# AlphaZero-Like Tree-Search can Guide Large Language Model Decoding and Training

Ziyu Wan [*1] Xidong Feng [*2]
Muning Wen [1] Stephen Marcus McAleer [3] Ying Wen [1] Weinan Zhang [1] Jun Wang [2]

## Abstract

Recent works like Tree-of-Thought (ToT) and Reasoning via Planning (RAP) aim to augment the multi-step reasoning capabilities of LLMs by using tree-search algorithms. These methods rely on prompting a pre-trained model to serve as a value function and focus on problems with low search depth. As a result, these methods cannot benefit from in-domain training and only rely on pretraining process — they will not work in domains where the pre-trained LLM does not have enough knowledge to serve as an effective value function or in domains that require long-horizon planning. To address these limitations, we present an AlphaZero-like tree-search learning framework for LLMs (termed TS-LLM), systematically illustrating how tree-search with a learned value function can guide LLM decoding. TS-LLM distinguishes itself in two key ways. (1) Leveraging a learned value function and AlphaZero-like algorithms, our approach can be generally adaptable to a wide range of tasks, language models of any size, and tasks of varying search depths. (2) Our approach can guide LLMs during both inference and training, iteratively improving the LLMs. Empirical results across reasoning, planning, alignment, and decision-making tasks show that TS-LLM outperforms existing approaches and can handle trees with a depth of 64.

## 1. Introduction

Large language models (LLMs) (OpenAI, 2023; Touvron et al., 2023a) have demonstrated their potential in a wide range of natural language tasks. A plethora of recent studies have concentrated on improving LLMs task-solving capability, including curation of larger and higher-quality general or domain-specific data (Touvron et al., 2023a; Zhou et al., 2023; Gunasekar et al., 2023; Feng et al., 2023; Taylor et al., 2022), more sophisticated prompt design (Wei et al., 2022; Zhou et al., 2022; Creswell et al., 2022), or better training algorithms with Supervised Learning or Reinforcement Learning (RL) (Dong et al., 2023; Gulcehre et al., 2023; Rafailov et al., 2023). When training LLMs with RL, LLMs' generation can be naturally formulated as a Markov Decision Process (MDP) and optimized with specific objectives. Following this formulation, ChatGPT (Ouyang et al., 2022) emerges as a notable success, optimizing LLMs to align human preference by leveraging Reinforcement Learning from Human Feedback (RLHF) (Christiano et al., 2017).

LLMs can be further guided with planning algorithms such as **tree search**. Preliminary work in this field includes Tree-of-Thought (ToT) (Yao et al., 2023; Long, 2023) with depth/breadth-first search and Reasoning-via-Planing (RAP) (Hao et al., 2023) with MCTS. They successfully demonstrated a performance boost of searching on trees expanded by LLM through self-evaluation. Despite these advances, current methods come with distinct limitations. First, the value functions in the tree-search algorithms are obtained by prompting LLMs. As a result, such algorithms lack general applicability and heavily rely on both well-designed prompts and the robust capabilities of advanced LLMs. Beyond the model requirements, we will also show in Sec. 4.2.1 that such prompt-based self-evaluation is not always reliable. Second, ToT and RAP use BFS/DFS and MCTS for tree search, restricting their capabilities to relatively simple and shallow tasks. They are capped at a maximum depth of only 10 or 7, which is significantly less than the depth achieved by AlphaZero in chess or Go (Silver et al., 2017). As a result, ToT and RAP might struggle with complex problems that demand large analytical depths and longer-term planning horizons, decreasing their scalability.

To address these problems, we introduce tree-search enhanced LLM (TS-LLM), an AlphaZero-like framework that utilizes tree-search to improve LLMs' performance on gen-

[*]Equal contribution, order determined by flipping a coin. [1]Shanghai Jiao Tong University [2]University College London [3]Carnegie Mellon University. Correspondence to: Ying Wen <ying.wen@sjtu.edu.cn>, Jun Wang <jun.wang@ucl.ac.uk>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

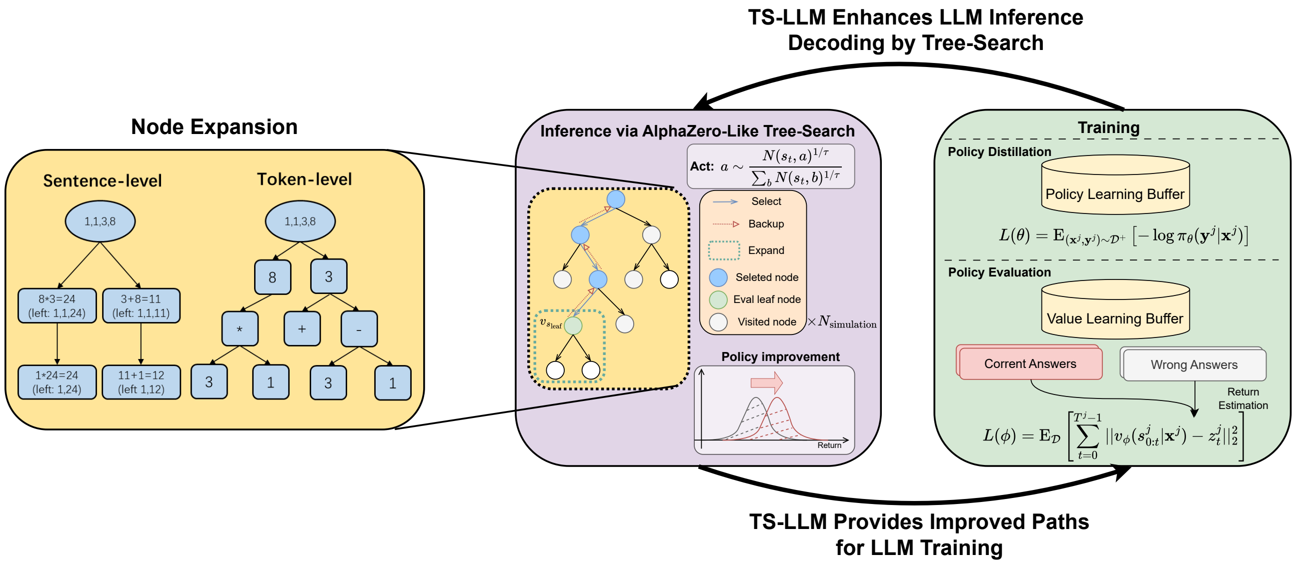
TS-LLM Enhances LLM Inference Decoding by Tree-Search
Node Expansion
Sentence-level
Token-level
1,1,3,8
8*3=24 (left: 1,1,24)
3+8=11 (left: 1,1,11)
1*24=24 (left: 1,24)
11+1=12 (left 1,12)
8
3
*
+
-
3
1
3
1
Inference via AlphaZero-Like Tree-Search
Act: $a \sim \frac{N(s_t,a)^{1/\tau}}{\sum_b N(s_t,b)^{1/\tau}}$
Select
Backup
Expand
Seleted node
Eval leaf node
Visited node
$\times N_{\text{simulation}}$
$v_{s_{\text{leaf}}}$
Policy improvement
Return
Training
Policy Distillation
Policy Learning Buffer
$L(\theta) = \mathrm{E}_{(\mathbf{x}^j,\mathbf{y}^j)\sim\mathcal{D}^+}\left[-\log \pi_\theta(\mathbf{y}^j|\mathbf{x}^j)\right]$
Policy Evaluation
Value Learning Buffer
Corrent Answers
Wrong Answers
Return Estimation
$L(\phi) = \mathrm{E}_{\mathcal{D}}\left[\sum_{t=0}^{T^j-1} ||v_\phi(s^j_{0:t}|\mathbf{x}^j) - z^j_t||^2_2\right]$
TS-LLM Provides Improved Paths for LLM Training

Figure 1: (a) Left: Two node expansion paradigms on Game24: sentence-level and token-level. We adopt sentence-level setting in this task. (b) Right: TS-LLM consists of an iterative process over tree-search and training. First, TS-LLM enhances LLM inference by tree-search to obtain improved trajectories, augmenting the training set. LLM can be further trained to improve by conducting policy distillation and value function learning (policy evaluation) over the augmented training set.

eral natural language tasks. TS-LLM extends previous work to AlphaZero-like deep tree-search with a learned LLM-based value function which can guide the LLM during both inference and training. Compared with previous work, TS-LLM has the following two new features:

- **TS-LLM offers a generally applicable and scalable pipeline.** It is generally **applicable**: With a learned value function, TS-LLM can be applied to various tasks and LLMs of any size. Our learned value function can be more reliable than the prompt-based counterpart and does not require any well-designed prompts or advanced, large-scale LMs. Our experiments show that TS-LLM can work for LLMs ranging from 125M to 7B parameters, providing better evaluation even compared with GPT-3.5. TS-LLM is also **scalable**: TS-LLM can conduct deep tree search, extending tree-search for LLM generation up to a depth of 64. This is far beyond 10 in ToT and 7 in RAP.

- **TS-LLM can potentially serve as a new LLM training paradigm beyond inference decoding.** By treating the tree-search operation as a policy improvement operator, we can conduct an iterative processes of improving the policy via tree search and then improving the policy through distillation and the value function through the ground-truth training labels on the tree search trajectories.

Through comprehensive empirical evaluations on reasoning, planning, alignment, and decision-making tasks, we present an in-depth analysis of the core design elements in TS-LLM, delving into the features, advantages, and limitations of different variations. This showcases TS-LLM's potential as a universal framework to guide LLM decoding and training.

## 2. Related Work

**Multistep Reasoning in LLMs** Multistep reasoning in language models has been widely studied, from improving the base model (Chung et al., 2022; Fu et al., 2023; Lewkowycz et al., 2022) to prompting LLMs step by step (Kojima et al., 2022; Wei et al., 2022; Wang et al., 2022; Zhou et al., 2022). Besides, a more relevant series of work has focused on enhancing reasoning through evaluations, including learned reward models (Uesato et al., 2022; Lightman et al., 2023) and self-evaluation (Shinn et al., 2023; Madaan et al., 2023). In this work, we apply evaluations to multistep reasoning tasks and token-level RLHF alignment tasks by learning a value function and reward model under the setting of a multistep decision-making process.

**Search-Guided Reasoning in LLMs** While most CoT approaches have used a linear reasoning structure, recent efforts have been made to investigate non-linear reasoning structures such as trees (Jung et al., 2022; Zhu et al., 2023; Chaffin et al., 2021; Lamprier et al., 2022; Scialom et al., 2021). More recently, various approaches for searching on trees have been applied to find better reasoning paths, e.g. beam search in Xie et al. (2023), depth-/breadth-first search in Yao et al. (2023) and Monte-Carlo Tree Search in Hao et al. (2023). Compared with these methods, TS-

LLM is a tree search guided LLM decoding and training framework with a learned value function, which is more generally applicable to both reasoning tasks and other scenarios like RLHF alignment tasks. Moreover, we leave a comprehensive comparison in Appendix A.

