TreeRL: LLM Reinforcement Learning with On-Policy Tree Search

Zhenyu Hou[*,1] **Ziniu Hu**[*,2] **Yujiang Li**[*,1] **Rui Lu**[*,1] **Jie Tang**[1] **Yuxiao Dong**[1]
[1]Tsinghua University [2]California Institute of Technology

Abstract

Reinforcement learning (RL) with tree search has demonstrated superior performance in traditional reasoning tasks. Compared to conventional independent chain sampling strategies with outcome supervision, tree search enables better exploration of the reasoning space and provides dense, on-policy process rewards during RL training but remains under-explored in On-Policy LLM RL. We propose TreeRL, a reinforcement learning framework that directly incorporates on-policy tree search for RL training. Our approach includes intermediate supervision and eliminates the need for separate reward model training. Existing approaches typically train a separate process reward model, which can suffer from distribution mismatch and reward hacking. We also introduce a cost-effective tree search approach that achieves higher search efficiency under the same generation token budget by strategically branching from high-uncertainty intermediate steps rather than using random branching. Experiments on challenging math and code reasoning benchmarks demonstrate that TreeRL achieves superior performance compared to traditional ChainRL, highlighting the potential of tree search for LLM. TreeRL is open-sourced at `https://github.com/THUDM/TreeRL`.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across diverse complex reasoning tasks (Achiam et al., 2023; Team et al., 2023; Dubey et al., 2024), including mathematics (Shao et al., 2024b), programming (Lozhkov et al., 2024; Zhu et al., 2024), and autonomous agents (Zhou et al., 2024). Reinforcement learning (RL) has emerged as a powerful approach to significantly improve the reasoning abilities of LLMs by optimizing the policy through reward

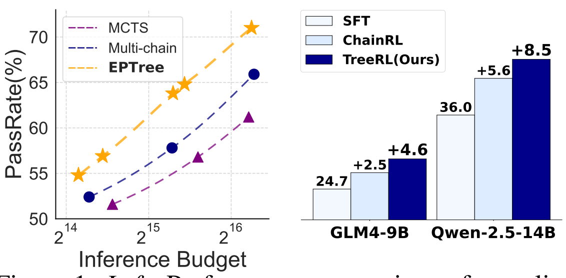


Figure 1: *Left*: Performance comparison of sampling strategies. EPTree consistently outperforms i.i.d multi-chain sampling and MCTS under different inference budgets. *Right*: TreeRL powered with EPTree demonstrates better performance than ChainRL with i.i.d multi-chain sampling.

feedback (OpenAI, 2024; Guo et al., 2025; Hou et al., 2025; Shao et al., 2024b).

Current RL methods for LLM training generally independently sample multiple trajectories (Shao et al., 2024b; Wang et al., 2024b; Touvron et al., 2023) and obtain reward signals based on the final answers. However, tree search, which has demonstrated significant success in other domains like AlphaZero (Silver et al., 2017), remains underdeveloped in reinforcement learning for LLM reasoning. Existing efforts have mainly focused on using tree search to enhance inference-time performance alongside an external reward (Zhang et al., 2024a; Chen et al., 2024a), or to produce data for offline training (Chen et al., 2024a; Xie et al., 2024; Zhang et al., 2024a) (e.g., finetuning or DPO (Rafailov et al., 2023)), as illustrated in Figure 2. But Guo et al. (2025) also demonstrates the limitation of distribution shift and reward hacking in offline tree search compared to online RL training. Up to now, the potential of on-policy RL training incorporating tree search to improve LLM reasoning remains largely unexplored.

The challenges are two-fold. First, classical Monte Carlo Tree Search (MCTS) (Browne et al., 2012) can be less effective nd efficient than inde-

[*]Equal contribution; order is alphabetical.

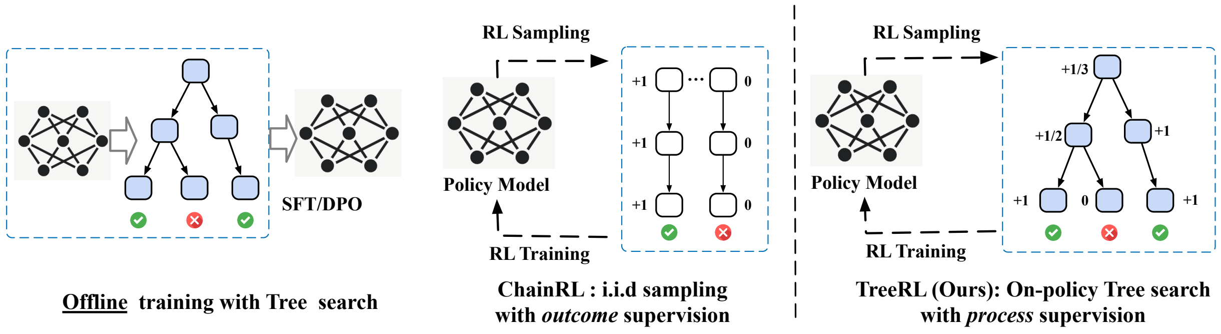

SFT/DPO
Offline training with Tree search
RL Sampling
Policy Model
RL Training
+1
0
ChainRL : i.i.d sampling with *outcome* supervision
RL Sampling
Policy Model
RL Training
+1/3
+1/2
+1
0
TreeRL (Ours): On-policy Tree search with *process* supervision

Figure 2: Illustration of offline training with tree search (Left), traditional ChainRL with online i.i.d multi-response sampling (Middle), and TreeRL with tree search (Right).

pendently sampling multiple responses. MCTS achieves a lower PassRate performance under the same inference cost, as shown in Figure 1. The step-by-step generation requires numerous iterations and is not friendly to modern LLM inference engines. Second, though tree search could provide fine-grained process supervision, the derived offline process reward model almost contributes to no performance improvement in RL training (Guo et al., 2025).

To address this gap, we propose TreeRL, a reinforcement learning (RL) approach for LLMs employed with tree search. Under the same inference cost as independent multiple sampling, our method generates more diverse and effective responses and also provides on-policy process supervision signals to further boost RL performance.

First, we introduce an efficient and effective tree search strategy EPTree. Unlike MCTS which breaks answers into smaller parts to allow the model to explore step-by-step, we obtain a new response by forking branches from the most uncertain intermediate tokens in the existing tree based on entropy and continuing the generation until the final answer. Thus EPTree requires fewer tokens but encourages effective exploration for the diverse answers. And it typically requires only around two iterations to form the generation trees.

One step further, we leverage the tree search for reinforcement learning with process supervision. Each step in the trees is assigned a credit based on advantage, i.e., how much that step improves the likelihood of reaching a correct solution compared to other steps. The process signal for a given reasoning step is calculated as a weighted sum of two components: 1) global advantage, which reflects the step's potential over the overall correctness rate for the question, and 2) local advantage, which quantifies the improvement the step provides compared to its parent step in the tree. Since these advantage signals are derived directly from the on-policy generated trees, they are inherently resistant to reward hacking (Skalse et al., 2022) and do not rely on any additional reward models.

