard i.i.d. sequential sampling with a heuristic tree-based rollout mechanism. By implementing heuristic-driven exploration strategies, including dynamic divergence and probability-based fallback, this mechanism enhances the model’s ability to explore the reasoning space effectively while significantly improving computational efficiency by leveraging KV-caching.

- We propose a new tree-based advantage estimation function that enables more precise credit assignment and is uniquely suited for training LLMs from a base model, without requiring an initial instruction tuning stage.
- We demonstrate through extensive experiments that TreePO provides a superior trade-off between computational cost and model performance, establishing a more efficient and scalable frontier for training large reasoning models.

2 TreePO: A Tree-based Training Scheme for Policy Optimization

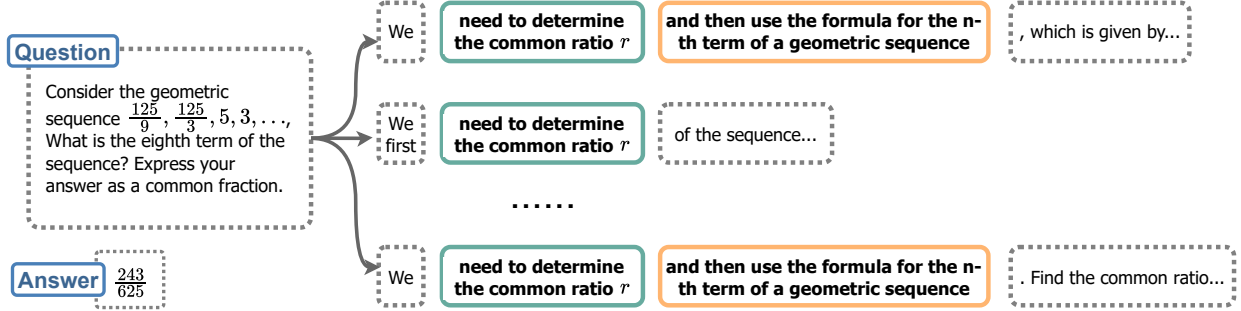


Figure 2 Multiple sampled trajectories from the same prompt, with shared reasoning segments highlighted in matching colors. Despite stochastic generation, key problem-solving steps are consistently reproduced.

2.1 Case Study: The Aligned Model Produces Shared Prefix

We begin with an empirical observation on the structure of reasoning trajectories. Given a fixed prompt, we perform 16 independent stochastic rollouts using a temperature of 0.8 to encourage diverse generation while preserving coherence. Upon close inspection, we find that despite the variation in final solutions, the generated trajectories share extensive overlapping segments, particularly in the early and intermediate stages of reasoning. As illustrated in Figure 2, components such as problem interpretation, variable assignment, and initial logical deductions appear nearly identical across multiple rollouts. These recurring segments are highlighted with consistent colors, visually demonstrating the emergence of stable reasoning prefixes.

This phenomenon indicates that, even under stochastic sampling, the model consistently follows a common path for the initial stages of reasoning before diverging at later decision points. Such redundancy across trajectories suggests a fundamental inefficiency in standard on-policy reinforcement learning: each rollout independently recomputes the same prefix tokens, leading to duplicated computation and KV cache storage. Since reasoning paths naturally form a tree-like structure where common prefixes branch into diverse continuations, it is both feasible and highly beneficial to model sequence generation as a tree-structured search process. By explicitly representing shared prefixes only once and amortizing computation over them, using TreePO avoids redundant forward passes. Furthermore, the natural branching points provide ideal locations for uncertainty-driven exploration, enabling efficient and targeted expansion of reasoning paths.

2.2 Tree-based Rollout Algorithm

Preliminaries. For a given query $q_i \in Q$, we formalize the problem of complex reasoning with chain of thought (CoT) [13] as the search algorithm to acquire a group of corresponding answers, $o_{i,j} \in O$, under a certain constraint of the computing budget. Specifically, we define the exact input of model as a prompt p , to distinguish the query itself as the input might contain additional context. In the TreePO sampling setting, we align terminology of RLVR and tree search to define:

1. the query q as the root node at depth 0;
2. the number of complete trajectories as the tree width, w ;

Algorithm 1 Tree-based Sampling

Require: An array of queries $Q = \{q_1, q_2, \dots, q_n\}$

Ensure: Rollout responses O that satisfy the budget requirement for all $q \in Q$.

```
1:  $P \leftarrow Q$  ▷ Init inference prompts with queries
2:  $P \leftarrow \text{BRANCHING}(P)$  ▷ Fork the prompts with designed policy
3: while  $P \neq \emptyset$  do
4:    $S \leftarrow \text{INFERENCE}(P)$  ▷ Inference one step
5:    $P^{last} \leftarrow P$ 
6:    $P \leftarrow \emptyset$  ▷ Clean up the inference queue
7:   for  $s_k$  in  $S$  do ▷ Iterate through the generated segments
8:     if  $\text{FINISH}(s_k)$  or  $\text{FAILED\_NODE}(s_k)$  then
9:        $O \leftarrow O \cup \{p_k^{last} \oplus s_k\}$  ▷ Build the full response for final output
10:    else
11:       $P \leftarrow P \cup \{p_k^{last} \oplus s_k\}$  ▷ Concatenate the segment as new prompt
12:    end if
13:  end for
14:   $P \leftarrow \text{BRANCHING}(P)$  ▷ Fork the prompts with designed policy
15:   $P \leftarrow \text{FALLBACK}(P, O)$  ▷ Do fallback for insufficient outputs
16: end while
17: return  $O$  ▷ Return the final outputs
```

3. the maximum decoding steps of a trajectory as the depth, d ;
4. the maximum decoding token of each time as the length of the segment, l ; and
5. branching budget b for each segment node.

Under the context of RL training of large language models, the computing budget for sampling the trajectories of a given set of queries could be defined by the trajectory group size of each query (also noted as the tree width w), if the maximum trajectory length $d \times l$ is fixed.