**Finetuning LLMs with Augmentation** Recent efforts have also been made to improve LLMs with augmented data. Rejection sampling is a simple and effective approach for finetuning data augmentation to improve LLMs' ability on single/multiple task(s) such as multistep reasoning (Yuan et al., 2023a; Zelikman et al., 2022) and alignment with human preference (Dong et al., 2023; Bai et al., 2022). Given an augmented dataset, reinforcement learning approaches have also been used to finetune LLMs (Gulcehre et al., 2023; Luo et al., 2023). Compared to previous works, TS-LLM leverages tree search as a policy improvement operator to generate augmented samples to train both the LLMs and the value function.

## 3. Enhancing LLMs with Tree Search

In this section, we propose a versatile tree-search framework TS-LLM for guiding LLMs decoding and training. We conduct a systematic and comprehensive analysis of its key components. TS-LLM is summarized in Fig. 1.

### 3.1. Problem Formulation

We formulate the language generation process as a multi-step Markov Decision Process (MDP). The particularity of natural language tasks is that both the action and state are in language space. LLMs can serve as a policy $\pi_\theta$ that samples sequences of tokens as actions. Assuming the length of the output sequence and input prompt are $T$ and $L$ respectively, the probability for an LLM policy to produce an output sequence $\mathbf{y} = (y_0, y_1, \dots, y_{T-1})$ conditioned on a prompt (input prefix) $\mathbf{x} = (x_0, x_1, \dots, x_{L-1})$ is: $\pi_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=0}^{T-1} \pi_\theta(y_t|\mathbf{x}_{0:L-1}, \mathbf{y}_{0:t-1})$.

For a given natural language task, we can define a reward function $R(y_t|\mathbf{x}_{0:L-1}, \mathbf{y}_{0:t-1})$ as the task performance feedback for intermediate generation $y_t$ at timestep $t$. Due to the lack of large-scale and high-quality intermediate reward labels for general tasks, it is usually a sparse reward setting where any intermediate reward from the first $T-1$ timestep is zero except the last $T$-th step. A typical case can be RLHF alignment task, where LLM can receive the reward signal after it completes the full generation. Following the same logic, $\mathbf{y}$ can also be viewed as a sequence of sentences.

Given the problem formulation above, we successfully transfer the problem of better generation to optimization for higher cumulative reward. In this paper, we focus on how we can optimize it with **tree-search algorithms**. A specific natural language task typically predefines the state space (with language) and reward function (with task objective/metrics). What remains is the definition of action space, or in the context of tree-search algorithm, the action node.

Tree search algorithms have validated their effectiveness for different action spaces in traditional RL research, including discrete action space (Silver et al., 2017; Schrittwieser et al., 2020a) and continuous action space (Hubert et al., 2021). For tree-search on LLMs, we consider the following two action space designs, as shown in the left side of Fig. 1.

**Sentence-level action nodes**: For the tasks that have a step/sentence-level structure(e.g. chain-of-thought reasoning), it is natural to treat each thought as a sentence-level action node. This is also the technique adopted by ToT (Yao et al., 2023) and RAP (Hao et al., 2023). For each non-terminal node, the search tree is expanded by sampling several possible subsequent intermediate steps and dropping the duplicated generations.

**Token-level action nodes**: Analogous to tree-search in discrete action space MDP, we can treat each token as a discrete action for LLM policy and the tree search can be conducted in token-level. For those tasks in which intermediate steps are not explicitly defined(e.g. RLHF), splitting an output sequence into tokens might be a good choice.

We refer to Appendix B.1 for a comparison about advantages and limitations over the search space of these two action space designs.

### 3.2. Guiding LLM Inference Decoding with Tree Search

One of the benefits of tree-search algorithms is that they can optimize the cumulative reward by mere search, without any gradient calculation or update. In this section, given a fixed LLM policy, we present the full pipeline to illustrate how to guide LLM inference decoding with tree search approaches.

#### 3.2.1. Learning an LLM-based Value Function

For tree-search algorithms, how to construct reliable value function $v$ and reward model $\hat{r}$ is the main issue. ToT and RAP obtain these two models by prompting advanced LLMs, such as GPT-4 or LLaMA-33B. To make the tree search algorithm generally applicable, our method leverages a learned LLM-based value function $v_\phi(s)$ conditioned on state $s$ and a learned final-step outcome reward model (ORM) $\hat{r}_\phi$ since most tasks can be formulated as sparse-reward problems (Uesato et al., 2022). Since we mainly deal with language-based tasks, we utilize a shared value network and reward model whose structure is a decoder-only transformer with an MLP to output a scalar on each position of the input tokens. And typically, LLM value's decoder is adapted from original LLM policy $\pi_\theta$'s decoder, or the LLM value $v_\phi$ and policy $\pi_\theta$ can have a shared decoder (Silver et al., 2017). For a sentence-level expanded intermediate step $s_t$, we use the prediction scalar at the last token as its value prediction $v_\phi(s_t)$. The final reward can be obtained at the last token when feeding the full sentences $(\mathbf{x}_{0:L-1}, \mathbf{y}_{0:T-1})$ into the model.

Therefore, we use language model $\pi_\theta$ as the policy to sample generations using the task training dataset. With true label or a given reward function in training data, a set of sampled tuple $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^j, \mathbf{y}^j, r^j)\}_j$ of size $|\mathcal{D}_{\text{train}}|$ can be obtained, where $\mathbf{x}^j$ is the input text, $\mathbf{y}^j = s^j_{0:T^j-1}$ is the output text of $T^j$ steps and $r^j = R(\mathbf{y}^j|\mathbf{x}^j)$ is the ground-truth reward. Similar to the critic training in most RL algorithms, we construct the value target $z^j_t$ by TD-$\lambda$ (Sutton, 1988) or MC estimate (Sutton & Barto, 2018) on each single step $t$ . The value network is optimized by mean squared error:

$$L(\phi) = \mathrm{E}_{\mathcal{D}}\left[\sum_{t=0}^{T^j-1} ||v_\phi(s^j_{0:t}|\mathbf{x}^j) - z^j_t||_2^2\right]. \tag{1}$$

The ORM $\hat{r}_\phi(\mathbf{y}_{0:T-1}|\mathbf{x}_{0:L-1})$ is learned with the same objective. Training an accurate value function and ORM is quite crucial for the tree-search process as they provide the main guidance. We will further illustrate how to learn a reliable value function and ORM in our experiment section.

### 3.2.2. Tree Search Algorithms

Given a learned value function, in this section, we present five types of tree-search algorithms. We leave detailed background, preliminaries, and comparisons in Appendix B.2.

**Breadth-First and Depth-First Search With Value Function-Based Tree-Pruning (BFS-V/DFS-V)**: These two search algorithms were adopted in ToT (Yao et al., 2023). The core idea is to utilize the value function to prune the tree for efficient search, while such pruning happens in tree breadth or depth respectively. BFS-V can be regarded as a beam-search with cumulative reward as the objective.

**MCTS**: This approach was adopted in RAP (Hao et al., 2023), which refers to classic MCTS (Kocsis & Szepesvári, 2006). It back-propagates the value on the terminal nodes, relying on a Monte-Carlo estimate of value, and it starts searching from the initial state node.

In addition to these algorithms adopted by ToT and RAP, we consider two new variants of AlphaZero-like tree-search.

**MCTS with Value Function Approximation** (named as MCTS-$\alpha$): This is the MCTS variants utilized in AlphaZero (Silver et al., 2017). Starting from the initial state, we choose the node of state $s_t$ as the root node and do several times of search simulations consisting of *select, expand and evaluate* and *backup*, where the leaf node value evaluated by the learned value function will be backpropagated to all its ancestor nodes. After the search, we choose an action proportional to the root node's exponentiated visit count, i.e. $a \sim \frac{N(s_t,a)^{1/\tau}}{\sum_b N(s_t,b)^{1/\tau}}$, and move to the corresponding next state. The above iteration will be repeated until finished. MCTS-$\alpha$ has two main features. Firstly, MCTS-$\alpha$ cannot trace back to its previous states once it takes an action. So it cannot restart the search from the initial state unless multiple searches are conducted which will be discussed in Section 3.2.3. Secondly, in contrast to MCTS, MCTS-$\alpha$ utilizes a value function so it can conduct the backward operation during the intermediate steps, without the need to complete the full generation to obtain a Monte-Carlo estimate.

**MCTS-Rollout**: Combining the features from MCTS and MCTS-$\alpha$, we propose a new variant MCTS-Rollout for tree search. Similar to MCTS, MCTS-Rollout always starts from the initial state node. It further does the search simulations analogous to MCTS-$\alpha$, and the *backup* process can happen in the intermediate step with value function. It repeats the operations above until the process finds $N$ complete answers or reaches the computation limit (e.g. maximum number of tokens.) MCTS-Rollout can be seen as an offline version of MCTS-$\alpha$ so they may have similar application scope. The only difference is that MCTS-Rollout can scale up the token consumption for better performance since it always reconducts the search from the beginning.

### 3.2.3. Multiple Search and Search Aggregation

Inspired by Wang et al. (2022) and Uesato et al. (2022) that LLM can improve its performance on reasoning tasks by sampling multiple times and aggregating the candidates, TS-LLM also has the potential to aggregate $N$ complete answers generated by multiple tree searches or multiple generations from one search (set BFS beam size $> 1$).

When conducting multiple tree searches, we usually adopt **intra-tree search** setting. Intra-tree search conducts multiple tree searches on the same tree, thus the state space is exactly the same. Such a method is computationally efficient as the search tree can be reused multiple times. However, the diversity of multiple generations might decrease because the former tree search might influence the latter tree searches. Also, the search space is limited in sentence-level action space because they will be fixed once expanded across multiple tree searches.

We refer to Appendix D.7 for an alternative setting called Inter-tree Search where we allow resampling in the expansion process, and without further specification, all settings in our paper are under the intra-tree search setting. Our next step is to aggregate these search results to obtain the final answer. With a learned ORM, we consider the following three different aggregation methods:

**Majority-Vote.** Wang et al. (2022) aggregates answers using majority vote: $f^* = \arg\max_f \sum_{\mathbf{y}^j} \mathbf{1}_{\text{final\_ans}(\mathbf{y}^j)=f}$, where $\mathbf{1}$ is the indicator function.