We evaluate TreeRL on challenging college-level and competition-level math and code reasoning benchmarks based on Qwen (Qwen, 2024) and GLM (GLM et al., 2024). Experiments show that TreeRL achieves superior performance and demonstrates advantages over traditional independent multi-chain sampling. The gain benefits from both EPTree with promising PassRate performance and process supervision. These results highlight the potential of RL with tree search to advance complex reasoning capabilities for LLM. The implementation of TreeRL is available at `https://github.com/THUDM/TreeRL`.

2 Preliminary

To align the fine-tuned model π_θ with feedback signals, Ouyang et al. (2022) proposes to apply reinforcement learning (RL) to enable LLM to learn from self-exploration. Reinforcement learning maximizes a reward signal, e.g., human preference or final answer correctness. The typical RL for the LLM process works as follows: for a given prompt $\boldsymbol{x}$, the policy model π_θ generates K possible responses, denoted as $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K)$. The reward function $r(\boldsymbol{x}, \boldsymbol{y}_i)$ then assigns a scalar reward to each pair $(\boldsymbol{x}, \boldsymbol{y}_i)$. Afterward, the policy model π_θ is updated using reinforcement learning to optimize the following objective:

$$\mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}, \boldsymbol{y}\sim\pi_\theta} \frac{1}{K}\sum_{i}^{K} A(\boldsymbol{x}, \boldsymbol{y}_i) \log \pi_\theta(\boldsymbol{y}_i|\boldsymbol{x}) \quad (1)$$

where $A(\cdot)$ is the advantage function and usually defined as $A(\boldsymbol{x}, \boldsymbol{y}_i) = \beta(r(\boldsymbol{x}, \boldsymbol{y}_i) - b)$, where b

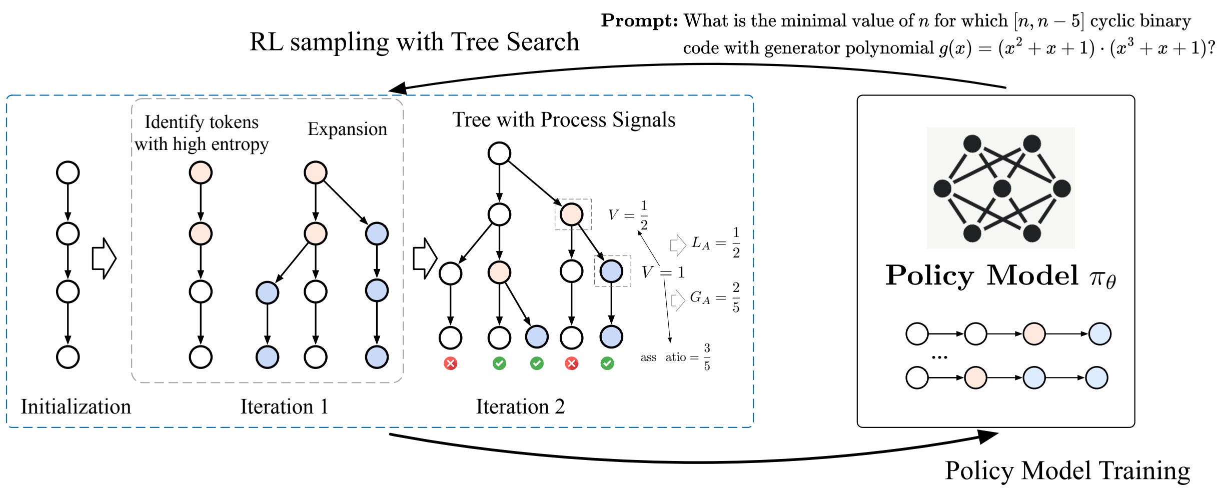

Prompt: What is the minimal value of n for which $[n, n-5]$ cyclic binary code with generator polynomial $g(x) = (x^2+x+1)\cdot(x^3+x+1)$?
RL sampling with Tree Search
Identify tokens with high entropy
Expansion
Tree with Process Signals
$V = \frac{1}{2}$
$L_A = \frac{1}{2}$
$V = 1$
$G_A = \frac{2}{5}$
ass atio = $\frac{3}{5}$
Initialization
Iteration 1
Iteration 2
Policy Model π_θ
Policy Model Training

Figure 3: Illustration of TreeRL. In each iteration, TreeRL first performs a tree search using EPTree, progressively expanding branches from the top-N most uncertain tokens. The resulting tree, along with the process supervision signals derived from each step, is then fed into reinforcement learning to update the policy model.

is the baseline (Mei et al., 2022; Chung et al., 2021) and varies in different methods. For example, $A(\boldsymbol{y}_i) = \frac{r(\boldsymbol{y}_i) - \text{mean}\{r(\boldsymbol{y}_i)\}}{\text{std}\{r(\boldsymbol{y}_i)\}}$ in GRPO (Shao et al., 2024b) and $A(\boldsymbol{y}_i) = r(\boldsymbol{y}_i) - \frac{1}{K-1}\sum_{j\neq i}^{K} r(\boldsymbol{y}_i)$ in RLOO (Ahmadian et al., 2024).

3 TreeRL: Reinforcement Learning with Tree Search

In this section, we present TreeRL to improve LLM reasoning with tree search. Figure 3 shows the overview of TreeRL. We first present an efficient and effective tree search algorithm EPTree, which guides tree search by token-level uncertainty instead of traditional Monte Carlo Tree Search (MCTS) (Browne et al., 2012). Then, we show how to integrate the tree search into reinforcement learning with process supervision to improve reasoning capability further.

3.1 Entropy-Guided Tree Search

We aim to optimize the tree-search algorithm for RL training and thus emphasize the PassRate metric, which evaluates the algorithm's ability to generate diverse yet correct answers under a given inference budget. Furthermore, the tree-search algorithm must be efficient and highly parallelizable. In contrast, traditional MCTS requires numerous iterative generations, making it less efficient in current LLM inference engines like VLLM (Kwon et al., 2023).

We propose an entropy-guided tree search algorithm, EPTree. The core idea is to iteratively expand the search tree by forking new branches (nodes) from the top-N most uncertain tokens (as measured by entropy) across existing trees. This encourages exploration in regions of high model uncertainty, leading to improved performance. Critically, EPTree can generate multiple trees in parallel, and the expansion process requires only around 2 iterations to build a diverse and informative tree, making it highly efficient. The algorithm runs the following steps.

Initialization. To generate M trees in parallel, we first construct M chains by generating M responses for a given prompt $\boldsymbol{x}$ as initialization of $\mathcal{T}_i$ for further expansion:

$$Y^{(i)} = \{\boldsymbol{y}_i \sim \pi_\theta(\cdot \mid \boldsymbol{x})\}, \text{ for } i = 1, 2, \ldots, M$$

where π_θ is the policy model and $\boldsymbol{x}$ is the prompt.