Segment-level Tree Sampling. As shown in Figure 1 (Upper Right), the vanilla sampling design requires the model to conduct *token-level* decoding and stem multiple complete trajectories from the same query independently. We re-organize such a sampling progress into a hybrid of *segment-level* tree searching and *token-level* decoding as in Figure 1 (Lower Right): for each trajectory, the model generates a segment s in max length l step by step, until it hits the maximum response length or meets the *any self-designed criteria* of early stopping. We maintain a queue of prompts P to manage the sampling progress, and assign the queries as the initial prompt set. For an input query set q , the *token-level* decoding stops when the model generates [EOS] token or reaches the preset maximum segment token l ; and the overall *segment-level* tree sampling progress ends when the prompt queue becomes empty ($P = \emptyset$). Specifically, given a P in each step of decoding, the inference engine would produce a set of output segments in the exact number of $|P|$. And each generated segment will be either **appended** to existing contexts to form a new input prompt in the queue, or **stop generation** as a leaf node if it contains flawed sub-string patterns or answer **boxed**. We introduce the branching of each search paths by forking the corresponding prompts b times before segment inference, where the value of b is dynamically calculated and assigned by design (see the details in the following literature). To fulfill the requirement of acquiring w trajectories for each q when the searching paths stop early before the tree reaches w , we introduce the feedback mechanism and stem new branches from the stopped paths to achieve better efficiency.

Branching and Fallback. After reformulating the sampling progress into a tree-based search, a subtle balance between the rollout efficiency and model exploration space could be achieved by a well defined branching and fallback protocol. In TreePO, we define a vanilla N -ary tree as a baseline searching strategy, i.e., the branching budget for a the root node q (query) at depth d is N^d until it reaches the maximum width w . To

avoid the inference loading skew caused by the scarce long responses and the over-bias on the short paths, we coordinate two balancing tree searching strategies with the inference engine:

1. **Branching Budget Transfer:** As early stopped short search paths could derive a small request batch to the inference engine and thus cause low utilization, we assign the maximum branching budget N^d at depth d to all existing active paths evenly (or determined by heuristic information).
2. **Depth-First Search Fallback:** To avoid sampling progress overly conducts fallback on the early stopped short paths and lose the capability of long complex reasoning, TreePO launches the fallback mechanism only when there is no active path for q and the tree does not have enough trajectories $w_q < w$.

Heuristic Sampling. With the designed segment-level tree sampling protocol, we can now accordingly introduce a more fine-grained and flexible control over the sampling progress with heuristic information. Without waiting for external signals, the TreePO sampling could exploit more in the desired search space by leveraging heuristic control on early stopping, branching, and fallback strategies. We first introduce a simple early stopping trick for the flawed searching path by detecting the pattern with repetitive substrings within the new generated segment, which could reduce redundant computing, and forcedly prune the branches within the mumbling distribution that are usually generated by the less aligned base models. While conducting fallback, only those stopped paths containing formatted answer or ending with [EOS] can be selected as the candidate to randomly fallback in segment level. Other than the average branching budget assignment and random fallback strategy, there are more possible customized heuristic metrics could be applied when maintaining efficiency of TreePO, as long as no additional bubble of the pipeline is introduced. In the later §4.4, we take advantage of the log probabilities to steer the sampling progress without additional cost, as they are calculated during token-level decoding and returned from the inference engines by default.

2.3 Tree-based Advantage Estimation for Policy Optimization

We take the GRPO [2] optimizing objective and adopt the improved modifications proposed in DAPO [3] as our starting point, which further highlights clip-higher gradient, dynamic sampling, and token-level loss:

$$\begin{aligned} \mathcal{J}_{\text{TreePO}}(\theta) = & \mathbb{E}_{(q,a) \sim \mathcal{D}, \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)} \\ & \left[\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip} \left(r_{i,t}(\theta), 1 - \varepsilon_{\text{low}}, 1 + \varepsilon_{\text{high}} \right) \hat{A}_{i,t} \right) \right], \\ \text{s.t. } & 0 < \left| \{o_i \mid \text{is_equivalent}(a, o_i)\} \right| < G. \end{aligned} \quad (1)$$

where

$$r_{i,t}(\theta) = \frac{\pi_{\theta}(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}, \quad \hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^G)}{\text{std}(\{R_i\}_{i=1}^G)}. \quad (2)$$

Although the delicate modifications in DAPO [3] largely improve the stability of the vanilla GRPO, the parallel-generated responses could still look “homogeneous” in the sequence-level to the policy model under certain circumstances (e.g., inference with low temperature or train with an over-confident model). Benefiting from the tree structure in the proposed rollout algorithm, the searching paths could be sourced during advantage calculation. Given arbitrary trajectory o_i , it can be divided into multiple segments S_j by its inference step j :

$$\begin{aligned} o_i &= s_1 \oplus s_2 \oplus \cdots \oplus s_{j-1} \oplus s_j, \\ \{j \in J \mid j \leq \max \text{depth}\} \end{aligned} \quad (3)$$

Such a prior allow us to reveal the nuanced segment-level difference among the trajectories, and introduce more accurate intra-response variations for the advantages to alleviate the obscurity brought by similar responses. Leveraging the shared prefixes, the advantage estimation function for the trajectory could be further calibrated by the subgroups derived from the shared predecessor nodes for the leaf nodes. Let the root

In the context of math reasoning, we set this condition as including a legal answer surrounded by `boxed{}`.

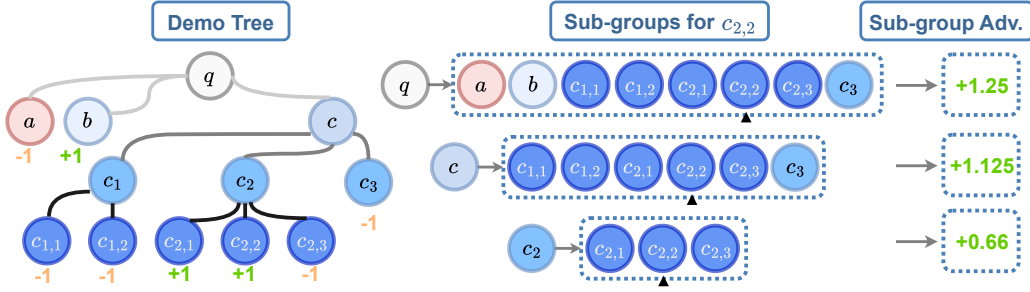


Figure 3 Demonstration of the TreePO Advantage Estimation. Assuming that the tree-based sampling has derived 8 trajectories (leaf nodes) given a query q , we take node $c_{2,2}$ as an example to calculate the sub-group advantages. The tree-based sub-groups could be further defined by its predecessors c_2 , c , and q . Thus the final advantages can be calculated as the averagely aggregated sub-group advantages.