Table 1: Task setups. The node, tree max with and tree max depth are search space parameters. Refer to Appendix C.2 and D.5 for how max tree-width and tree-depth are determined.

| | | | Search Space Hyperparameters | | |
|---|---|---|---|---|---|
| Task | Category | Train/test size | Node | Tree Max width | Tree Max depth |
| GSM8k | Mathematical Reasoning | 7.5k / 1.3k | Sentence | 6 | 8 |
| Game24 | Mathematical Planning | 1.0k / 0.3k | Sentence | 20 | 4 |
| PrOntoQA | Logical Reasoning | 4.5k / 0.5k | Sentence | 6 | 15 |
| RLHF | Alignment | 30k / 3k | Token | 50 | 64 |
| Chess Endgame | Decision Making | 0.1M/0.6k | Sentence | 5 | 50 |

**ORM-Max.** Given an outcome reward model, the aggregation can choose the answer $f$ with maximum final reward, $f^* = \text{final\_ans}(\arg\max_{\mathbf{y}^j} \hat{r}_\phi(\mathbf{y}^j|\mathbf{x}^j))$.
**ORM-Vote.** Given an outcome reward model, the aggregation can choose the answer $f$ with the sum of rewards, namely $f^* = \arg\max_f \sum_{\mathbf{y}^j;\text{final\_ans}(\mathbf{y}^j)=f} \hat{r}_\phi(\mathbf{y}^j|\mathbf{x}^j)$.

### 3.3. Enhancing LLM Training with Tree Search

In section 3.2 we discuss how tree-search can guide LLM's decoding process during inference time. Such guidance leads to a better decoding strategy and improves the performance of given tasks. In other words, tree-search guidance can serve as a policy improvement operator. Based on this, we propose a new training and finetuning paradigm.

Assume we have an initial LLM policy $\pi_{\theta_{\text{old}}}$ (trained by conducting supervised finetuning over the original training set) and initial LLM value and ORM: $v_{\phi_{\text{old}}}$, $\hat{r}_{\phi_{\text{old}}}$ (trained by Equ. 1 from sampling the original training questions), we can have the following iterative process:

**Policy Improvement**: We conduct tree-search over training set based on $\pi_{\theta_{\text{old}}}$, $v_{\phi_{\text{old}}}$, and $\hat{r}_{\phi_{\text{old}}}$ to obtain improved generations, resulting in the augmented dataset $\mathcal{D}$ and also the filtered positive examples $\mathcal{D}^+$.

**Policy Distillation:** With the tree-search-improved dataset $\mathcal{D}^+$, by imitating the tree-search positive trajectories, LLM policy can be further improved to $\pi_{\theta_{\text{new}}}$ with supervised loss.

$$L(\theta) = \mathrm{E}_{(\mathbf{x}^j,\mathbf{y}^j)\sim\mathcal{D}^+}\left[-\log \pi_\theta(\mathbf{y}^j|\mathbf{x}^j)\right]. \tag{2}$$

**Policy Evaluation**: We train value function $v_{\phi_{\text{new}}}$ and ORM $\hat{r}_{\phi_{\text{new}}}$ over the augumented dataset $\mathcal{D}$ under loss of Equ 1.

These three processes can be conducted cyclically to iteratively refine the LLM. Such iterative process belongs to generalized policy iteration (Sutton & Barto, 2018), which is also the procedure used in AlphaZero's training. In our case, the training process involves finetuning three networks on the tree-search augmented dataset: (1) Policy network $\pi_\theta$: Use cross-entropy loss with trajectories' tokens as target (2) Value network $v_\phi$: Mean squared error loss with trajectories' temporal difference (TD) or Monte-Carlo (MC) based value estimation as target, and (3) ORM $\hat{r_\phi}$: Mean squared error loss with trajectories' final reward as target.

### 3.4. Tree Search's Extra Computation Burdens

Tree-search algorithms will inevitably bring in additional computation burdens, especially in the node expansion phase for calculating legal child nodes and their corresponding value. Prior methodologies, such as ToT and RAP, tend to benchmark their performance against baseline algorithms using an equivalent number of generation paths (named Path@$N$). This approach overlooks the additional computational demands of the tree-search process. We also refer readers to Appendix C.3 for discussion about the computation efficiency and engineering challenges.

A more fair comparison requires monitoring the number of tokens generated for node expansion. This provides a reasonable comparison of algorithms' performance when operating under comparable token generation conditions. We address this issue in our experiments (Sec. 4.2.2).

## 4. Experiments

In this section, we conduct thorough experiments to address and analyze each subsection we mention in Section 3. We refer to (Niu et al., 2024) for implementation of tree-search algorithms. Our code is open-sourced at https://github.com/waterhorse1/LLM_Tree_Search.

### 4.1. Experiment Setups

**Task Setups** For a given MDP, the nature of the search space is primarily characterized by two dimensions: depth and width. To showcase the efficacy of tree-search algorithms across varied search spaces, we evaluate all algorithms on five tasks with different search widths and depths, including the mathematical reasoning task GSM8k (Cobbe et al., 2021), mathematical planning task Game24 (Yao et al., 2023), logical reasoning task PrOntoQA (Saparov & He, 2022), RLHF alignment task using synthetic RLHF data (Dahoas), and chess endgames (Abdulhai et al., 2023). The specific task statistics and search space hyperparameters are listed in Table 1. These hyperparameters, especially max search width and search depth, are determined by our exploration experiments. They can effectively present the characteristics of a task and define its search space. Refer to Appendix D.1 for more details of our evaluation tasks.

**Benchmark Algorithms.** We compare ToT-GPT3.5 and TS-LLM in Table 2 to verify the effectiveness of learned value function. All tree-search algorithms will be benchmarked, including MCTS-$\alpha$, MCTS-Rollout, MCTS, BFS-V, and DFS-V. Note that BFS-V, DFS-V, and MCTS are TS-LLM's variants instead of ToT (Yao et al., 2023) or RAP (Hao et al., 2023) baselines because we adopt a learned value function rather than prompting LLM. We compare these variants with direct decoding baselines, including CoT greedy decoding, and CoT with self-consistency (Wang et al., 2022) (denoted as CoT-SC). Considering the search space gap between direct decoding and tree decoding (especially the sentence-level action node), we include the CoT-SC-Tree baseline which conducts CoT-SC over the tree's sentence nodes.

**Model and Training Details.** For the rollout policy used in tree-search, we use LLaMA2-7B (Touvron et al., 2023b) on three reasoning tasks, and GPT-2-small (125M) on the RLHF task and Chess endgame. All LLMs will be first supervise-finetuned (SFT) on the training set, enabling their CoT ability without examples in prompt. For value and ORM training, the data are generated by sampling the SFT policy's rollouts on the training set. Our policy LLM and value LLM are two separate models but are adapted from the same base model, and we train different models on each task. Refer to Appendix C.4 for experiments on shared models.

## 4.2. Results and Discussions

### 4.2.1. Learned Value function (Sec 3.2.1)

We successfully show that **the learned value function can be more reliable than prompt-based GPT-3.5**, even though GPT-3.5 is much stronger than LLaMA2-7B. In Table 2, we conduct BFS Path@1 comparisons with different combinations of policy and value over Game 24 and GSM8K. The policy choices include few-shot GPT-3.5 and our supervised finetuned LLaMA2-7B. For the value, we utilize prompt-based GPT-3.5/LLaMA2-7B (TOT) and our learned value function LLaMA2-V. Even though the few-shot GPT-3.5 policy is an out-of-distribution policy for LLaMA2-V's evaluation, LLaMA2-V still presents dominant performance over prompt-based GPT-3.5/LLaMA2-7B in all settings. The phenomenon of LLM's limited self-evaluation ability by prompting aligns with other papers (Huang et al., 2023; Stechly et al., 2023). This finding largely increases the necessity of a learned value function.

Table 2: ToT-BFS Path@1 results with different combinations of policy and value. LLaMA-SFT and LLaMA-V refer to the trained policy and value, LLaMA and GPT-3.5 (ToT) refer to the prompt-based model for policy or value. LLaMA-V dominates compared with prompt-based value.

| Task | Policy | Value | Accuracy(%) |
|---|---|---|---|
| GSM8K | GPT-3.5 | GPT-3.5 (ToT) | 72.7 |
| | GPT-3.5 | LLaMA-V (Ours) | **74.0** |
| | LLaMA-SFT | LLaMA (ToT) | 37.4 |
| | LLaMA-SFT | GPT-3.5 (ToT) | 45.8 |
| | LLaMA-SFT | LLaMA-V (Ours) | **52.5** |
| Game24 | GPT-3.5 | GPT-3.5 (ToT) | 15.5 |
| | GPT-3.5 | LLaMA-V (Ours) | **19.1** |
| | LLaMA-SFT | LLaMA (ToT) | 9.2 |
| | LLaMA-SFT | GPT-3.5 (ToT) | 21.0 |
| | LLaMA-SFT | LLaMA-V (Ours) | **64.8** |

### 4.2.2. Performance on different algorithms (Sec 3.2.2, Sec 3.4)

With a reliable learned value function, we compare the performance of different generation methods. First, in the upper part of Table 3, we present the Path@1 results of MCTS-$\alpha$ and MCTS-Rollout compared to BFS-V (BFS-/DFS-V and MCTS degenerate to greedy value tree-search in path@1 case) and CoT-Greedy. The experiment results show that **AlphaZero-like search algorithms, MCTS-$\alpha$ and MCTS-Rollout significantly outperforms the baselines in tasks where long-horizon planning matters** (RLHF and Chess Endgame). When searching on shallow trees, they are robust enough to maintain comparable accuracy to the baselines.

Despite the superiority of TS-LLM, we argue that the Path@1/Path@$N$ metric may not be reasonable. We also include the number of computations used in Path@1 generation (average number of tokens in sentence-level and average number of forward computation in token-level of solving one single problem). We refer readers to the **second** row of Fig 2 for Path@$N$ result, with token/forward number as the x-axis. TS-LLM variants consume much more computation than CoT, making the comparison unfair.