Forking token selection. Next, we aim to expand the trees by forking new branches from the existing trees. We propose forking from the tokens with the highest uncertainty, as these tokens provide the most informative signals for the model and encourage exploration. We use cross-entropy as a measure of uncertainty, which quantifies the uncertainty in the policy model π_θ when predicting a given token. To promote expansion, the top-N tokens with the highest entropy values are selected across the whole tree $\mathcal{T}_i$. Specifically, the entropy of each token v in the tree $\mathcal{T}_i$ is calculated as follows:

$$B_i = \text{Top-}N_{H(\cdot|\boldsymbol{x})}\left\{(t, H\left(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}\right)) \mid t \in \mathcal{T}_i\right\}$$

where $H(\boldsymbol{y}_t) = -\log \pi_\theta(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})$ denotes the entropy of token $\boldsymbol{y}_t$. Additionally, we mask tokens

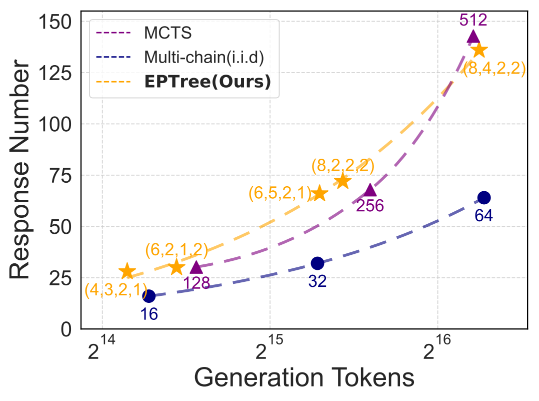


Figure 4: Generation diversity comparison of EPTree, MCTS, and i.i.d. multi-chain sampling. Both EPTree and MCTS produce approximately 2× different responses compared to i.i.d. multi-chain sampling.

near the end of sequences, as the model is expected to explore different reasoning paths rather than simply revisiting previous answers.

Expansion. Given the selected tokens for each tree, we continue the generation process from these tokens to form new branches. For each forking point t, with the prefix $\boldsymbol{y}_t$ and prompt $\boldsymbol{x}$, we generate T different candidate responses until completion:

$$Y_{\text{new}}^{(i)} \sim \{\pi_\theta\left(\cdot \mid \boldsymbol{x}, \boldsymbol{y}_{<t}\right),\ \text{for}\ (t, \cdot) \in B_i\}^T$$

where $\boldsymbol{y}_{<t}$ denotes the prefix response before token t. This results in $M \times N \times T$ tree nodes in total($N \times T$ for each tree $\mathcal{T}_i$), each corresponding to a new response. The tree structure $\mathcal{T}_i$ is updated to include these new nodes:

$$\mathcal{T}_i \leftarrow \mathcal{T}_i \cup Y_{\text{new}}^{(i)}$$

After initialization, the forking and expansion process is repeated for L iterations, leading to $M \times (T \times N \times L + 1)$ leaves (responses). We denote this entropy-guided tree search as an (M, N, L, T)-tree, where M is the number of parallel trees, N is the number of forked points per iteration, L is the number of iterations, and T is the branching factor at each forking point.

In comparison to independent multi-chain sampling, the EPTree is capable of producing a larger number of diverse responses under the same inference cost, as illustrated in Figure 4. This offers the potential to enhance RL performance by learning from more varied responses.

3.2 Reinforcement Learning with EPTree

In this part, we show how to integrate EPTree into RL training. Beyond the superior potential to find correct trajectories, hierarchical tree structures also provide more fine-grained process supervision for intermediate steps for RL training.

3.2.1 Process Supervision from Tree Search

At each step, we estimate the value using Monte Carlo methods. Specifically, for a step s_n (corresponding to a node in the tree), let $L(s_n)$ denote the set of all leaf nodes that are descendants of node s_n (including node s_n itself if it is a leaf). The value $V(s_n)$ of node s_n is computed as the ratio of correct leaf nodes among its descendants:

$$V(s_n) = \frac{1}{|L(s_n)|} \sum_{l \in L(s_n)} \mathbf{1}(l \text{ is correct})$$

This value reflects the potential of the node s_n to lead to correct answers. Based on value, process supervision for each step is defined using advantages, i.e., how much a step is better than other steps, which include both *global* and *local advantages*.

The *global advantage* of a step s_n represents its potential to lead to a correct outcome compared to the overall correctness ratio of all samples. Without loss of generality, assume that there exists a virtual root node for all subtrees, and the value of the root node $V(root)$ represents the average correctness of all generated responses. The global advantage is then computed as:

$$G_A(s_n) = V(s_n) - V(\text{root}) \tag{2}$$

Essentially, the global advantage of s_n is equivalent to the normalized value, which can be obtained by first normalizing the reward across all leaf nodes—subtracting the average reward—and then calculating the average rewards for $L(s_n)$.

The *local advantage* of a node s_n quantifies the improvement the step s_n provides compared to its parent step $p(s_n)$ and is defined as:

$$L_A(s_n) = V(s_n) - V(p(s_n)) \tag{3}$$

where $V(p(s_n))$ denotes the value of the node $p(s_n)$. $L_A(s_n) > 0$ indicates that step s_n is more likely to lead to the correct result than its parent node, suggesting that this step should be encouraged, and vice versa.

To compute the final reward for each step, we combine the global and local advantages as follows:

$$R(s_n) = G_A(s_n) + L_A(s_n) \tag{4}$$

The definition could be viewed as a special case of Generalized Advantage Estimation

Algorithm 1 TreeRL

```
Input: Prompt $\boldsymbol{x}$, Policy $\pi_\theta$, Number of Trees $M$, Forking Points $N$, Iterations $L$, Branches $T$
Output: Optimized Policy $\pi_\theta^*$
for $i = 1$ to $M$ do                                              ▷ Create $M$ trees with $M(1+NLT)$ leaves
    $Y^{(i)} \leftarrow \{\boldsymbol{y}_i \sim \pi_\theta(\cdot \mid \boldsymbol{x})\}, \mathcal{T}_i \leftarrow \{Y^{(i)}\}$              ▷ Initialization
    for $l = 1$ to $L$ do                                          ▷ Iterative Expansion
        $H(\boldsymbol{y}_t) \leftarrow -\log \pi_\theta(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t}), \forall\, t \in \mathcal{T}_i$
        $B_{i,l} \leftarrow \text{Top-}N_{H(\cdot|\boldsymbol{x})} \{(t, H(\boldsymbol{y}_t \mid \boldsymbol{x}, \boldsymbol{y}_{<t})) \mid t \in \mathcal{T}_i\}$
        for each selected forking point $(t, \cdot) \in B_{i,l}$ do
            $Y_{\text{new}}^{(i,l)} \sim \{\pi_\theta(\cdot \mid \boldsymbol{x}, \boldsymbol{y}_{<t}),\ \text{for}\ (t,\cdot) \in B_{i,l}\}^T$
            $\mathcal{T}_i \leftarrow \mathcal{T}_i \cup Y_{\text{new}}^{(i,l)}, j \in \{1, \cdots, T\}$
for each step $s_n$ in $\mathcal{T}_i$ do                                     ▷ Process Reward Calculation
    $V(s_n) \leftarrow \frac{1}{|L(s_n)|} \sum_{l \in L(s_n)} \mathbf{1}(l \text{ is correct})$
    $R(s_n) \leftarrow \underbrace{|L(s_n)|^{-1/2}}_{\text{Re-weight Factor}} \cdot [\underbrace{V(s_n) - V(\text{root})}_{\text{Global Advantage}} + \underbrace{V(s_n) - V(p(s_n))}_{\text{Local Advantage}}]$
Update $\pi_\theta$ using RL with process reward by Policy Gradient
```

(GAE) (Schulman et al., 2015). The general form of GAE in LLM is defined as:

$$A(s_n \rightarrow s_{n+t}) = \gamma^t V(s_{n+t}) - V(s_n) \qquad (5)$$

where t represents any integer time step, and γ is typically set to 1 in most cases. Thus, the local advantage defined in Eq 3 corresponds to the case where $t = 1$, while the global advantage corresponds to the case where $n = 0$. Inspired by the GAE, the process supervision signal can be defined as a more generalized format by considering not only the root and direct parent but also all of its ancestor nodes in the trajectory:

$$R_{GAE}(s_n) = \sum_{j \in P(s_n)} \lambda_j \cdot [V(s_n) - V(s_j)]$$

where $P(s_n)$ represents the set of ancestor nodes of s_n and λ_j denotes the weight of each step. In this work, we focus on a special format that only considers the direct parent and the root node.