node q be the sharing parent as the largest group G , we could denote a sub-group G_j as the set of trajectories sharing the same predecessor node at inference depth j , satisfying:

$$\begin{aligned} G_{|J|} \subseteq G_{|J|} \subseteq \dots \subseteq G_2 \subseteq G_1 \subseteq G, \\ \{j \in J \mid j < \max \text{ depth}\} \end{aligned} \quad (4)$$

Given the formulated sub-groups, we keep using the average reward within sub-groups as the advantage baselines and conduct mean pooling on the relative advantages as the aggregated estimation. Furthermore, we incorporate the global variance normalization strategy as in REINFORCE++ [14] to improve the robustness of the estimation function, as the probability-based branching could bring potential turbulent rollout rewards across queries, and conduct dynamic rejection sampling to remove the queries with all correct or all wrong responses as in DAPO [3]. Hence the final TreePO advantage estimation function could be depicted as:

$$\begin{aligned} \hat{A}_{i,t} &= \frac{\sum_{j=1}^J \hat{A}_{i,t,j}}{|J| \cdot \text{std}(\{\hat{A}_{i,t,j}\}^{J-1})}, \\ \hat{A}_{i,t,j} &= R_i - \text{mean}(\{R_{i,j}\}^{G_j}), \\ \text{s.t. } \text{std}(\{R_i\}^G) &\neq 0 \end{aligned} \quad (5)$$

3 Experiment

3.1 Hyper Parameters

Model. The main part of the reinforcement training experiments are trained from the Qwen2.5-7B base model [15]. Moreover, to further probe on the efficiency performance of the tree-based sampling on well aligned LLMs, we use the Qwen2.5-7B-Instruct and Qwen2.5-Math-7B-Instruct to compare the vanilla sequential and the tree-based sampling.

Data and Evaluation. One source of the training samples is the the MATH dataset [16], deriving about 8 thousands queries of difficulty level 3 to 5 from, same as the setting in SimpleRL [17]. Another part of the training set consists 40 thousands samples from the DeepScaler [18] collection. For evaluation, we use the AIME 2024 [19], AMC 2023 [20], MATH500[16], MINERVA [21], and Olympiad Bench [22]. During validation and testing, we set the rollout N as 16 and use the majority voting accuracy via 1000 times of sampling as the main metric. For the overall metric, we use the weighted average among the individual benchmarks bases on the sizes of the test sets.

Tree Setting. With the constraint of response length, we search three sets of the depth d and segment token budget l of tree sampling in online training: $\{28 \times 256, 14 \times 512, 7 \times 1024\}$. The fixed branching budget at each depth is set as 2^d , i.e., it will form a binary tree search paths if no early stop happens. And the maximum tree width is set as $w = 16$, where the sequential sampling baselines share the same group size parameter. During training, we explore whether additional initial branching budget by randomly assign 2 to 8 divergences, which is expected to improve the distribution diversity and thus break through the upper bound. We use “More Init Divergence” and “Fixed Init Divergence” to distinguish whether additional initial branching budget is allowed.

Training. We filter out the prompts longer than 1024 tokens, and set the response length as 7×1024 in training. The trainings run on 64 GPUs with the VeRL framework [23] on FSDP mode, and use vLLM [24] as the inference backend. The learning rate is set as $1e - 6$ with 10 warm up steps. And the training batch size are set as 512 with the limit of maximum 20 epoch. The checkpoint saving interval as 50 steps. As the dynamic sampling strategy from DAPO is adopted, queries of $3 \times \text{bsz}$ would be sent to sampling out a group of 16 trajectories, where 512 queries with $\text{std}(\{R_i\}^G) \neq 0$ are randomly selected. When there is not sufficient queries to form a training batch, maximum two other additional samplings will be conducted, which could cause a less training steps due to the enumeration logic of the data loader.

3.2 Main Results

The results of the main experiment set are provided in Table 1, where we use sequential sampling to validate the full potential of the model performances. Based on the provided results and the training curves in Figure 1 (Left, Mid), the introduction of tree-based methods — TreePO sampling and advantage estimator—serves to significantly enhance training stability and computational efficiency, albeit with a trade-off against raw convergence speed and peak accuracy in some configurations.

The effect of Tree Sampling is twofold. First, as shown in Table 1, adding TreePO sampling to the baseline GRPO model provides a substantial performance boost across all datasets, increasing the overall accuracy from 46.63% to 54.61%. This improvement is corroborated by the validation metric curves, where GRPO w/ TreePO Sampling (orange line) demonstrates far greater training stability compared to the volatile performance of the GRPO (blue line). Second, Table 2 reveals that while tree-based sampling does not always outperform a strong sequential baseline in final accuracy (e.g., 58.21% for Sequential vs. 58.06% for TreePO $b=8$ in the “More Init Divergence” model), it consistently and significantly reduces computation time, cutting GPU hours by 12% to 43%.

Beyond that, the TreePO advantage estimator, when used in conjunction with tree sampling, further enhances the training process, either with “More Init Divergence” (3.6% \uparrow) or “Fixed Init Divergence” setting (2.27% \uparrow). The green line in the validation curves shows the most stable and consistently high-performing trajectory during training. This indicates that the estimator component provides a more precise reward signal bases on the tree hierarchy, guiding the model, and leading to more reliable convergence.

Table 1 Performance Comparison with Sequential Sampling with Major@16 Accuracy.

Model	AIME	AMC	MATH	MINERVA	Olympiad Bench	Overall
GRPO	17.13%	44.42%	72.89%	30.94%	35.09%	46.63%
GRPO w/ TreePO Sampling	19.66%	51.63%	81.85%	33.74%	44.76%	54.61%
TreePO w/ Fixed Init Divergence	28.89%	56.63%	82.41%	35.76%	47.75%	56.88%
TreePO w/ More Init Divergence	27.83%	55.53%	85.34%	34.98%	49.15%	58.21%

4 Discussion

This chapter is organized around a set of research questions (RQs) that guide our investigation, with targeted ablation studies presented in the subsections that follow.