To enable a fair comparison, in the bottom part of Table 3, we show the "Equal-Token" results that try to compare results by controlling a similar scale of computation consumption with Path@1 TS-LLM. First, we provide additional baselines, CoT-SC with two aggregation methods: majority-vote (MAJ) and ORM-vote (denoted as ORM, and it utilizes the learned ORM in TS-LLM). Under this situation, TS-LLM's advantages largely decrease when compared with CoT-SC$_{\text{ORM}}$, especially on GSM8K (only BFS greedy value search is the best). We are surprised to see that such simple algorithms can also have outstanding performance when

Table 3: Path@1 and Equal-token results of all TS-LLM variants and the CoT baseline with #token. For the alignment task and Chess Endgame, $SC_{ORM}$ means the best score of the sampled candidates, and $SC_{MAJ}$ refers to the average. We only present BFS-V in Path@1 because MCTS/BFS-V/DFS-V degenerates to greedy value search. For the Equal-token setting, We do not show the results of BFS-/DFS-V/$MCTS_{ORM}$ in GSM8K since token consumption is similar in the Path@1 setting.

| Setting | Method | Performance(%) / # Tokens | | | | | | Reward / # Forward | | Win Rate / # Tokens | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GSM8k | | Game24 | | PrOntoQA | | RLHF(token-level) | | Chess Endgame | |
| Path@1 | CoT-greedy | 41.4 | 98 | 12.7 | 76 | 48.8 | 92 | 0.318 | 57.8 | 58.14 | 37.4 |
| | BFS-V (Ours) | **52.5** | 485 | 64.8 | 369 | 94.4 | 126 | -1.295 | 61.8 | 67.75 | 402 |
| | MCTS-$\alpha$ (Ours) | 51.9 | 561 | 63.3 | 412 | **99.4** | 190 | **2.221** | 186 | 96.90 | 797 |
| | MCTS-Rollout (Ours) | 47.8 | 3.4k | **71.3** | 670 | 96.9 | 210 | 1.925 | 809 | **98.76** | 615 |
| Equal-Token | CoT-$SC_{MAJ}$ | 46.8 | 500 | 14.6 | 684 | 61.1 | 273 | -0.253 | 580 | 9.84 | 782 |
| | CoT-$SC_{ORM}$ | **52.3** | 500 | 50.6 | 684 | 83.2 | 273 | 1.517 | 580 | 73.80 | 782 |
| | BFS-$V_{ORM}$ (Ours) | - | - | 70.90 | 1.6k | - | - | -1.065 | 613 | 93.18 | 854 |
| | DFS-$V_{ORM}$ (Ours) | - | - | 69.09 | 962 | 96.4 | 195 | -0.860 | 86 | 71.01 | 511 |
| | $MCTS_{ORM}$ (Ours) | - | - | 69.34 | 649 | **99.6** | 182 | 0.160 | 592 | 94.26 | 706 |

compared fairly. Despite this, **most tree-search algorithms are still dominant in the rest four tasks given the larger search space (CoT-SC)**.

Besides, we also compare the behaviors of BFS-/DFS-V and MCTS when searching for multiple paths (aggregated by the ORM model) within a comparable range of computation consumptions. Comparing these 3 variants, MCTS is almost the best w.r.t. both performance and computation cost, this indicates **the importance of value back-propogation**. While comparing with the Path@1 results, MCTS-$\alpha$ and MCTS-Rollout achieve comparable accuracy in shallow-search problems (GSM8k, Game24, and ProntoQA), and dominate in deep-search ones (RLHF and Chess Endgame). It verifies the necessity of **Alphazero-style intermediate value back-propagation under deep-search problems**.

### 4.2.3. Search Aggregation (Sec 3.2.3)

In Fig. 2, we demonstrate the mean/max reward for the RLHF task and the best of 3 aggregation results for GSM8K, Game24 and ProntoQA. We measure the performance of aggregation w.r.t path number and token consumption.

From the figure, we mainly summarize two conclusions: First, **Most TS-LLM variants benefit from aggregation and can show large strengths compared with other baselines.** CoT-SC only beats TS-LLM in GSM8k with the same token size, mainly because of its larger search space. We refer the readers to Appendix C.2 for additional results on GSM8K and ProntoQA. Second, **tree-search algorithms' aggregation benefits less than CoT-SC in small-scale problems.** In GSM8K and Game24, TS-LLM struggles to improve under large aggregation numbers. We believe this is because of: (1) The search space gap between CoT-SC and tree-search algorithms. Tree-search algorithms inherently explore fewer sentences, which is validated by comparing token consumption between CoT-SC-Tree@50 and CoT-SC@50. (2) Tree-search algorithms already leverage the value function and ORM, the benefits of aggregation with the ORM again become less obvious.

In all, the scalability of tree-search aggregation is an open question that is worth further exploration in future work.

Table 4: Iterative update results. $\theta_0, \phi_0$ are the old parameters while $\theta_1, \phi_1$ are the new ones. TS-LLM can boost performance by training LLM policy, value, or both.

| Task | Method | Policy | Value | Accuracy(%) |
|---|---|---|---|---|
| GSM8K | Greedy | $\pi_{\theta_0}$ | - | 41.4 |
| | | $\pi_{\theta_1}$ | - | **47.9** |
| | | RFT-50 | - | 47.0 |
| | | RFT-100 | - | 47.5 |
| | MCTS-$\alpha$ | $\pi_{\theta_0}$ | $v_{\phi_0}, \hat{r}_{\phi_0}$ | 51.9 |
| | | $\pi_{\theta_0}$ | $v_{\phi_1}, \hat{r}_{\phi_1}$ | 53.2 |
| | | $\pi_{\theta_1}$ | $v_{\phi_0}, \hat{r}_{\phi_0}$ | 54.1 |
| | | $\pi_{\theta_1}$ | $v_{\phi_1}, \hat{r}_{\phi_1}$ | **56.5** |
| RLHF | Greedy | $\pi_{\theta_0}$ | - | 0.39 |
| | | $\pi_{\theta_1}$ | - | 1.87 |
| | | RFT N=5 | - | 1.16 |
| | | PPO | - | **2.53** |
| | MCTS-$\alpha$ | $\pi_{\theta_0}$ | $v_{\phi_0}, \hat{r}_{\phi_0}$ | 2.22 |
| | | $\pi_{\theta_0}$ | $v_{\phi_1}, \hat{r}_{\phi_1}$ | 2.48 |
| | | $\pi_{\theta_1}$ | $v_{\phi_0}, \hat{r}_{\phi_0}$ | 2.53 |
| | | $\pi_{\theta_1}$ | $v_{\phi_1}, \hat{r}_{\phi_1}$ | **2.67** |

### 4.2.4. TS-LLM for training LLM (Sec. 3.3)

We conduct initial experiments for one iterative update in GSM8k and RLHF alignment task. We utilize MCTS-$\alpha$ with old policy $\pi_{\theta_0}$, value $v_{\phi_0}$ and ORM $\hat{r}_{\phi_0}$, to sample answers on the training dataset as an augmentation to the origin one. It will be further used to finetune these models to $(\pi_{\theta_1}, v_{\phi_1}, \hat{r}_{\phi_1})$. We include two baselines, RFT (Yuan

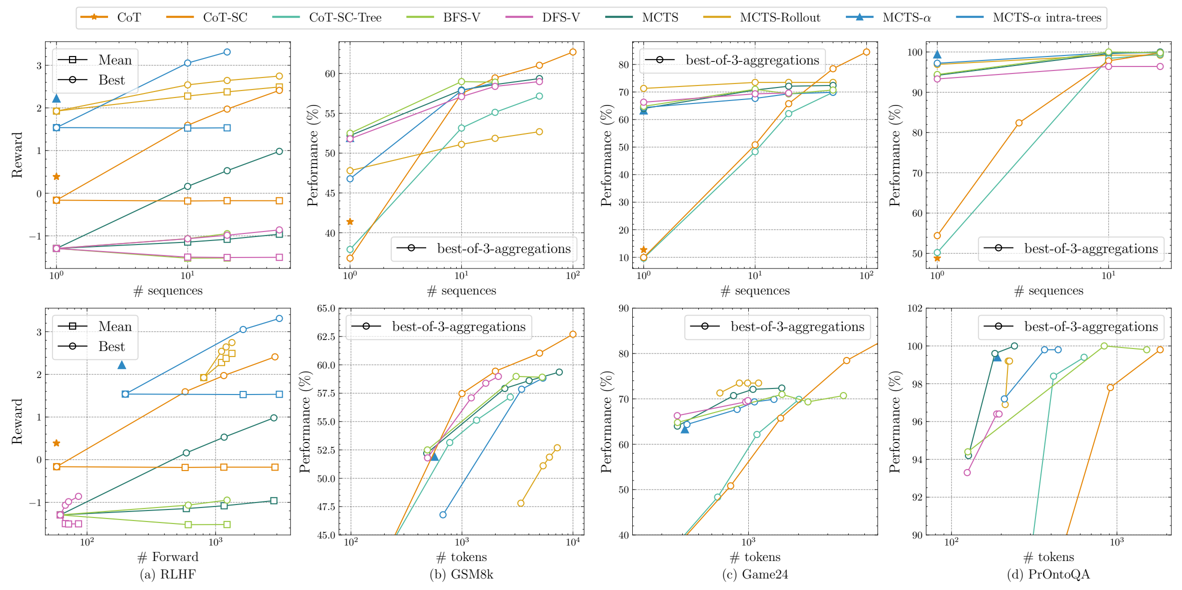

Figure 2: Aggregation results for four tasks w.r.t. number of sequences(Path@$N$) on the 1st row and the number of tokens on the 2nd row. TS-LLM benefits from aggregation but struggles to scale in small-scale problems.

et al., 2023b), which utilizes rejection sampling to finetune the policy, with different sampling numbers $k$ or top $N$, and PPO (Schulman et al., 2017) for the RLHF task. Note that PPO conducts multiple iterative updates. It is not fully comparable to our method and we only add it for completeness. Refer to Appendix D.8 and D.9 for more experimental details and Appendix C.1 for an ablation about how to use the collected data to optimize the the value and ORM.

In Table. 4, we list results of iterative update on the GSM8K and RLHF, covering greedy decoding and MCTS-$\alpha$ over all policy and value combinations. **Our empirical results validate that TS-LLM can further train LLM policy, value and ORM, boosting performance with the new policy $\pi_{\theta_1}$, new value and ORM $\{v, \hat{r}\}_{\phi_1}$, or both $(\pi_{\theta_1}, \{v, \hat{r}\}_{\phi_1})$** in CoT greedy decoding and MCTS-$\alpha$. $\pi_{\theta_1}$'s greedy performance is even slightly better than RFT which is specifically designed for GSM8k. We believe by further extending TS-LLM to multi-update, we can make it more competitive though currently $\pi_{\theta_1}$ still cannot beat PPO-based policy.

### 4.2.5. Ablation studies

**How to learn value function** We investigate data collection and training paradigms for value function and ORM in TS-LLM. In Table 5, we investigate the influence of data amount and diversity by training with mixed data uniformly sampled from checkpoints of all SFT epochs (*mixed*); data purely sampled from the last checkpoint (*pure*); and we sample 1/3 data of the (*pure*) setting and discard the rest to formulate the (*pure, less*) setting. The results of CoT-SC$_{\text{ORM-vote}}$@10 underscore the diversity of sampled data in learning a better ORM. The Path@1 results of 3 TS-LLM variants show that the amount of sampled data is of great importance. Our analysis suggests that collecting a diverse dataset can contribute to improvements in the ORM, though the effect observed in our study was relatively modest. Our final conclusion is that **collecting as much data as possible is better for value function training.** We also leave an ablation about how to use the collected data to optimize the value and ORM for iterative update in Appendix C.1.