3.2.2 Training with Process Supervison

At each iteration, we first utilize EPTree described in Section 3.1 to generate M trees $\mathcal{T} = \{\mathcal{T}_i\}_i^M$ for the prompt, where each leaf node in $\mathcal{T}$ together with its all prefix corresponds to a complete sequence. For each step in the tree, the process supervision signal is assigned based on Equation (4) to reflect its importance.

The sequences $\{S_1, S_2, \ldots, S_{(M \times N \times L \times T + M)}\}$ extracted from the trees are used for RL training. As all non-leaf steps appear in multiple sequences and will be repeatedly computed in optimization, we downweight the reward of these steps to prevent overfitting. The reward for each non-leaf step is modified by dividing the square root of the number of leaf nodes in its subtree: $R(s_n) = R(s_n)/\sqrt{|L(s_n)|}$. This adjustment leads to improved performance in our experiments. The overall pipeline of TreeRL is illustrated in Alg 1.

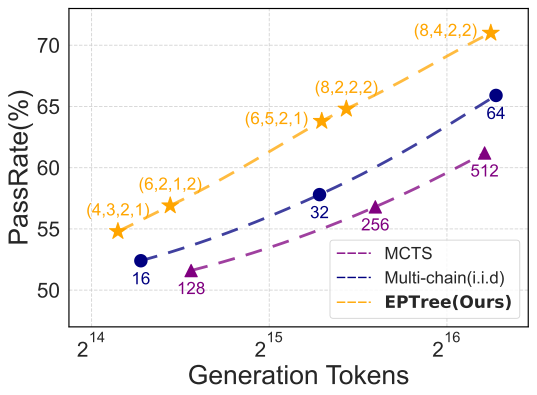


Figure 5: Search performance of EPTree, MCTS, and multi-chain sampling on Omni-MATH-500 using Qwen-2.5-14B-SFT. EPTree consistently outperforms all baselines under the different inference costs.

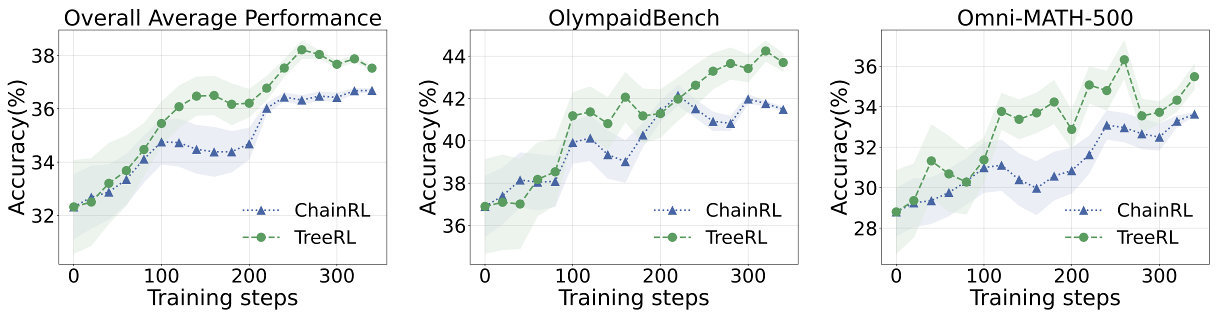
4 Experiment

4.1 Setup

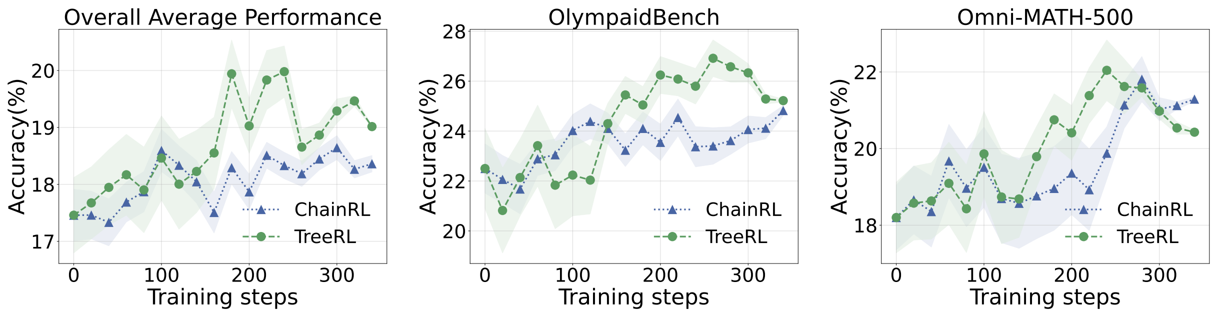
Training Details. We train SFT models based on Qwen-2.5-14B (Qwen, 2024) and GLM4-9B (GLM et al., 2024) as initialization for RL training and finetune them using the public dataset from (Hou et al., 2025) for 2 epochs. Then, to iden-

Table 1: Experiment results on math reasoning tasks. We report Accuracy(%) for all datasets.

	MATH500	Omni-MATH-500	AIME2024	AMC	Olympiad Bench	LiveCode Bench	Avg
GPT-4o	76.6	26.8	9.3	45.8	43.3	29.5	38.6
Llama-3.1-8B-Instruct	52.8	15.0	10.9	22.6	15.6	11.6	21.4
Llama-3.3-70B-Instruct	73.9	27.9	24.2	50.9	35.7	25.5	39.7
GLM4-9B-chat	50.1	12.9	1.7	17.2	14.7	16.5	18.9
Qwen-2.5-7B-Instruct	76.5	26.0	13.3	41.9	35.0	16.8	34.9
Qwen-2.5-Math-7B-Instruct	82.7	29.7	16.7	50.6	40.7	8.1	38.1
Qwen-2.5-14B-Instruct	78.9	28.7	13.7	54.5	41.8	27.7	40.9
SFT (GLM-9B)	56.0	18.2	8.3	29.2	22.5	14.2	24.7
ChainRL (GLM-9B)	63.0	21.8	6.1	31.6	23.9	16.6	27.2
TreeRL (GLM-9B)	64.5	20.8	11.4	38.5	24.8	15.8	**29.3**
SFT (Qwen-2.5-14B)	76.6	29.5	10.6	48.0	36.9	14.5	36.0
ChainRL (Qwen-2.5-14B)	81.6	32.7	22.2	53.9	41.1	18.2	41.6
TreeRL (Qwen-2.5-14B)	81.7	36.7	28.0	55.9	44.6	20.8	**44.5**



(a) Experiments on Qwen-2.5-14B



(b) Experiments on GLM4-9B

Figure 6: Performance comparison between TreeRL and ChainRL during training. We report the average performance across all six datasets (Left), OlympiadBench (Middle), and Omni-MATH-500 (Right). The results for the other benchmarks can be found in the Appendix D.

tify the best tree search setting for RL under a given inference cost, we investigate various combinations of hyperparameters (M, N, L, T) using the trained Qwen-2.5-14B-SFT model on the Omni-MATH-500 dataset with PassRate as the target metric, and the results can be found in Table 4 in Appendix. The baseline i.i.d multi-chain samplings correspond to setting $N = L = T = 0$.