RQ1. Does tree-based sampling improve sampling efficiency relative to non-tree baselines, and under which segment and branching configurations?

Table 2 Performance Comparison Between Sequential and Tree-based Sampling with Major@16 Accuracy.

Model	Sampling	AIME	AMC	MATH	MINERVA	Olympiad Bench	Overall \uparrow	GPU Hour \downarrow
TreePO w/ Fixed Init Divergence	Sequential	28.89%	56.63%	82.41%	35.76%	47.75%	56.88%	5.78
	8x2048, $b = 2$	23.33%	57.83%	81.80%	36.76%	45.93%	56.03%	4.29 ($\downarrow 26\%$)
	8x2048, $b = 4$	23.33%	57.83%	84.00%	36.03%	48.00%	57.50%	4.82 ($\downarrow 17\%$)
	8x2048, $b = 8$	26.67%	55.42%	83.60%	36.40%	46.22%	56.60%	5.09 ($\downarrow 12\%$)
TreePO w/ More Init Divergence	Sequential	27.83%	55.53%	85.34%	34.98%	49.15%	58.21%	6.40
	8x2048, $b = 2$	21.52%	53.99%	81.89%	33.93%	44.41%	54.67%	3.65 ($\downarrow 43\%$)
	8x2048, $b = 4$	22.90%	57.24%	84.66%	35.66%	47.19%	57.26%	4.56 ($\downarrow 29\%$)
	8x2048, $b = 8$	26.21%	56.72%	85.23%	35.02%	48.81%	58.06%	5.05 ($\downarrow 22\%$)

RQ2. How does the design of the TreePO advantage (e.g., subgroup aggregation choices) shape the optimization dynamics during training?

RQ3. How do tree-sampling hyperparameters (segment length, branching factor, depth, and prefix alignment) affect stability and convergence?

RQ4. What are the trade-offs between offline efficiency and efficacy across tasks and compute budgets under different tree-sampling settings?

RQ5. How can we leverage branching budget assignment at segment-level modeling and provide more control signal in the heuristic sampling?

4.1 Sampling Efficiency Analysis

Setup. To isolate efficiency, we conduct offline efficiency analyses using three variants of the Qwen2.5: Qwen2.5-Math-7B, Qwen2.5-Math-7B-instruct, and Qwen2.5-7B-instruct. We benchmark throughput on randomly sampled prompts from a held-out pool independent of model training (used solely for efficiency measurement). Experiments were run on NVIDIA H100 80GB GPUs without any parallel, such as tensor parallel and data parallel, maintaining a GPU utilization of 60%. Single inference maximum output length is determined by the maximum segment length. Unless noted, each run processes a batch of **64 prompts** and, for tree-based sampling, **64 rollouts** per prompt. We fix a per-trajectory token budget $B = 7,000$ and vary tree depth d and max segment length L_{seg} subject to $d \times L_{\text{seg}} = B$ (segments are equal-length chunks). The non-tree baseline generates the same number of completions per prompt with identical sampling hyperparameters and the same budget B . We report Tokens per second (TokenPS; total model-processed tokens, including prefill and decode) and Trajectories per second (TrajPS; completed continuations per second), measured as wall-clock throughput.

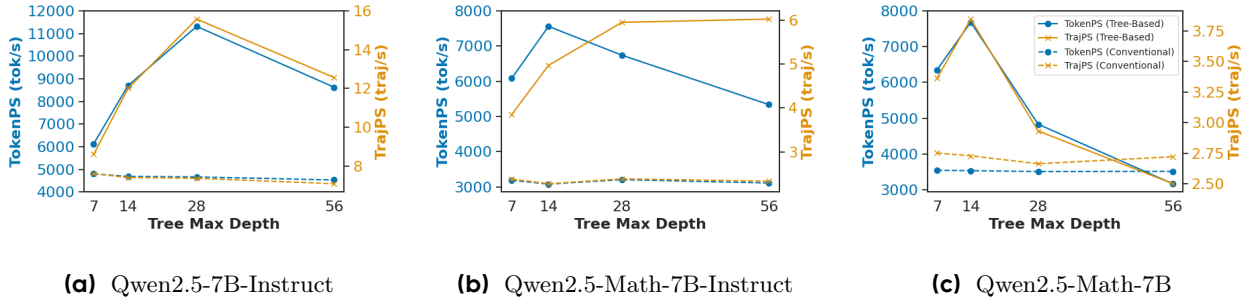


Figure 4 Performance comparison between Tree-based Sampling and Conventional Sampling across different tree depths.

Tree-based sampling improve efficiency Under the same batch size, rollout count, and budget B , tree-based sampling yields on average **+40% TrajPS** and **+30% TokenPS** across the three models (geometric mean across

configurations).

Efficiency peaks at an intermediate depth–segment trade-off. Figure 4 shows that both TokenPS and TrajPS peak at intermediate depth–segment combinations rather than grow monotonically with depth. Prefill prefers longer segments and shallower trees, which reduces repeated KV cache and attention computation; decoding prefers deeper trees with more branches and parallel rollouts, better exploiting speculative execution and batched sampling. If segments are too short, the extra recomputation offsets the gains from depth, and the peak appears where these opposing effects balance.

The optimal depth–segment configuration is model-specific. **Qwen2.5-7B-Instruct** peaks at depth 28, likely because instruction-following finds a mid-depth balance: segments are not too short (better batched prefilling and context retention) while depth still yields sufficient decoding parallelism. **Qwen2.5-Math-7B** peaks at depth 14; for compute-intensive math reasoning, longer segments at shallower depth reduce repeated KV-cache and attention recomputation, improving throughput under the fixed budget. **Qwen2.5-Math-7B-Instruct** splits—TokenPS peaks at 14, whereas TrajPS peaks at 28 and 56—consistent with deeper trees (which shorten segments under the 7k-token budget) lowering token-level throughput via recomputation and decoder overhead, but raising trajectory-level throughput by enabling more branching and parallel rollouts.