Table 5: Path@1 results with different training settings.

| Search Algorithms | Training Setting | Accuracy(%) |
|---|---|---|
| CoT-SC@10@ORM | Pure, Less | 55.5 ± 0.6 |
| | Pure | 55.3 ± 0.5 |
| | Mixed | **55.9** ± **0.7** |
| BFS | Pure, Less | 50.0 ± 0.3 |
| | Pure | **52.7** ± **0.8** |
| | Mixed | 52.5 ± 1.3 |
| MCTS-$\alpha$ | Pure, Less | 49.7 ± 1.1 |
| | Pure | **52.7** ± **0.8** |
| | Mixed | **51.9** ± **0.6** |

**Search space and search width** As discussed in Sec. 3.1, the search space is limited by maximum tree width. We demonstrate the influence introduced by different tree constructions on Game24 with different node expansion sizes. Specifically, we set the number of maximal expanded node sizes as 6, 20, and 50. Table 6 lists the Path@1 performance and the number of tokens generated comparing TS-LLM's variants, CoT-SC and CoT-SC-Tree. The almost doubled

Table 6: Game24 Path@1 result with different tree-width. Larger search space leads to much better performance.

| | Performance(%) / # tokens | | |
|---|---|---|---|
| Method | width=6 | width=20 | width=50 |
| MCTS-$\alpha$ | 41.6 | 63.3 | 74.5 |
| MCTS-Rollout | 43.8 | 71.3 | 80.7 |
| BFS-V | 43.2 | 64.8 | 74.6 |
| CoT-SC-Tree@10 | 38.8 | 48.3 | 48.3 |
| CoT-SC@10 | - | - | 52.9 |

performance boost from 43.8 to 80.7 indicates the impact of different expansion sizes and tree-width, improving TS-LLM's performance upper bound. Refer to Appendix C.2 for additional results on GSM8K and ProntoQA.

## 5. Conclusion

In this work, we propose TS-LLM, an LLM inference and training framework guided by Alphazero-like tree search that is generally versatile for different tasks and scaled to token-level expanded tree spaces. Empirical results validate that TS-LLM can enhance LLMs decoding and serve as a new training paradigm.

### 5.1. Limitation and Future Work

Currently, our method TS-LLM still cannot scale to really large-scale scenarios due to the extra computation burdens introduced by node expansion and value evaluation. Additional engineering work such as key value caching is required to accelerate the tree-search. We add detailed results in terms of wall-time and engineering challenges in Appendix C.3. In addition, we do not cover all feasible action-space designs for tree search and it is flexible to propose advanced algorithms to automatically construct a tree mixed with both sentence-level expansion and token-level expansion, etc. We leave such exploration for future work. For MCTS aggregation, the current method still struggles to improve under large aggregation numbers. some new algorithms that can encourage multi-search diversity might be needed. Currently, we are still actively working on scaling up our method both during inference and training.

## Acknowledgements

The SJTU team is supported by National Key R&D Program of China (2022ZD0114804), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102) and National Natural Science Foundation of China (62106141, 62322603, 62076161).

## Impact Statement

Drawing inspiration from AlphaZero's remarkable achievements in the game of Go, our work leverages a similar tree search approach to guide the decoding and training of large language models. We believe our method holds promise for replicating such success, potentially leading to the realization of super-intelligent capabilities in a large range of natural language abilities, such as reasoning, planning, and decision-making.

From an ethical perspective, current large language models can often produce hallucinations of false information. Our framework aims to instill more step-by-step reasoning/planning in these models, leading to more interpretable and trustworthy language generation.

## References

Abdulhai, M., White, I., Snell, C., Sun, C., Hong, J., Zhai, Y., Xu, K., and Levine, S. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models, 2023.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Chaffin, A., Claveau, V., and Kijak, E. Ppl-mcts: Constrained textual generation through discriminator-guided mcts decoding. *arXiv preprint arXiv:2109.13582*, 2021.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Creswell, A., Shanahan, M., and Higgins, I. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*, 2022.

Dahoas. Synthetic-instruct-gptj-pairwise. https://huggingface.co/datasets/Dahoas/synthetic-instruct-gptj-pairwise.

Dong, H., Xiong, W., Goyal, D., Pan, R., Diao, S., Zhang, J., Shum, K., and Zhang, T. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023.

Feng, X., Luo, Y., Wang, Z., Tang, H., Yang, M., Shao, K., Mguni, D., Du, Y., and Wang, J. Chessgpt: Bridging policy learning and language modeling. *arXiv preprint arXiv:2306.09200*, 2023.

Fu, Y., Peng, H., Ou, L., Sabharwal, A., and Khot, T. Specializing smaller language models towards multi-step reasoning. *arXiv preprint arXiv:2301.12726*, 2023.

Gulcehre, C., Paine, T. L., Srinivasan, S., Konyushkova, K., Weerts, L., Sharma, A., Siddhant, A., Ahern, A., Wang, M., Gu, C., et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Hao, S., Gu, Y., Ma, H., Hong, J. J., Wang, Z., Wang, D. Z., and Hu, Z. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Hubert, T., Schrittwieser, J., Antonoglou, I., Barekatain, M., Schmitt, S., and Silver, D. Learning and planning in complex action spaces. In *International Conference on Machine Learning*, pp. 4476–4486. PMLR, 2021.

Jung, J., Qin, L., Welleck, S., Brahman, F., Bhagavatula, C., Bras, R. L., and Choi, Y. Maieutic prompting: Logically consistent reasoning with recursive explanations. *arXiv preprint arXiv:2205.11822*, 2022.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Lamprier, S., Scialom, T., Chaffin, A., Claveau, V., Kijak, E., Staiano, J., and Piwowarski, B. Generative cooperative networks for natural language generation. In *International Conference on Machine Learning*, pp. 11891–11905. PMLR, 2022.

Leblond, R., Alayrac, J.-B., Sifre, L., Pislar, M., Lespiau, J.-B., Antonoglou, I., Simonyan, K., and Vinyals, O. Machine translation decoding beyond beam search. *arXiv preprint arXiv:2104.05336*, 2021.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liu, J., Cohen, A., Pasunuru, R., Choi, Y., Hajishirzi, H., and Celikyilmaz, A. Making ppo even better: Value-guided monte-carlo tree search decoding, 2023.

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

Luo, H., Sun, Q., Xu, C., Zhao, P., Lou, J., Tao, C., Geng, X., Lin, Q., Chen, S., and Zhang, D. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

Matulewicz, N. Inductive program synthesis through using monte carlo tree search guided by a heuristic-based loss function. 2022.

Niu, Y., Pu, Y., Yang, Z., Li, X., Zhou, T., Ren, J., Hu, S., Li, H., and Liu, Y. Lightzero: A unified benchmark for monte carlo tree search in general sequential decision scenarios. *Advances in Neural Information Processing Systems*, 36, 2024.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Rosin, C. D. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3): 203–230, 2011.

Saparov, A. and He, H. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020a.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020b.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Scialom, T., Dray, P.-A., Staiano, J., Lamprier, S., and Piwowarski, B. To beam or not to beam: That is a question of cooperation for language gans. *Advances in neural information processing systems*, 34:26585–26597, 2021.

Segal, R. B. On the scalability of parallel uct. In *International Conference on Computers and Games*, pp. 36–47. Springer, 2010.

Shinn, N., Labash, B., and Gopinath, A. Reflexion: an autonomous agent with dynamic memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Stechly, K., Marquez, M., and Kambhampati, S. Gpt-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. *arXiv preprint arXiv:2310.12397*, 2023.

Sutton, R. S. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V., and Stojnic, R. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Xie, Y., Kawaguchi, K., Zhao, Y., Zhao, X., Kan, M.-Y., He, J., and Xie, Q. Decomposition enhances reasoning via self-evaluation guided decoding. *arXiv preprint arXiv:2305.00633*, 2023.

Xu, H. No train still gain. unleash mathematical reasoning of large language models with monte carlo tree search guided by energy function. *arXiv preprint arXiv:2309.03224*, 2023.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Yuan, Z., Yuan, H., Li, C., Dong, G., Tan, C., and Zhou, C. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023a.

Yuan, Z., Yuan, H., Li, C., Dong, G., Tan, C., and Zhou, C. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023b.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Zhang, S., Chen, Z., Shen, Y., Ding, M., Tenenbaum, J. B., and Gan, C. Planning with large language models for code generation. In *The Eleventh International Conference on Learning Representations*, 2022.

Zheng, L., Yin, L., Xie, Z., Huang, J., Sun, C., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., et al. Efficiently programming large language models using sglang. *arXiv preprint arXiv:2312.07104*, 2023.

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., et al. Lima: Less is more for alignment. *arXiv preprint arXiv:2305.11206*, 2023.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q., et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

Zhu, X., Wang, J., Zhang, L., Zhang, Y., Huang, Y., Gan, R., Zhang, J., and Yang, Y. Solving math word problems via cooperative reasoning induced language models. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4471–4485, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.245. URL https://aclanthology.org/2023.acl-long.245.

## A. More Related Work and Comparisons

Here we discuss the differences between TS-LLM and relevant work mentioned in Sec 2 in detail.

Recent efforts have been made to investigate non-linear reasoning structures such as trees (Jung et al., 2022; Zhu et al., 2023; Xu, 2023; Xie et al., 2023; Yao et al., 2023; Hao et al., 2023). Approaches for searching on trees with LLM's self-evaluation have been applied to find better reasoning paths, e.g. beam search in Xie et al. (2023), depth-/breadth-first search in Yao et al. (2023) and Monte-Carlo Tree Search in Hao et al. (2023). Compared with these methods, TS-LLM is a tree search guided LLM decoding framework with a learned value function, which is more generally applicable to reasoning tasks and other scenarios like RLHF alignment. TS-LLM includes comparisons between different tree search approaches, analysis of computation cost, and shows the possibility of improving both the language model and value function iteratively. The most relevant work, CoRe (Zhu et al., 2023), proposes to finetune both the reasoning step generator and learned verifier for solving math word problems using MCTS for reasoning decoding which is the most relevant work to ours. Compared with CoRe, in this work TS-LLM distinguishes itself by:
**1.** TS-LLM is generally applicable to a wide range of reasoning tasks and text generation tasks under general reward settings, from sentence-level trees to token-level trees. But CoRe is proposed for Math Word Problems and only assumes a binary verifier (reward model).
**2.** In this work, we conduct comprehensive comparisons between popular tree search approaches on reasoning, planning, and RLHF alignment tasks. We fairly compare linear decoding approaches like CoT and CoT-SC with tree search approaches w.r.t. computation efficiency.
**3.** TS-LLM demonstrates potentials to improve LLMs' performance of direct decoding as well as tree search guided decoding, while in CoRe the latter cannot be improved when combining the updated generator (language model policy) with the updated verifier (value function) together.