For the RL training, we compare the performance of RL using multi-chain sampling, referred to as ChainRL, to the proposed TreeRL under the same inference cost derived from the previous search to ensure a fair comparison. For TreeRL, we used sampling parameters $(M, N, L, T) = (6, 2, 1, 2)$, generating 30 responses per prompt. This is comparable to multi-chain sampling with 16 responses per prompt, given similar generation token budgets. Each iteration used 16 prompts for a single gradient update, resulting in a batch size of 256 for ChainRL and 480 for TreeRL. Therefore, TreeRL actually performs reinforcement learning with the same inference cost but with more training computation during training. We use rule-based reward based on the correctness of the final answer, i.e., $+1$ for correct and 0 for wrong. The KL efficiency is set to $\beta = 10^{-4}$, with a learning rate of 1.5×10^{-6}. During sampling, the temperature is 1.2, the top-p value is 0.95, and the maximum sequence length is 8,192. For RL training, the training data all come from publicly available datasets, including MATH-train (Hendrycks et al., 2021) and NuminaMath (Li et al., 2024a), and we sample a subset for training.

Evaluation We use PassRate to evaluate the effectiveness of the proposed tree search algorithm. PassRate measures the potential of a model to reach at least one correct answer among all the generated solutions. It should be noted that we compare tree sampling and chain sampling under similar inference budgets, i.e., generation tokens. Specifically, given an evaluation dataset $\mathcal{D}$, for each sample $d \in \mathcal{D}$, let $\mathcal{L}_d$ be the set of generated responses and $c(l) \in \{0, 1\}$ be a binary indicator of correctness for a response l. The PassRate is computed as:

$$\text{PassRate} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \max_{l \in \mathcal{L}_d} c(l)$$

For reinforcement learning, we evaluate the policy model on 6 challenging reasoning benchmarks using greedy sampling, including MATH(Hendrycks et al., 2021), Omni-MATH (Gao et al., 2024), AIME2024(Li et al., 2024a), AMC(Li et al., 2024a), OlympiadBench(He et al., 2024), and LiveCodeBench (Jain et al., 2024). For the MATH dataset, we use a subset known as MATH500 based on the split defined by (Lightman et al.). For Omni-MATH, we randomly sample a subset for evaluation, named Omni-MATH-500, which consists of 500 examples for efficient yet comprehensive assessment. Each dataset undergoes multiple evaluations to minimize variance; for instance, we evaluate MATH500 and Omni-MATH-500 three times each, while AIME is evaluated 32 times, considering AIME2024 consists of only 30 questions. For LiveCodeBench, we report the performance using the data between 202407 to 202411.

4.2 Results of EPTree

Table 4 shows the overall performance on Omni-MATH-500 and illustrates the efficiency and effectiveness trade-off across different sampling methods. EPTree demonstrates a significant advantage over the baseline multi-chain sampling method and the MCTS method under the same inference cost. And EPTree shows consistently better performance across different generation costs and outperforms the multi-chain sampling by around 3% in PassRate. Compared to MCTS, EPTree demonstrates a more significant advantage, with the margin widening as the inference cost increases.

4.3 Results on RL training

Table 1 presents the performance of various sampling strategies in RL training. Notably, TreeRL equipped with EPTree sampling outperforms traditional multi-chain sampling across different benchmarks. In particular, Figure 6 illustrates the evaluation performance over various training steps. While both sampling strategies exhibit similar performance during the early stages of training, TreeRL begins to show an advantage around 100 steps and continues to improve consistently. This indicates that the TreeRL achieves better prompt efficiency by delivering enhanced performance with the same number of training prompts. Overall, these results underscore the promise of integrating tree search and process supervision into RL training.

4.4 Ablation Study on EPTree Sampling

Effects of entropy-based forking. Table 2 illustrates the performance across different strategies when forking new branches in tree expansion. EPTree shows better PassRate than random forking with fewer generation tokens, which demonstrates the advantage of EPTree. In addition, both tree sampling strategies outperform multi-chain sampling, offering the potential of tree search.

Case study on forking tokens. To help better understand EPTree, we conduct case studies to analyze what type of tokens tend to be selected. We examine frequency by identifying the top-10 tokens that most often serve as forking points, and the results are illustrated in Figure 7. It can be observed that mathematical operators (\\(), logical conjunctions, and transitional terms (“Since”, “But”) are frequently selected. Notably, the token “wait” appears frequently, as o1-like models often use it for self-reflection steps.

Table 2: Ablation of EPTree on Omni-MATH-500. We compare forking branches using random and entropy-based strategies on Qwen-2.5-14B-SFT. $(16, 0, 0, 0)$ corresponds to multi-chain sampling. *Entropy* denotes whether to use entropy-based forking strategy. EPTree model shows better PassRate performance yet with fewer generation tokens.

(M, N, L, T)	Entropy	PassRate ↑	#Token ↓
$(6, 2, 1, 2)$	✓	**56.9**	**22268**
$(6, 2, 1, 2)$	✗	54.8	24213
$(16, 0, 0, 0)$	-	52.4	19858
$(8, 4, 2, 2)$	✓	**71.0**	**77768**
$(8, 4, 2, 2)$	✗	70.0	89452
$(64, 0, 0, 0)$	-	67.4	79367

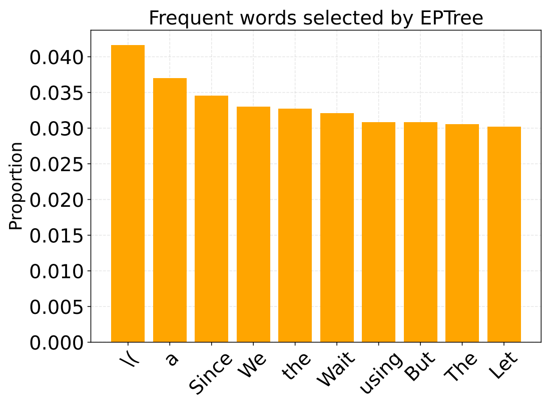


Figure 7: Frequent words of sampled forking tokens in EPTree sampling on Omni-MATH-500.

4.5 Ablation on TreeRL

To test the effectiveness of process supervision in TreeRL, we compare different designs of process supervision signals. Additionally, since the TreeRL uses more traces than RL with multi-chain sampling, we also evaluate how the increased training cost impacts performance. The results are presented in Table 3. The results demonstrate that RL with reweighted local and global advantage achieves the best performance. Removing either component could lead to a decline in performance. Moreover, using a subset of sampled responses shows less improvement compared to utilizing the entire set. This suggests that, in addition to higher PassRate and process supervision, tree search enhances RL performance by enabling more training within the same inference cost.