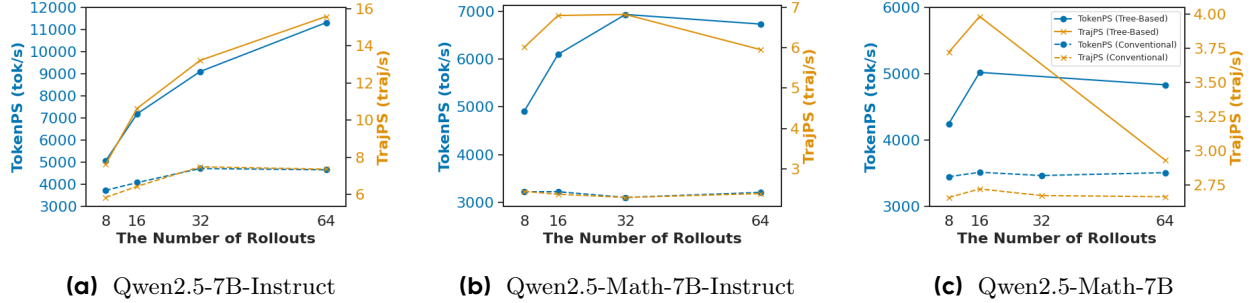


Figure 5 Performance comparison between Tree-based Sampling and Conventional Sampling across different numbers of rollouts.

Rollout scaling is model- and workload-dependent. **Qwen2.5-7B-Instruct** shows nearly linear TokenPS/TrajPS growth as rollouts increase under tree-based sampling (with query count fixed at 64 and tree depth 28), reaching roughly $2\times$ the baseline thanks to shared-prefix prefilling and more parallel decoding; by contrast, standard autoregressive decoding yields only modest gains. **Qwen2.5-Math-7B-Instruct** maintains a stable $\approx 2\times$ speedup across rollout counts, as structured, semantically aligned math trajectories sustain high cache-hit rates and efficient KV reuse, keeping batched decoding effective. **Qwen2.5-Math-7B** is non-monotonic: throughput peaks around 16 rollouts, then TokenPS/TrajPS decline as trajectory divergence reduces shared prefixes, KV-cache fragmentation and management overhead grow, memory pressure rises, and batching efficiency degrades; the lack of instruction tuning further loosens output structure. Overall, more rollouts can boost parallelism and cache reuse but also amplify memory and synchronization costs when trajectories diverge, implying a model- and workload-dependent optimum.

4.2 Analysis on the TreePO Advantage Estimation

Setup. (1)–(3) use depth \times segment 14×512 with a 512-token fallback; (4) uses 7×1024 rollout but still a 512-token fallback, inducing prefix misalignment. Figure 7 reports MATH/AIME accuracy, entropy loss, and response length, and the “Subgroup-size Weighted” curve serves as a reference baseline for comparison across variants.

$$\hat{A}_{i,t} = \frac{\sum_{j=1}^J |G_j| \cdot \hat{A}_{i,t,j}}{\text{std}(\{\hat{A}_{i,t,j}\}^{J-1}) \sum_{j=1}^J |G_j|}, \quad (6)$$

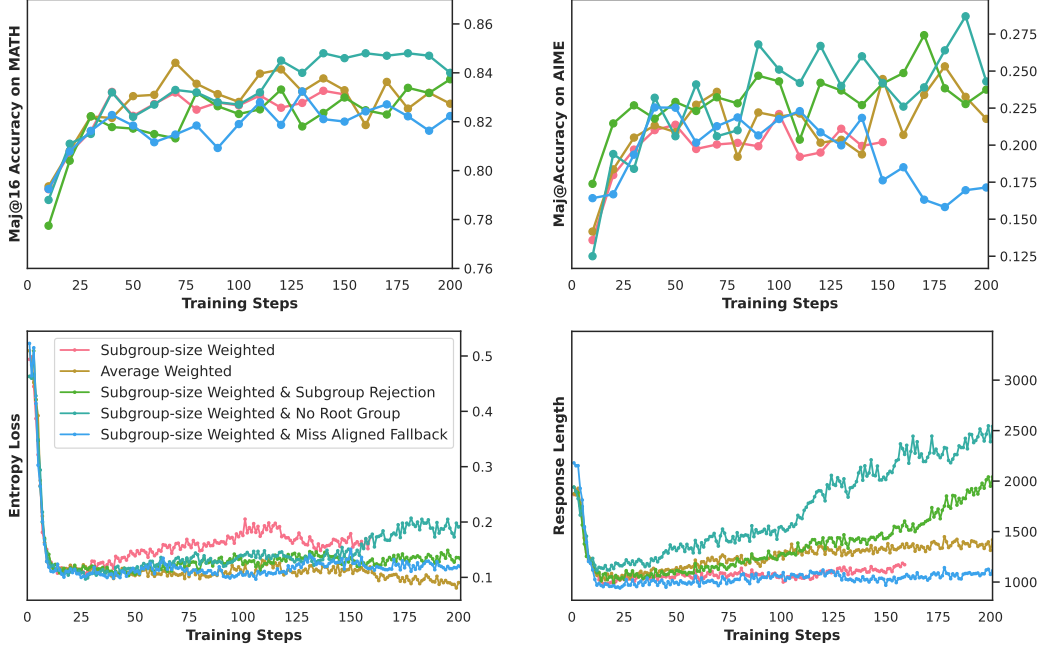


Figure 6 Study on the Terms in TreePO Advantage. These group of experiments sets the depth \times segment as 7×1024 and uses the subgroup size weighted aggregation advantage as the baseline.