Other related topic:
**Search guided decoding in LLMs** Heuristic search and planning like beam search or MCTS have also been used in NLP tasks including machine translation (Leblond et al., 2021) and code generation (Zhang et al., 2022; Matulewicz, 2022). During our preparation for the appendix, we find a concurrent work (Liu et al., 2023) which is proposed to guide LLM's decoding by reusing the critic model during PPO optimization to improve language models in alignment tasks. Compared with this work, TS-LLM focuses on optimizing the policy and value model through tree search guided inference and demonstrates the potential of continuously improving the policy and value models. And TS-LLM is generally applicable to both alignment tasks and reasoning tasks by conducting search on token-level actions and sentence-level actions.

## B. Backgrounds and details of each tree-search algorithms in TS-LLM

### B.1. Pros and Cons of token-level and action-level node expansion

Typically, the search space is determined by two algorithm-agnostic parameters, the **tree max width** $w$ and **tree max depth** $d$. In LLM generation, both action space designs have their advantages and limitations over the search space. By splitting the generation into sentences, **sentence-level action nodes** provide a relatively shallow tree (low tree max-depth), simplifying the tree-search process. However, the large sample space of sentence-level generation makes full enumeration of all possible sentences infeasible. We have to set a maximum tree width $w$ to subsample $w$ nodes during the expansion, similar to the idea of Sampled MuZero (Hubert et al., 2021) (The node will be fixed once it is expanded). Such subsampling results in the gap, determined by $w$, between the tree-search space and the LLM generation space. For **token-level action nodes**, though it can get rid of the search space discrepancy and extra computational burdens, it greatly increases the depth of the tree, making tree-search more challenging.

### B.2. Preliminaries of Monte Carlo Tree-search Algorihtms

Once we build the tree, we can use various search algorithms to find a high-reward trace. However, it's not easy to balance between exploration and exploitation during the search process, especially when the tree is sufficiently deep. Therefore we adopt Monte Carlo Tree Search(MCTS) variants as choices for strategic and principled search. Instead of the four operations in traditional MCTS (Kocsis & Szepesvári, 2006; Coulom, 2006), we refer to the search process in AlphaZero (Silver et al., 2017) and introduce 3 basic operations of a standard search simulation in it as follows, when searching actions from current state node $s_0$:

***Select*** It begins at the root node of the search tree, of the current state, $s_0$, and finishes when reaching a leaf node

$s_L$ at timestep $L$. At each of these $L$ timesteps(internal nodes), an action(child node) is selected according to $a_t = \arg\max_a (Q(s_t, a) + U(s_t, a))$ where $U(s_t, a)$ is calculated by a variant of PUCT algorithm (Rosin, 2011):

$$U(s,a) = c_{\text{puct}} \cdot \pi_\theta(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \tag{3}$$

where $N(s, a)$ is the visit count of selecting action $a$ at node $s$, and $c_{\text{puct}} = \log((\sum_b N(s, b) + c_{\text{base}} + 1)/c_{\text{base}}) + c_{\text{init}}$ is controlled by visit count and two constants. This search control strategy initially prefers actions with high prior probability and low visit count, but asymptotically prefers actions with high action-value.

***Expand and evaluate*** After encountering a leaf node $s_L$ by *select*, if $s_L$ is not a terminal node, it will be expanded by the language model policy. The state of the leaf node is evaluated by the value network, noted as $v(s_L)$. If $s_L$ is a terminal node, if there is an oracle reward function $R$, then $v(s_L) = R(s_L)$, otherwise, in this paper, we use an ORM $\hat{r}$ as an approximation of it.

***Backup*** After *expand and evaluate* on a leaf node, backward the statistics through the path $s_L, s_{L-1}, \dots, s_0$, for each node, increase the visit count by $N(s_t, a_t) = N(s_t, a_t) + 1$, and the total action-value are updated as $W(s_t, a_t) = W(s_t, a_t) + v(s_L)$, the mean action-value are updated as $Q(s_t, a_t) = W(s_t, a_t)/N(s_t, a_t)$.

### B.3. Comparison of tree-search algorithms in TS-LLM

In this paper, we introduce three variants of MCTS based on the above basic operations. Among the 3 variants, MCTS-$\alpha$ is closer to AlphaZero(Silver et al., 2017) and MCTS is closer to traditional Monte-Carlo Tree Search(Kocsis & Szepesvári, 2006). While MCTS-Rollout is closer to best-first search or A*-like tree search. We show the pseudocode of MCTS-$\alpha$ and MCTS-Rollout in Algorithm 3 and Algorithm 2 respecitvely.

**Algorithm 1** Simplied_MCTS_Simulation

```
1: Input: current tree node with state $s_t$
2: Use PUCT to select a path from current node untill find a leaf node with state $s_L$.
3: Evaluate value of the leaf with state $s_L$ by $v_\phi$ or $\hat{r}_\phi$
4: if the leaf with state $s_L$ is not a terminal node then
5:     Expand the leaf node with state $s_L$ by $\pi_\theta(\cdot|s_L)$
6: end if
7: Backup values and other information like $W(s_i, a_i), N(s_i, a_i)$ from the leaf node to all
   its ancestors denoted as $s_i$.
```

**Algorithm 2** MCTS-Rollout

```
1: Input: an MCTS tree T
2: repeat
3:     Simplied_MCTS_Simulation(T.root)
4: until Stoppiong criterion like reaching maximal number of simulations/generated tokens
5: Extract Answer(s) $ans$ from the tree T
6: Return $ans$
```

**Algorithm 3** MCTS-$\alpha$

```
1:  Input: an MCTS tree T, number of simulations $n_{\text{simulation}}$
2:  Initialize node of current state $s$ = T.root
3:  repeat
4:      for $i = 0$ to $n_{\text{simulation}} - 1$ do
5:          Simplied_MCTS_Simulation(node with current state $s$)
6:      end for
7:      Select an action $a \sim \frac{N(s_t,a)^{1/\tau}}{\sum_b N(s_t,b)^{1/\tau}}$
8:      Update current node to be the child node with state $s$ of executing the selected action $a$
9:  until Current state $s$ is a terminal state.
10: Extract Answer $ans$ from the tree T
11: Return $ans$
```

The major difference between the first three MCTS variants and BFS-V/DFS-V adopted from the ToT paper(Yao et al., 2023) is that the first three MCTS variants will propagate (i.e. the *backup* operation) the value and visit history information through the search process. MCTS-$\alpha$ and MCTS-Rollout bacpropagate after the *expand* operation or visiting a terminal node. MCTS backpropagates the information only after visiting a terminal node. For DFS-V, the children of a non-leaf node is traversed in a non-decreasing order by value. For efficient exploration, we tried 2 heuristics to prune the subtrees, (1) drop the children nodes with low value by *prune_ratio*. (2) drop the children nodes lower than a *prune_value*.

## C. Extra Experiments and Discussions

### C.1. Different Value Training of iterative update

Table 7: Different value training for iterative update on GSM8k

| Method | Policy | Value | Performance(%) |
|---|---|---|---|
| MCTS-$\alpha$ | $\pi_{\theta_0}$ | $\{v,\hat{r}\}_{\phi_0}$ | 51.9 ± 0.6 |
|  | $\pi_{\theta_0}$ | $\{v,\hat{r}\}_{\phi_1}^{\text{RL}}$ | 52.0 ± 0.5 |
|  | $\pi_{\theta_0}$ | $\{v,\hat{r}\}_{\phi_1}$ | 53.2 ± 0.3 |
| MCTS-$\alpha$ | $\pi_{\theta_1}$ | $\{v,\hat{r}\}_{\phi_0}$ | 54.1 ± 0.9 |
| MCTS-$\alpha$ | $\pi_{\theta_1}$ | $\{v,\hat{r}\}_{\phi_1}^{\text{RL}}$ | 55.2 ± 1.2 |
| MCTS-$\alpha$ | $\pi_{\theta_1}$ | $\{v,\hat{r}\}_{\phi_1}$ | **56.5 ± 0.6** |

To figure out the best way of training the value function during iterative update. We also compare MCTS-$\alpha$ in Table 7. We train value and ORM in two paradigms, one ($\{v,\hat{r}\}_{\phi_1}$) is optimized from the initial weights and mixture of old and new tree-search data; another($\{v,\hat{r}\}_{\phi_1}^{\text{RL}}$) is optimized from $\{v,\hat{r}\}_{\phi_0}$ with only new tree-search data. This is called RL because training critic model in RL utilizes a similar process of continual training. The results show that $\{v,\hat{r}\}_{\phi_1}$ outperforms $\{v,\hat{r}\}_{\phi_1}^{\text{RL}}$ on both old and new policy when conducting tree search, contrary to the normal situation in traditional iterative RL training.

### C.2. Results of different node expansion on tasks

Table 8, Table 9 and Table 10 show the path@1 results of MCTS-$\alpha$, MCTS-Rollout, BFS-V under different numbers of *tree-max-width* $w$ on GSM8k, Game24 and ProntoQA. And we also show the results of CoT-SC-Tree$_{\text{ORM}}$@10 and CoT-SC$_{\text{ORM}}$@10, which are aggregated by ORM-vote. The results are conducted under 3 seeds and we show the average value and standard deviation.

Let us first clarify how we choose the specific tree max width. For *tree-max-width* $w$ in GSM8k, Game24, and ProntoQA, we first start with an initial value $w = 6$. Then by increasing it (10 in GSM8K, 20 in Game 24, and 10 in ProntoQA), we can see the trends in performance and computation consumption. In ProntoQA/GSM8K, the performance gain is quite limited while the performance gain is quite large in Game24. So at last, we in turn try smaller *tree-max-width* in GSM8K/ProntoQA (3) and also try even larger *tree-max-width* (50) in Game24. Our final choice of $w$ (in Table 1) is based on the trade-off between the performance and the computation consumption. Currently, our selection is mainly based on empirical trials and it might be inefficient to determine the appropriate *tree-max-width* $w$. We think this procedure can be more efficient and automatic by comparing it with the results of CoT-SC on multiple samples to balance the tradeoff between performance and computation consumption. Because CoT-SC examples can already provide us with information about the model generation variation and diversity. We can also leverage task-specific features, e.g. in Game24, the correctness of early steps is very important, so a large $w$ can help to select more correct paths from the first layers on the search trees.

For the analysis of the results in Table 8, Table 9 and Table 10. We can mainly draw two conclusions aligned with that in Q1/Q2 in the main paper.

First, the overall trend that larger search space represents better tree-search performance still holds. For most tree-search settings, larger *tree-max-width* $w$ and search space bring in performance gain. The only exception happens at MCTS-Rollout on GSM8k decreases when the *tree-max-width* $w = 10$, this is due to the limitation of computation(limitation on the number of generated tokens which is $51200$ per problem) is not enough in wider trees which results in more null answers. Despite the gain, the number of generated tokens also increases as the tree-max-width $w$ becomes larger.