5 Related Work

5.1 Reinforcement Learning for LLM Reasoning

Recent advanced (Guo et al., 2025; Zhu et al., 2024; OpenAI, 2024; Ouyang et al., 2022) have demonstrated the effectiveness of reinforcement learning in LLM alignment and reasoning. Most existing methods train a reward model or use rule-based rewards for the entire response. In parallel, process supervision(Lightman et al., 2023) has shown a particularly promising performance than outcome supervision. Most existing works resort to training a process reward model (PRM) based on human- or auto-annotated process signals(Lightman et al., 2023; Wang et al., 2024a; Luo et al., 2024; Setlur et al., 2024) and apply the static PRM for RL training (Shao et al., 2024a; Wang et al., 2024a). However, a static PRM could suffer from distribution shift and reward hacking as RL training progresses. (Kazemnejad et al., 2024) overcomes this problem by conducting Monte Carlo rollouts to estimate the value for each step, but it suffers from high computation cost with around quadratic computational complexity and thus hinders scalability.

5.2 Tree Search for LLM Reasoning

Tree search has mainly been explored in LLM alignment and inference, especially for data synthesis and offline preference training. Xie et al. (2024) employs MCTS to generate step-level preference pairs for DPO (Rafailov et al., 2024). Chen et al. (2024b); Feng et al. (2024); Zhang et al. (2024b) employ MCTS to iteratively generate high-quality SFT data or produce process supervision signals for training. Other works explore reward-guided search (Snell et al., 2024; Yao et al., 2023; Long, 2023) to boost the performance in the inference stage. However, few efforts have been devoted to studying how to explicitly integrate tree search into reinforcement learning training like AlphaZero (Silver et al., 2017) to improve LLM reasoning.

6 Conclusion

This work presents TreeRL, an RL approach that combines tree search with process supervision to enhance LLM reasoning. EPTree improves response diversity and performance over traditional methods like MCTS and i.i.d multi-chain sampling. Then, we conduct reinforcement learning with EPTree and the derived process supervision from tree search. Experiments on math reasoning tasks show

Table 3: Ablation on the effectiveness of different process signals for RL training using Qwen2.5-14B. G_A and L_A denote the global and local advantages, respectively. n refers to the number of leaf nodes that can be reached from the given step. *#Response* denotes the number of responses used for training.

Reward	#Responses	MATH500	Omni-MATH-500	AIME2024	AMC	Olympiad Bench	LiveCode Bench	Avg Gain
$(G_A + L_A)/\sqrt{n}$	30	**81.7**	**36.7**	**28.0**	55.9	**44.6**	**20.8**	-
$G_A + L_A$	30	81.5	32.0	24.1	**56.2**	42.1	19.7	-1.9 ↓
$G_A/\sqrt{n}$	30	80.1	35.1	24.7	55.5	42.8	20.7	-1.3 ↓
$(G_A + L_A)/\sqrt{n}$	16	80.1	32.5	24.5	52.9	41.7	15.8	-3.2 ↓

that the TreeRL outperforms existing techniques, highlighting the potential of RL with tree search to advance LLM in complex reasoning tasks.

7 Limitation

In this work, we propose to improve reinforcement learning with on-policy tree search. While this approach demonstrates promising performance, it does come with several limitations. First, current LLM inference engines do not offer special optimizations for tree search, meaning the proposed EPTree still requires 2+ iterations, resulting in a performance that is approximately 2× slower than multi-chain sampling. Additionally, we utilize process supervision from tree search for RL training and attempt to optimize it from the perspective of advantage and re-weighting. Further efforts, including how to assign appropriate weights to the importance of different steps, how to define more meaningful process signals from the tree structure, and how to implement step-level reward normalization, deserve more exploration.

Acknowledgment. This work has been in part supported by the Natural Science Foundation of China (NSFC) 62495063, Tsinghua University (Department of Computer Science and Technology) - Siemens Ltd., China Joint Research Center for Industrial Intelligence and Internet of Things (JCIIOT), and the New Cornerstone Science Foundation through the XPLORER PRIZE. The corresponding author: Yuxiao Dong (yuxiaod@tsinghua.edu.cn).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

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

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

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

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

Wesley Chung, Valentin Thomas, Marlos C Machado, and Nicolas Le Roux. 2021. Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In *International Conference on Machine Learning*, pages 1999–2009. PMLR.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. Alphazero-like tree-search can guide large language model decoding and training. *Preprint*, arXiv:2309.17179.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. 2025. Advancing language model reasoning through reinforcement learning and inference scaling. *arXiv preprint arXiv:2501.11651*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

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

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. 2024a. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2024b. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

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*.

Jieyi Long. 2023. Large language model guided tree-of-thought. *Preprint*, arXiv:2305.08291.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. 2024. Improve mathematical reasoning in language models by automated process supervision. *Preprint*, arXiv:2406.06592.

Jincheng Mei, Wesley Chung, Valentin Thomas, Bo Dai, Csaba Szepesvari, and Dale Schuurmans. 2022. The role of baselines in policy gradient optimization. *Advances in Neural Information Processing Systems*, 35:17818–17830.

OpenAI. 2024. Learning to reason with llms. `https://openai.com/index/learning-to-reason-with-llms`.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 27730–27744.

Qwen. 2024. Qwen2.5: A party of foundation models.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024.

Rewarding progress: Scaling automated process verifiers for llm reasoning. *Preprint*, arXiv:2410.08146.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024a. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. 2024b. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward hacking. *Advances in Neural Information Processing Systems*, 35:9460–9471.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Peiyi Wang, Lei Li, Zhihong Shao, R. X. Xu, Damai Dai, Yifei Li, Deli Chen, Y. Wu, and Zhifang Sui. 2024a. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *Preprint*, arXiv:2312.08935.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024b. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. 2024c. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. *arXiv preprint arXiv:2405.00451*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Preprint*, arXiv:2305.10601.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024b. Rest-mcts*: Llm self-training via process reward guided tree search. *Preprint*, arXiv:2406.03816.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

A Position Distribution Analysis

We study the position of selected forking tokens in their branches. As illustrated in Figure 8, we plot the relative positions of forking points, calculated as the ratio between a token's forking position and its branch length. The resulting distribution shows a roughly uniform pattern, which supports our assumptions presented in Appendix B.

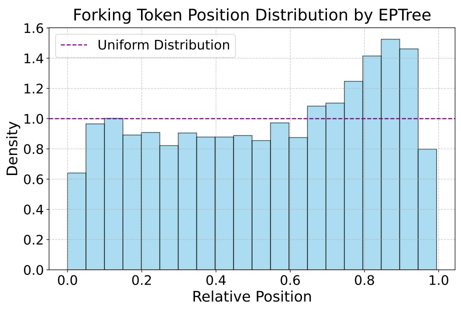


Figure 8: Distribution of the relative positions of forking tokens selected by EPTree on Omni-MATH-500, based on the ratio of forking position to branch length.

B Theoritical Analysis of EPTree

Theorem 1. *Consider a setting where forking tokens follow a uniform distribution $\mathcal{U}(0, 1)$ within each branch, and generation lengths remain fixed without repetition, subject to the constraint that parameter $l \leq 2$. Under these conditions, the entropy-guided tree search algorithm presented in Section 3.1 yields a leaf count that is bounded between $\frac{4}{3}$ and $\frac{12}{5}$ times that of a multi-sampling approach, while maintaining identical computational complexity in terms of total token evaluations.*

Proof. We present the proof of Theorem 1 by analyzing two cases based on the branching process illustrated in Figure 9. Let $x_1, x_2, \ldots, x_n$ be i.i.d. random variables uniformly distributed on $[0, 1]$.