Simple averaging across subgroups is better than subgroup-size weighting. We further propose a modified estimation function Equation 6 from Equation 5 to validate whether a simple modification on the aggregation, based on subgroup size, is more appropriate for modeling the advantages. Averaging tracks higher accuracy on both MATH and AIME, with lower and more stable entropy and no unnecessary growth in response length. Size-weighting over-emphasizes large/easy subgroups and down-weights informative small/hard ones, whereas simple averaging preserves a balanced signal; we therefore adopt averaging in the method and keep size-weighting only for discussion.

$$\begin{aligned}
 \hat{A}_{i,t} &= \frac{\sum_{j=1}^J |G_j| \cdot \hat{A}_{i,t,j}}{\text{std}(\{\hat{A}_{i,t,j}\}^{J-1}) \sum_{j=1}^J |G_j|}, \\
 \hat{A}_{i,t,j} &= R_i - \text{mean}(\{R_{i,j}\}^{G_j}), \\
 \text{s.t. } &\text{std}(\{R_{i,j}\}^{G_j}) \neq 0
 \end{aligned} \tag{7}$$

Naïve rejection hurts performance Demonstrated in Equation 7, we also test the effectiveness of dynamic rejection sampling at the subgroup level as additional the subgroup hierarchy information is provided. Such a DAPO-style subgroup rejection that discards all-positive or all-negative subgroups biases the feedback signal and weakens learning: accuracy lags, entropy is less favorable, and generations become longer. These “extreme” subgroups actually calibrate margins; removing them strips away high-signal cases, so we avoid subgroup-level rejection.

Removing the root-group advantage does not degrade performance Using only the aggregated subgroup advantages while dropping the root-group term yields comparable curves, indicating that integrated subgroup signals can approximate the full-group optimization signal. This redundancy suggests the root term is not strictly necessary and is a promising direction for further analysis of credit assignment.

Misaligned fallback degrades accuracy and inflates response length. With 7×1024 segments but a 512-token random fallback, trajectories can share an abstract tree prefix while being token-misaligned. Figure 6 shows a drop in AIME accuracy and a sharp rise in response length for the misaligned variant, highlighting that token-aligned segments are important for stable optimization and precise stopping behavior.

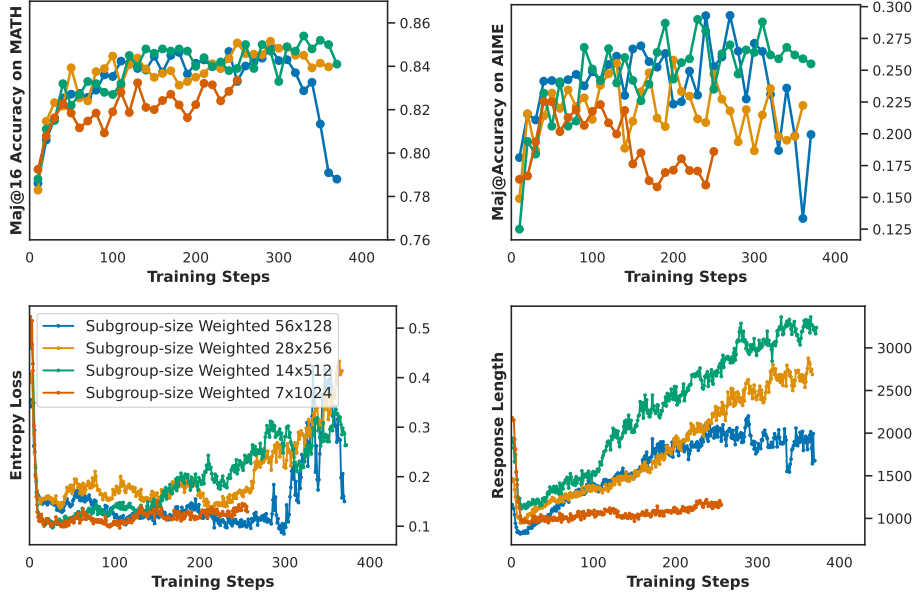


Figure 7 Study on the Online Depth-Segment under Setting the Group Size Weighted TreePO Advantage.

4.3 Analysis of Segment Budget

Setup We adopt the same subgroup-size weighted setting as in §4.2 to explore a the combination parameter of $d \times L_{seg}$. Here we set the $L_{seg} \in 128, 256, 512, 1024$ and adjust the maximum depth to fit the response length limit 7×1024 accordingly. The training curves are shown in Figure 7

Depth-segment trade-off, 14×512 is the sweet spot while 7×1024 underperforms. Under the group-size weighted advantage, 14×512 attains the highest final MATH/AIME accuracy; 56×128 and 28×256 are close, whereas 7×1024 lags—especially on AIME—indicating that deeper trees with moderate segments provide stronger credit assignment than shallow rollouts with very long segments.

Accuracy-length coupling, Better accuracy comes with longer generations. The best-performing 14×512 also drives the largest growth in response length (and higher entropy), while 7×1024 keeps outputs shorter but sacrifices accuracy. This suggests online TreePO benefits from more exploratory, longer reasoning traces; shorter traces trade accuracy for brevity.

4.4 Analysis of Probability-based Branching Assignment

Setup With segment-level control, TreePO sampling provide a more feasible environment to study the training dynamics of the LLM bounding to the decoding progress. Stemming from the TreePO “w/ More Init Divergence” setting, we conduct a set of experiments to control modify the branching assignment at a given depth d . Under this setting, the total branching budget 2^d is assign among the active paths conditioned on the log probabilities of their last segment, return from the inference engine. To prevent a sudden truncation of the search, the probabilities as passed through a softmax function with temperature set as 2.0, and all the active search paths are guaranteed with at least one branching budget. The comparison between different branching budget controls are shown in Figure 8, where the "Low Prob Encourage" suggests the paths with lower probability get more branching budget, and vice versa for "High Prob Encourage". We also try with a