Secondly, the conclusions in the main paper about comparing different search algorithms still hold. BFS performs pretty well in shallow search problems like (GSM8K/Game24). Though we can still see MCTS-$\alpha$ and MCTS-Rollout improve by searching in large *tree-max-width* (such as Game24 expanded by 50), the performance gain is mainly attributed to the extra token consumption and is quite limited. For deeper search problems like ProntoQA (15) and RLHF (64), the performance gap is obvious and more clear among all expansion widths. This aligns with our conclusion in Q1's analysis.

Table 8: Path@1 metric on GSM8k with different node size.

| Method | Performance(%) / # tokens | | | | | |
|---|---|---|---|---|---|---|
| | expand by 3 | | expand by 6 | | expand by 10 | |
| MCTS-$\alpha$ | $49.2 \pm 0.04$ | 460 | $51.9 \pm 0.6$ | 561 | $51.7 \pm 0.5$ | 824 |
| MCTS-Rollout | $47.2 \pm 0.8$ | 856 | $47.8 \pm 0.8$ | 3.4k | $45.9 \pm 0.9$ | 7.1k |
| BFS-V | $49.1 \pm 0.8$ | 260 | $52.5 \pm 1.3$ | 485 | $52.2 \pm 0.9$ | 778 |
| CoT-SC-Tree$_{\text{ORM}}$@10 | $52.4 \pm 1.2$ | 604 | $54.6 \pm 0.7$ | 780 | $54.5 \pm 1.1$ | 857 |
| CoT-SC$_{\text{ORM}}$@10 | - | - | - | - | $56.4 \pm 0.6$ | 1.0k |

Table 9: Path@1 metric on Game24 with different node size.

| Method | Performance(%) / # tokens | | | | | |
|---|---|---|---|---|---|---|
| | expand by 6 | | expand by 20 | | expand by 50 | |
| MCTS-$\alpha$ | $41.6 \pm 0.8$ | 243 | $63.3 \pm 1.9$ | 412 | $74.5 \pm 0.7$ | 573 |
| MCTS-Rollout | $43.8 \pm 5.3$ | 401 | $71.3 \pm 2.5$ | 670 | $80.7 \pm 1.5$ | 833 |
| BFS-V | $43.2 \pm 2.0$ | 206 | $64.8 \pm 2.9$ | 370 | $74.6 \pm 0.5$ | 528 |
| CoT-SC-Tree$_{\text{ORM}}$@10 | $38.8 \pm 2.0$ | 508 | $48.3 \pm 3.0$ | 656 | $48.3 \pm 4.2$ | 707 |
| CoT-SC$_{\text{ORM}}$@10 | - | - | - | - | $52.9 \pm 2.1$ | 0.8k |

Table 10: Path@1 metric on ProntoQA with different node size.

| Method | Performance(%) / # tokens | | | | | |
|---|---|---|---|---|---|---|
| | expand by 3 | | expand by 6 | | expand by 10 | |
| MCTS-$\alpha$ | $94.1 \pm 0.1$ | 151 | $99.4 \pm 0.2$ | 190 | $99.8 \pm 0.2$ | 225 |
| MCTS-Rollout | $85.9 \pm 0.8$ | 151 | $96.9 \pm 0.6$ | 210 | $99.3 \pm 0.4$ | 264 |
| BFS-V | $83.7 \pm 1.0$ | 105 | $94.4 \pm 0.3$ | 126 | $97.6 \pm 0.3$ | 145 |
| CoT-SC-Tree$_{\text{ORM}}$@10 | $91.9 \pm 0.8$ | 290 | $98.2 \pm 0.4$ | 417 | $99.1 \pm 0.1$ | 494 |
| CoT-SC$_{\text{ORM}}$@10 | - | - | - | - | $98.0 \pm 0.7$ | 0.9k |

## C.3. Wall-time and Engineering challenges

Table 11, Table 12 and Table 13 show the wall-time of running different tree search algorithms implemented in TS-LLM, searching for one answer per problem (i.e. path@1). We also show the wall-time of CoT greedy decoding and CoT-SC@10 with ORM aggregation as comparisons. We record the wall-time of inferencing over the total test dataset of each task. We show the overall wall-time and average wall-time per problem. The average wall-time was an estimated value since we ran the evaluation with 8 GPUs in parallel, $T_{\text{avg}} \simeq T_{\text{overall}} \div N_{\text{problem}} \times N_{\text{process}}$.)

The experiments were conducted on the same machine with 8 NVIDIA A800 GPUs, the CPU information is Intel(R) Xeon(R) Platinum 8336C CPU @ 2.30GHz.

We can find that the comparisons of wall-time within all the implemented tree-search algorithms in TS-LLM are consistent with those of the number of generated tokens. However, compared to CoT greedy decoding, the wall-time results of most tree search algorithms in TS-LLM are between two and three times of CoT greedy decoding's wall-time, excepting MCTS-Rollout runs for a very long time on GSM8k. And when comparing the wall-time and number of generated tokens between the tree-search methods and CoT-SC$_{\text{ORM}}$@10, TS-LLM is not as computationally efficient as CoT-SC decoding due to the complicated search procedures and extra computation introduced by calling value functions in the intermediate states.

There are two more things we want to clarify. Firstly, as we mentioned in Appendix 5.1, our current implementation only provides an algorithm prototype without specific engineering optimization. We find a lot of repeated computations are performed in our implementation when evaluating the child node's value. Overall, there still exists great potential to accelerate the tree-search process, which will be discussed in the next paragraph. We are continuously working on this (we will discuss the engineering challenges in the next paragraph). Secondly, this wall-time is just Path@1 result so they have different token consumptions, and DFS-V, BFS-V, and MCTS will degenerate into greedy value search as we mentioned before. We are also working on monitoring the time consumption for Path@N results so we can compare them when given the same scale of token consumption.

**Engineering challenges and potentials.** Here we present several engineering challenges and potentials to increase the tree-search efficiency.

- **Policy and Value LLM with the shared decoder.** Our current implementation utilizes separate policy and value decoders but a shared one might be a better choice for efficiency. If so, most extra computation brought by value evaluation can be reduced to simple MLP computation (from the additional value head) by reusing computation from LLM's policy rollout. It can largely increase the efficiency. We only need to care about the LLM's rollout computations under this setting.

- **KV cache and computation reuse.** KV cache is used in most LLM's inference processes such as the Huggingface transformer's (Wolf et al., 2020) generation function. It saves compute resources by caching and reusing previously calculated key-value pairs in self-attention. In the tree search problem, when expanding or evaluating a node, all preceding calculations for its ancestor nodes can be KV-cached and reused. However, because of the large state space of tree nodes, we cannot cache all node calculations since the GPU memory is limited and the communication between GPU/CPU is also inefficient (if we choose to store such cache in CPU). More engineering work is needed to handle the memory and time tradeoff. For instance, recent advancements like PagedAttention in vLLM (Kwon et al., 2023) and RadixAttention in SGLang (Zheng et al., 2023) offer potentials to solve this issue.

- **Large-batch vectorization.** Currently, our node expansion and node evaluation are only vectorized and batched given one parent node. We may conduct batch inference over multiple parent nodes for large-batch vectorization when given enough computing resources.

- **Parallel tree-search over multi-GPUs.** Our implementation handles each tree over a single GPU. AlphaZero (Silver et al., 2017) leverages parallel search over the tree (Segal, 2010), using multi-thread search to increase efficiency. In LLM generation setting, the main bottleneck comes from the LLM inference time on GPU. Thus more engineering work is needed for conducting parallel tree-search over multi-GPUs.

- **Tree-Search with speculative decoding** Speculative decoding (Leviathan et al., 2023) is a pivotal technique to accelerate LLM inference by employing a smaller draft model to predict the target model's outputs. During the speculative decoding, the small LLM gives a generation proposal while the large LLM is used to evaluate and determine whether to accept or reject the proposal. This is similar to the tree-search process with value function pruning the sub-tree. There exists potential that by leveraging small LLMs as the rollout policy while large LLMs as the value function, we can also have efficient tree-search implementations.

Table 11: Wall-time results on GSM8k

| Method | Overall Time(sec) | Average Time(sec) | #Average Token |
|---|---|---|---|
| CoT-Greedy | 216.93 | 1.32 | 98 |
| CoT-SC$_{\text{ORM}}$@10 | 479.03 | 2.91 | 1k |
| BFS-V | 383.08 | 2.32 | 485 |
| MCTS-$\alpha$ | 527.31 | 3.20 | 561 |
| MCTS-Rollout | 2945.94 | 17.87 | 3.4k |

Table 12: Wall-time results on Game24

| Method | Overall Time(sec) | Average Time(sec) | #Average Token |
|---|---|---|---|
| CoT-Greedy | 44.88 | 0.99 | 76 |
| CoT-SC$_{\text{ORM}}$@10 | 86.53 | 1.91 | 0.8k |
| BFS-V | 79.48 | 1.76 | 369 |
| MCTS-$\alpha$ | 134.19 | 2.97 | 412 |
| MCTS-Rollout | 193.18 | 4.27 | 670 |

Table 13: Wall-time results on ProntoQA

| Method | Overall Time(sec) | Average Time(sec) | #Average Token |
|---|---|---|---|
| CoT-Greedy | 74.09 | 1.19 | 77 |
| CoT-SC$_{\text{ORM}}$@10 | 218.12 | 3.49 | 0.8k |
| BFS-V | 130.35 | 2.09 | 126 |
| MCTS-$\alpha$ | 236.26 | 3.78 | 190 |
| MCTS-Rollout | 238.62 | 3.82 | 210 |

## C.4. Discussion about Shared LLM decoder for both policy and critic.

As we mentioned in Appendix C.3, using a shared decoder for the policy and value LLM might further improve the computation efficiency for the tree-search process. Therefore, we conducted an ablation to compare the model under the settings of a *shared decoder* and the setting of *separated decoder*s on Game24.

We first describe the training setting of both types of models we compared. For the setting of *separated decoder*, we refer to Appendix D.2 for details about dataset and training hyperparameters. For the setting of *shared decoder*, we train the shared policy and value LLM with the same data used in the setting of *separated decoder*. During training, a batch from the supervised finetuning (SFT) dataset and a batch from the value training dataset are sampled, the total loss of the shared policy and value LLM is computed by $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{SFT}} + 0.5 \cdot \mathcal{L}_{\text{Value}}$, where $\mathcal{L}_{\text{SFT}}$ is the cross entropy loss of predicting the next token in the groundtruth answer and $\mathcal{L}_{\text{Value}}$ is the Mean Square Error loss as we described in Equation 1. The training is conducted on 8 NVIDIA A800 GPUs, using a cosine scheduler decaying from lr=2e-5 to 0.0 with a warmup ratio of 0.03, a batch size of 128 for the supervised finetuning dataset and a batch size of 128 for the value training dataset. And we trained the shared decoder model for 3 epochs. This training setting is the same as used in the *separated decoder* setting.