Case $l = 1$: Assume all completion lengths are unit 1. The total completion length of the entropy-tree is:

$$L = 1 + t \cdot \sum_{i=1}^{n} (1 - x_i).$$

The expected total length is:

$$\mathbb{E}[L] = 1 + t \cdot \sum_{i=1}^{n} \mathbb{E}[1 - x_i] = 1 + \frac{nt}{2}.$$

The number of leaves in the tree is:

$$N_{\text{tree}} = 1 + nt.$$

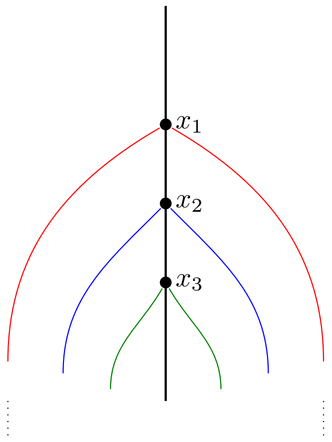


Figure 9: Forking token illustration when $n = 3, t = 2$, where $x_1, x_2, x_3 \overset{\text{i.i.d}}{\sim} \mathcal{U}(0, 1)$.

For the multi-sample approach with the same completion tokens, the number of leaves is:

$$N_{\text{multi}} = 1 + \frac{nt}{2}.$$

The ratio of the number of leaves in the tree to the multi-sample approach is:

$$R(n, t) = \frac{1 + nt}{1 + \frac{nt}{2}}.$$

Let $x = nt \geq 1$. Then:

$$R(x) = \frac{1 + x}{1 + \frac{x}{2}}.$$

The derivative of $R(x)$ with respect to x is:

$$R'(x) = \frac{1}{\left(1 + \frac{x}{2}\right)^2} > 0,$$

indicating that $R(x)$ is monotonically increasing. Evaluating at the endpoints:

$$R(1) = \frac{4}{3}, \quad \lim_{x \to \infty} R(x) = 2.$$

Thus, $R(n, t) \in \left[\frac{4}{3}, 2\right)$ when $l = 1$.

Case $l = 2$: In this scenario, the next top-n entropy token appears either on the main chain or on one of the branches. The probabilities of these events are:

$$\frac{1}{1 + t \sum_i (1 - x_i)} \quad \text{and} \quad \frac{1 - x_i}{1 + t \sum_i (1 - x_i)},$$

Respectively. The expected new completion length is:

$$\mathbb{E}\left[l_i\right] = \mathbb{E}\left[\frac{\frac{1}{2} + \frac{t}{2}\sum_i (1-x_i)^2}{1 + t\sum_i (1-x_i)}\right] := \varphi.$$

Through Monte Carlo simulation, it is observed that φ monotonically decreases with n. As $n \to \infty$, the ratio tends to $\frac{1}{3}$. For $n = 1$, the integral value is $\ln 2 - \frac{1}{4}$. Thus, $\varphi \in \left(\frac{1}{3}, \ln 2 - \frac{1}{4}\right]$.

The total generation length is:

$$1 + \frac{1}{2}nt + nt \cdot \varphi,$$

And the number of leaves is:

$$1 + 2nt.$$

For the same completion length, the number of leaves in the multi-sample approach is:

$$1 + \left(\frac{1}{2} + \varphi\right) nt.$$

The ratio of the number of leaves in the tree to the multi-sample approach is:

$$R(nt) = \frac{1 + 2nt}{1 + \left(\frac{1}{2} + \varphi\right) nt},$$

Which is also monotonically increasing in nt. Therefore:

$$R(nt) \in \left[\frac{6}{3 + 2\varphi}, \frac{4}{1 + 2\varphi}\right).$$

Specifically:

$$\begin{cases} R(nt) \geq \dfrac{6}{3 + 2 \cdot \left(\ln 2 - \frac{1}{4}\right)} \approx 1.544 \\ R(nt) < \dfrac{4}{1 + 2 \cdot \frac{1}{3}} = 2.4 \end{cases}$$

Combining both cases, the ratio $R(n, t)$ lies in the interval $\left[\frac{4}{3}, \frac{12}{5}\right)$.

□

Table 4: (M, N, L, T) Parameter Evaluation of EPTree on Omni-MATH-500. The table reports the leaf number, PassRate, and total generation tokens under different parameter combinations for $k = 16$ and $k = 64$.

(M, N, L, T)	#Leaf ↑	PassRate ↑	#Token ↓
		$k = 16$	
multi-16	16	52.4	19858
(8, 3, 1, 1)	32	58.4	25223
(7, 2, 1, 2)	35	59.2	26200
(6, 2, 2, 1)	30	54.4	20537
(6, 2, 1, 2)	30	56.9	22268
(5, 3, 1, 2)	35	58.6	25269
(5, 2, 1, 3)	35	56.0	25210
(5, 3, 2, 1)	35	58.0	22668
(5, 1, 2, 3)	35	58.6	21895
		$k = 64$	
multi-64	64	67.4	79367
(16, 2, 2, 2)	144	70.0	88257
(16, 4, 1, 2)	144	71.4	101744
(16, 2, 1, 4)	144	72.6	100468
(9, 5, 1, 3)	144	71.9	98193
(9, 3, 1, 5)	144	71.6	97193
(8, 8, 1, 2)	136	70.7	92946
(8, 4, 1, 4)	136	69.7	90015
(8, 8, 2, 1)	136	68.6	79582
(8, 4, 2, 2)	136	71.0	77768
(8, 2, 2, 4)	136	69.4	76750

C Detailed Evaluation of EPTree

Table 4 presents a comprehensive evaluation of EPTree in comparison to the multi-chain baseline for $k = 16$. We select the RL training configuration $(6, 2, 1, 2)$, as it offers a similar inference cost to the chain-16 setting while demonstrating improved efficiency (requiring only one interaction) and effectiveness (achieving 56.9 compared to 52.4).

D Results on Other Reasoning Benchmarks

Figure 10 and 11 provide additional RL training results on MATH500, AMC, LiveCodeBench, and AIME2024, demonstrating that TreeRL with EPTree sampling outperforms RL with traditional multi-chain sampling across various benchmarks.