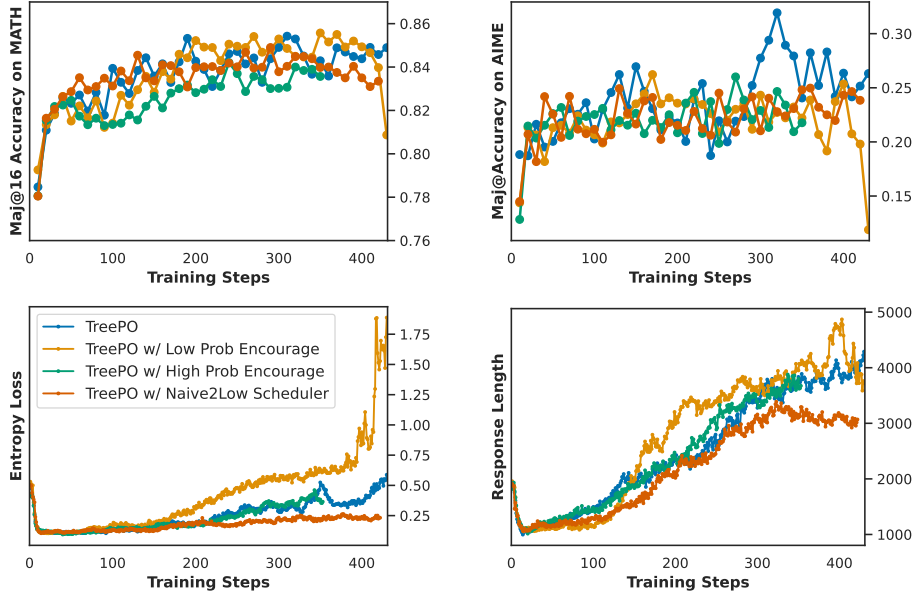


Figure 8 Study on a Probability-based Heuristic Tree Branching Budget Assignment.

more sophisticated "Low Prob Encourage" setting that the temperature of the softmax function is schedule from 5.0 to 1.0 across the training.

Monotonous Pattern Could Be Harmful. As shown in Figure 8, both static heuristic controls—"Low Prob Encourage" and "High Prob Encourage"—underperform the baseline and the scheduled variant. The "Low Prob Encourage" strategy, in particular, consistently yields the lowest accuracy on both benchmarks. This performance degradation is strongly correlated with a significant increase in response length and entropy loss, suggesting that forcing the model to explore low-probability states leads to less efficient and coherent search trajectories across the whole training. Conversely, the "High Prob Encourage" setting, while performing better, results in the lowest entropy and shortest responses, indicating a potentially overly greedy search that may prune promising, less obvious paths too early. Even when the scheduled "Low Prob Encourage" setting ensure a similar branching assignment scheme at the beginning of the training, it still does not provide any advantages.

Such Branching Control Does Not Show Significant Benefit Even with Higher Entropy. The most striking observation from our study is the disconnect between search diversity and task performance. The "Low Prob Encourage" setting was explicitly designed to increase exploration by allocating more resources to less likely search paths. This is reflected in its entropy loss, which is substantially higher than all other methods throughout training. However, this artificially inflated entropy does not translate into better results. Instead, it correlates with the worst performance on both benchmarks. This suggests that merely forcing the model to explore more diverse paths is not beneficial; the exploration must be meaningful. In this case, allocating budget to low-probability segments appears to push the model into irrelevant or erroneous reasoning paths, leading to longer, less effective solutions, as evidenced by the Response Length plot. The baseline maintains a more moderate entropy level, which proves more effective for complex reasoning tasks, indicating it strikes a better intrinsic balance between exploration and exploitation.

4.5 Compute Scaling for Tree Sampling

Setup As the sampling mechanism has been modified, the scaling curves along compute do not necessarily follow the same trend as sequential sampling. To investigate this, we analyze the test-time compute scaling of TreePO by evaluating model performance under various computational budgets, as shown in Figure 9. Note

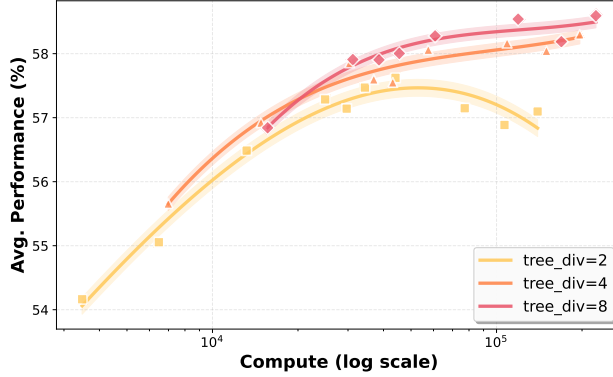


Figure 9 Test-time Compute Scaling of TreePO Sampling on the Aggregated Benchmark. The x-axis represents the compute budget on a log scale, while the y-axis shows average performance. Each curve corresponds to a different tree divergence factor $d = 2, 4, 8$. The results illustrate that a larger divergence factor can achieve higher peak performance at the cost of a larger compute budget, revealing a trade-off between the exploration strategy and computational cost that distinguishes it from the scaling behavior of conventional sequential sampling.

that the number rollout starts from d when calculating the compute curves.

Distinct Rules The experiment varies the tree divergence factor (`tree_div` in Figure 9), which controls the number of branches generated at each divergence point, to observe its effect on the performance-compute trade-off. The results show that all configurations follow a predictable scaling pattern: performance improves with increased compute before eventually reaching a point of diminishing returns, which is consistent with established inference scaling observations. However, the key distinction from conventional sequential sampling becomes apparent when comparing the different divergence strategies. In sequential sampling, scaling compute is typically achieved by increasing the number of independent samples (N), which generally traces a single performance-compute curve. In contrast, TreePO generates a family of scaling curves, where each curve corresponds to a different internal search strategy controlled by d . At lower compute budgets, a smaller divergence factor ($d = 2$) is more efficient, achieving better performance for less cost. As the compute budget increases, wider search strategies ($d = 4$ and $d = 8$) become superior, with $d = 8$ ultimately reaching the highest peak performance. This demonstrates that the optimal TreePO sampling strategy is dependent on the available compute budget, allowing for more flexible "compute-optimal inference". Rather than simply scaling the number of samples, one can select the optimal tree structure to maximize performance for a given computational constraint.

5 Related Work

Efficient Sampling. Recent work on efficient sampling for RL and inference concentrates on making the rollout loop lighter by batching many completions together, re-using the prompt KV-cache and hiding latency behind parallel decoding; typical examples are [25] [26], [27], and [28], which all treat a prompt as a mini-batch and schedule tokens in groups so that GPUs stay busy. [10] keeps this idea but breaks a large group into small "micro" groups, runs them with continuous inter-leaving, and adds a length-aware scheduler; this saves memory and keeps the buffer fixed, yet it does not look at the partial trajectories while they are generated, introduces extra scheduling logic, and leaves the advantage estimator untouched [10]. [9] improves wall-time by cutting every sampled chain after a short window and back-propagating early; the price is that long-range information is lost and credit assignment becomes harder.

Segment-level Modeling. A second line of research studies reinforcement learning with tree search. Recent systems such as [29], [26], [11] and [12] build explicit trees and use them to explore many reasoning branches in one rollout, giving denser feedback than plain chain sampling [11, 12, 26, 29]. TreeRL couples on-policy tree expansion with process-level rewards, but its trees stay shallow and the algorithm rolls one full answer

to compute log-probabilities before it can branch again, which doubles the running time [11]. [30] adopts an unconstrained tree and a “progress advantage” similar to Monte-Carlo returns; while this brings a simple tree-based update, it lacks depth control and is not validated against a frozen base policy. Similarly, ARPO [31] apply a segment-level entropy-guided divergence strategy based on the finished tool call trajectories, analogical to the FR3E algorithm [32] in math domain.

6 Conclusion

In this work, we introduced TreePO, a reinforcement learning framework designed to address the computational inefficiency and exploration instability in training large language models for complex reasoning. By reformulating on-policy rollouts as a segment-based tree search and using a hierarchical advantage estimator, TreePO significantly reduces reasoning computational costs while improving training stability and maintaining strong performance. The efficiency and structural modeling of TreePO open promising avenues for scaling reinforcement learning to more complex, long-horizon tasks such as multi-turn dialogue, tool use, and multi-agent systems.

Authorship

Yizhi Li ^{1,3*}, Qingshui Gu ^{1*}, Zhoufutu Wen ^{2*}, Ziniu Li ², Tianshun Xing ¹, Shuyue Guo ¹, Tianyu Zheng ¹, Xin Zhou ², Xingwei Qu ^{1,2,3}, Wangchunshu Zhou ¹, Zheng Zhang ², Wei Shen ², Qian Liu ¹, Chenghua Lin ^{3◇}, Jian Yang ^{1◇}, Ge Zhang ^{1, 2◇}, Wenhao Huang ²

¹M-A-P ²Bytedance Seed ³The University of Manchester

* Equal contribution

◇ Corresponding Author

References

- [1] 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.
- [2] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. [arXiv preprint arXiv:2402.03300](#), 2024.
- [3] Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. [arXiv preprint arXiv:2503.14476](#), 2025.
- [4] Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. In [International Conference on Machine Learning](#), pages 29128–29163. PMLR, 2024.
- [5] 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.
- [6] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In [European conference on machine learning](#), pages 282–293. Springer, 2006.
- [7] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharmashan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. [arXiv preprint arXiv:1712.01815](#), 2017.
- [8] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. [Artificial Intelligence Review](#), 56(3):2497–2562, 2023.
- [9] Tiantian Fan, Lingjun Liu, Yu Yue, Jiaze Chen, Chengyi Wang, Qiyang Yu, Chi Zhang, Zhiqi Lin, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Bole Ma, Mofan Zhang, Gaohong Liu, Ru Zhang, Haotian Zhou, Cong Xie, Ruidong Zhu, Zhi Zhang, Xin Liu, Mingxuan Wang, Lin Yan, and Yonghui Wu. Truncated proximal policy optimization, 2025.
- [10] Liangyu Wang, Huanyi Xie, Xinhai Wang, Tianjin Huang, Mengdi Li, and Di Wang. Infinite sampling: Efficient and stable grouped rl training for large language models, 2025.
- [11] Zhenyu Hou, Ziniu Hu, Yujia Li, Rui Lu, Jie Tang, and Yuxiao Dong. Treerl: Llm reinforcement learning with on-policy tree search. [arXiv preprint arXiv:2506.11902](#), 2025.
- [12] Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in rl for large language models, 2025.
- [13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. [Advances in neural information processing systems](#), 35:24824–24837, 2022.
- [14] Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: An efficient rlhf algorithm with robustness to both prompt and reward models. [arXiv preprint arXiv:2501.03262](#), 2025.
- [15] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.
- [16] 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. [NeurIPS](#), 2021.
- [17] Weihao Zeng, Yuzhen Huang, Wei Liu, Keqing He, Qian Liu, Zejun Ma, and Junxian He. 7b model and 8k examples: Emerging reasoning with reinforcement learning is both effective and efficient. <https://hkust-nlp.notion.site/simplerl-reason>, 2025. Notion Blog.

- [18] 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://pretty-radio-b75.notion.site/DeepScaler-Surpassing-01-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>, 2025. Notion Blog.
- [19] MAA. Aime 2024 problems, 2024. Accessed: 2025-05-11.
- [20] MAA. Amc 2023 problems, 2023. Accessed: 2025-05-11.
- [21] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.
- [22] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.
- [23] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.
- [24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- [25] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models, 2025.
- [26] Coleman Hooper, Sehoon Kim, Suhong Moon, Kerem Dilmen, Monishwaran Maheswaran, Nicholas Lee, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. Ets: Efficient tree search for inference-time scaling, 2025.
- [27] Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. Batchllm: Optimizing large batched llm inference with global prefix sharing and throughput-oriented token batching, 2025.
- [28] Ozgur Guldogan, Jackson Kunde, Kangwook Lee, and Ramtin Pedarsani. Multi-bin batching for increasing llm inference throughput, 2024.
- [29] Kaiwen Wang, Jin Peng Zhou, Jonathan Chang, Zhaolin Gao, Nathan Kallus, Kianté Brantley, and Wen Sun. Value-guided search for efficient chain-of-thought reasoning, 2025.
- [30] Xingzhou Lou, Junge Zhang, Jian Xie, Lifeng Liu, Dong Yan, and Kaiqi Huang. Spo: Multi-dimensional preference sequential alignment with implicit reward modeling, 2024.
- [31] Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization, 2025.
- [32] Tianyu Zheng, Tianshun Xing, Qingshui Gu, Taoran Liang, Xingwei Qu, Xin Zhou, Yizhi Li, Zhoufutu Wen, Chenghua Lin, Wenhao Huang, Qian Liu, Ge Zhang, and Zejun Ma. First return, entropy-eliciting explore, 2025.