E TreeRL on General tasks

To further evaluate TreeRL's generalizability, we further conducted experiments on general tasks. We evaluate TreeRL and ChainRL on 3 general

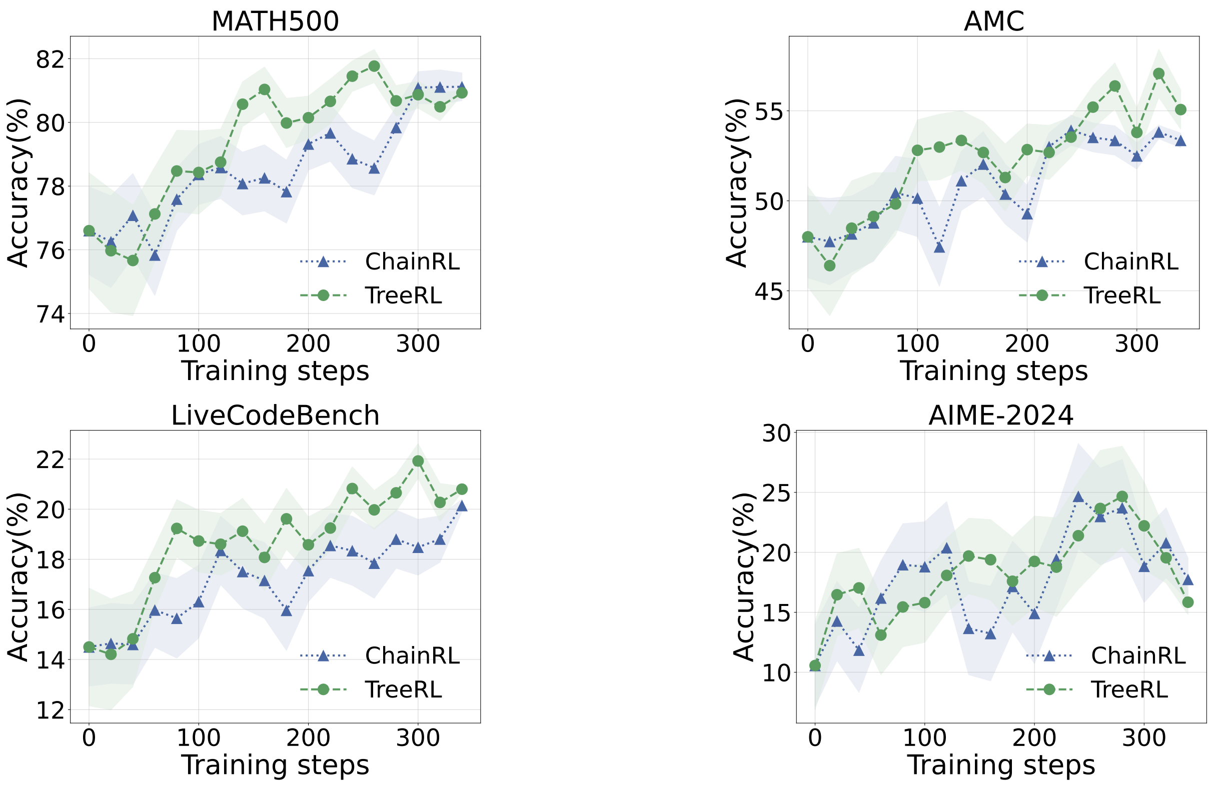


Figure 10: Performance comparison between TreeRL and ChainRL on MATH500 (Upper Left), AMC (Upper Right), LiveCodeBench (Lower Left), and AIME2024 (Lower Right); experiments are based on Qwen-2.5-14B.

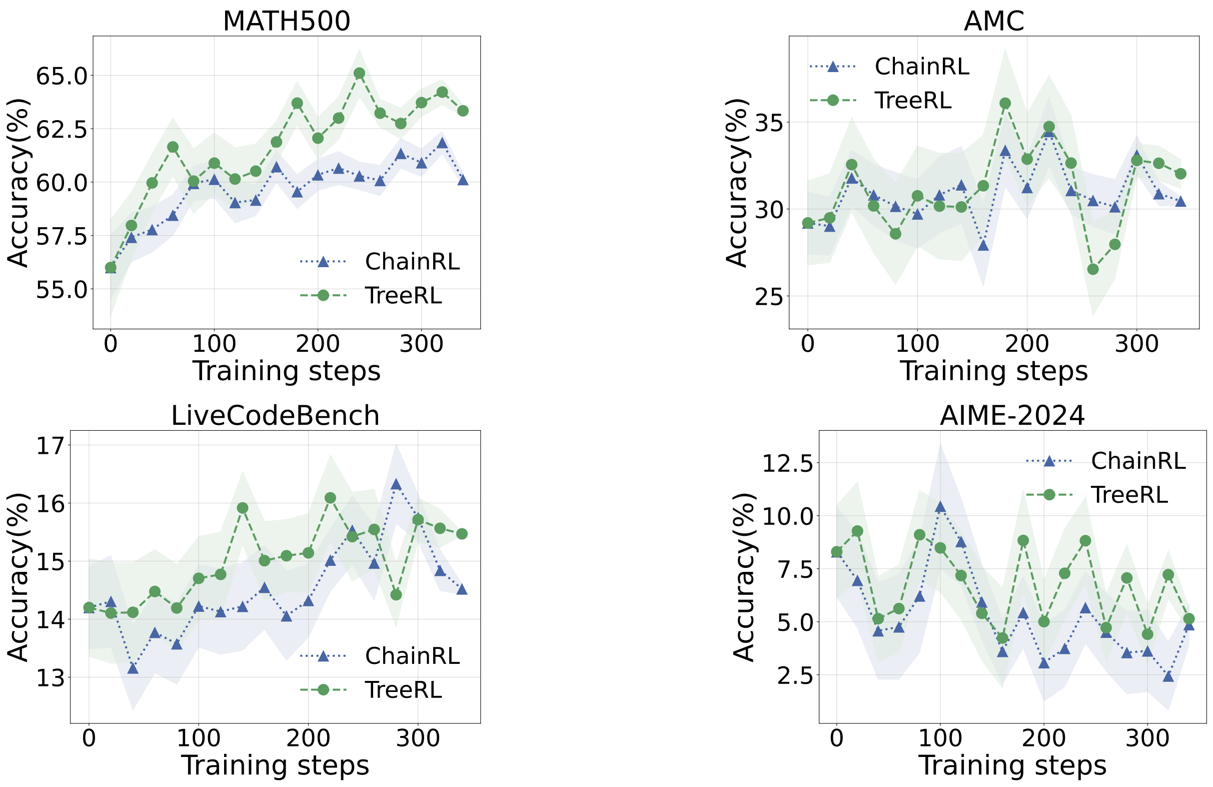


Figure 11: Performance comparison between TreeRL and ChainRL on MATH500 (Upper Left), AMC (Upper Right), LiveCodeBench (Lower Left), and AIME2024 (Lower Right); experiments are based on GLM4-9B.

tasks: MMLU-Pro (Wang et al., 2024c), Arena-Hard (Li et al., 2024b), and IFEval (Zhou et al., 2023).

- MMLU-Pro extends MMLU (Hendrycks et al., 2020) with more challenging reasoning tasks, fewer noisy questions, and a larger answer set (4-10 options). We use the chain-of-thought prompt and measure the pass rate.
- Arena-Hard consists of 500 difficult prompts from Chatbot Arena, focusing on human-like preferences. We evaluate using the win rate score, comparing against the GPT-4-0314 baseline.
- IFEval tests the model's ability to follow prompt instructions. We use the strict prompt metric for evaluation.

	MMLU-Pro	Arena-hard	IFEval	Avg
SFT	57.6	57.5	49.9	55.0
ChainRL	64.5	72.2	56.0	64.2
TreeRL	64.5	71.8	58.2	64.9

Table 5: Performance on general benchmarks.

As is shown in Table 5, we observe a comparable performance between TreeRL and ChainRL. This indicates that while TreeRL exhibits a particular advantage in reasoning tasks, it maintains its performance in general tasks, achieving robust performance in diverse task types.