We will compare these two types of model in the perspective of performance and computation efficiency. All tree-search algorithms are conducted under the same hyperparameters as those in Table 3 on Game24 in which the tree-max-width $w$ is set to 20.

Table 14 shows the comparisons of the two types of model on performance. Though the performance of CoT (CoT greedy decoding) of *shared decoder* model increases from 12.7 to 16.3, the number of tokens generated per problem also increases greatly from 76 to 166. By checking the models' outputs, we find the *shared decoder* model doesn't always obey the rules of Game24(There are only 4 steps of calculations and each number must be used exactly once). It usually outputs multiple steps, more than the four steps required in Game24. This might be regarded as hallucination problem which happens more frequently than in the *separated decoder* model. For the results of CoT-SC-Tree$_{\text{ORM}}$@10 (search by LLM's prior on trees and aggregated by ORM-vote), we observe close results of CoT-SC$_{\text{ORM}}$@10 Meanwhile, for the results of MCTS-$\alpha$, MCTS-Rollut, BFS-V and CoT-SC-Tree$_{\text{ORM}}$@10, there is only a small difference between the performance of the two models. We also observe an increase in the number of token generated per problem. This is because the *shared decoder* model is prone to output more invalid answers than the *separated decoder* model. Therefore, there are more distinct actions proposed in the last layers of the trees.

Next, we show some preliminary comparisons in Table 15 on the computation efficiency of the *separated decoder* model and the *shared decoder* model, from the results of expanding a tree node and evaluating its children. Specifically, Table 15 presents the node expansion time and value calculation time with/without KV Cache under token-level and sentence-level situations. For the token-level node, we set $w = 50$ while for the sentence-level node, we set $w = 20$. The results successfully present that a shared decoder can largely increase the computational efficiency for value estimation (20x in the token-level setting and 9x in the sentence-level setting).

In all, in this section, we initially conduct explorations on leveraging shared policy and value LLM decoder. The result proves the potential of computational efficiency for the shared structure. However, more work is needed to help the stability of policy/value performance.

Table 14: Comparisions of separated/shared LLM decoder policy and critic models on Game24

| Method | Performance(%) / # tokens | | | |
|---|---|---|---|---|
| | Separated Decoder | | Shared Decoder | |
| CoT | 12.7 | 76 | 16.3 | 166 |
| CoT-SC$_{\text{ORM}}$@10 | $52.9 \pm 2.1$ | 0.8k | $52.8 \pm 2.4$ | 1.9k |
| MCTS-$\alpha$ | $63.3 \pm 1.9$ | 412 | $64.1 \pm 1.3$ | 561 |
| MCTS-Rollout | $71.3 \pm 2.5$ | 670 | $70.6 \pm 0.4$ | 855 |
| BFS-V | $64.8 \pm 2.9$ | 370 | $63.0 \pm 1.0$ | 495 |
| CoT-SC-Tree$_{\text{ORM}}$@10 | $48.3 \pm 3.0$ | 656 | $45.5 \pm 2.0$ | 745 |

Table 15: Time (seconds) on policy expansion and value evaluation for a single tree node. When using a shared LLM decoder for policy and value LLM, we can use KV Cache for value calculation. It is much more efficient than a separate value decoder without KV cache.

| Node Type | Policy Expansion | Value with Cache | Value without Cache |
|---|---|---|---|
| Token-Level | 0.067 | 0.074 | 2.02 |
| Sentence-Level | 0.165 | 0.122 | 1.03 |

## D. Experiment Details

### D.1. Task setups

**GSM8k** GSM8k (Cobbe et al., 2021) is a commonly used numerical reasoning dataset, Given a context description and a question, it takes steps of mathematical reasoning and computation to arrive at a final answer. There are about 7.5k problems in the training dataset and 1.3k problems in the test dataset.
**Game24** We also test our methods on Game24 (Yao et al., 2023) which has been proven to be hard even for state-of-the-art LLMs like GPT-4. Each problem in Game24 consists of 4 integers between 1 and 13. And LLMs are required to use each number exactly once by $(+ - \times \div)$ to get a result equal to 24. We follow Yao et al. (2023) by using a set of 1362 problems sorted from easy to hard according to human solving time. We split the first 1k problems as the training dataset and the last 362 hard problems as the test dataset. For each problem in the training dataset, we collect data for SFT by enumerating all possible correct answers.
**PrOntoQA** PrOntoQA (Saparov & He, 2022) is a typical logical reasoning task in which a language model is required

to verify whether a hypothesis is true or false given a set of facts and logical rules. There are 4k problems in the training dataset and 500 problems in the test dataset.
**RLHF** We choose a synthetic RLHF dataset (Dahoas)[1] serving as the query data. We split the dataset to 30000/3000 as training and test set respectively. For the reward model, we choose reward-model-deberta-v3-large-v2[2] from OpenAssistant, which is trained from several RLHF datasets.
**Chess Endgame** Chess Endgame is introduced by Abdulhai et al. (2023). Chess endgames provide a simpler and more goaldirected variation of the chess task. A classic theoretical endgame position consists of a position where the only pieces on the board are the two kings and the queen. Although the board position appears simple, a sequence of carefully calculated moves is required to win. We followed the setting of Abdulhai et al. (2023) use an opponent of StockFish whose Elo is 1200. We modified the environment dynamics so that the agent fails if it makes an illegal move.

### D.2. SFT and value training details

**SFT in GSM8k, Game24 and PrOntoQA**: For GSM8k, Game24 and PrOntoQA, we first train LLaMA2-7b on the training dataset The training is conducted on 8 NVIDIA A800 GPUs, using a cosine scheduler decaying from lr=2e-5 to 0.0 with a warmup ratio of 0.03, batch size 128 for 3 epochs. For GSM8k and Game24 we use the checkpoint at the last epoch as the direct policy in experiments, while for PrOntoQA we use the checkpoint at the 1st epoch since the others overfit. In PrOntoQA, since we found that the language model policy can easily achieve 100% accuracy when supervised finetuning on all the 4000 training examples, we only train the LLaMA2-7b model on a subset of 1000 problems.

**Value training in GSM8k, Game24 and PrOntoQA**: Then we train the value function on the data rollout by the SFT policy. In GSM8k and Game24, For each model checkpoints of 3 epochs during SFT, we first collect 100 outputs per problem in the training dataset, then duplicate the overlapped answers, labeled each answer with our training set outcome reward ocracle. For data sampled by ech model checkpoint, we subsample 17 answers per problem, which is in total at most 51 answers per problem after deduplication. In PrOntoQA, we only sample 50 answers per problem with the first epoch model checkpoint and then do deduplication. We show the wall-time and total number of generated tokens of sampling rollout data for one policy of these three benchmarks on Table 16.

Table 16: Cost of sampling rollout data of one policy model in GSM8k, Game24 and ProntoQA

| Task | #Problems | #Rollout Samples | Wall-time(minute) | #Total Tokens |
|---|---|---|---|---|
| GSM8k | 7.4k | 50 | 84.9 | 36.7M |
| Game24 | 1.0k | 50 | 6.6 | 3.9M |
| ProntoQA | 1.0k | 50 | 10.5 | 4.5M |

The value functions are trained in the same setting as supervised finetuning. We set the reward to be 1 when the output answer is correct and -1 otherwise. Then we use MC with $\gamma = 1$ to compute the returns. We do model selection on a validation dataset sampled from the direct policy model. For GSM8k, we train the value function and ORM for one epoch, while for Game24 and PrOntoQA we train the value function and ORM for 3 epochs.

**SFT in RLHF alignment**: We utilize GPT2-open-instruct[3], a GPT2-Small model supervised-finetuned over several instruction-tuning dataset.

**Value training in RLHF alignment**: Based on the SFT model, we collect 50 rollouts by the SFT policy for each question in the training set and label their final reward with the reward model. Then we train the value function and ORM for 2 epochs.

Note that here we start training the value function and ORM from the data sampled by the SFT policy model through direct decoding just as an initialization of the value function and ORM. After that TS-LLM can optimize the policy model, value function, and ORM simultaneously by adding new data sampled from tree search into the training buffer.

**SFT in Chess Endgame**: Currently we use the opensourced behavior-cloning model in Abdulhai et al. (2023) as the SFT model[4]. Given a Forsyth-Edwards Notation (FEN) state description, the language model policy autoregressively outputs

[1]https://huggingface.co/datasets/Dahoas/synthetic-instruct-gptj-pairwise
[2]https://huggingface.co/OpenAssistant/reward-model-deberta-v3-large-v2
[3]https://huggingface.co/vicgalle/gpt2-open-instruct-v1
[4]https://github.com/abdulhaim/LMRL-Gym

SAN(Standard Algebraic Notation)-format actions.

**Value training in Chess Endgame** Currently we use the opensourced critic value function model trained by Proximal Policy Optimization (PPO) in Abdulhai et al. (2023). The language model value function receives a FEN state description and outputs a scalar as the value estimation.

### D.3. Details of comparing prompting-based value function and learned value function

For the prompting-based value function presented in Table 2, we use GPT-3.5-0613 for GPT3.5 model and design few-shot prompts for GPT3.5-policy/value and also LLaMA2-7B. We sample 1 time from GPT3.5 and 3 times from LLaMA2-7B to form the value evaluation.

### D.4. Details of value dataset ablation

Here we introduce the details of building *mixed*, *pure* and *pure,less* datasets on GSM8k for value training in Table 5. For each model checkpoints of 3 epochs during SFT, we first collect 100 outputs per problem in GSM8k training dataset, and then duplicate the overlapped answers, labeled each answer with our training set outcome reward ocracle. we sample multiple output sequences with temperature=0.7, top p=1.0 and top k=100.
***mixed*** dataset: For each deduplicated dataset sampled by models of 3 epochs, we subsample 17 answers per problem.
***pure*** dataset: we subsample 50 answers per problem from deduplicated dataset sampled by the last epoch policy model.
***pure,less*** dataset: we subsample 17 answers per problem from deduplicated dataset sampled by the last epoch policy model.

For the results in Table 7, the details of training $\{v, \hat{r}\}_{\phi_1}$ can be find in Sec D.9. We use MC with $\gamma = 1$ to compute the returns. Here we describe the details of training $\{v, \hat{r}\}_{\phi_1}^{\text{RL}}$, we use the collected 78.7k samples in Sec D.8 to optimize $\{v, \hat{r}\}_{\phi_0}$. The training uses a cosine scheduler decaying from lr=2e-5 to 0.0 with a warmup ratio of 0.03, batch size 128 for 3 epochs.

### D.5. Hyperparameter Selection Protocols

Here we present the selection protocols of hyperparameters.

**Tree search general hyperparameters**: the *tree-max-depth* $d$ limits the search depth of tree and *tree-max-width* $w$ controls the max number of child nodes during node expansion. For the *tree-max-width* $w$, we refer the reader to Appendix C.2 for more discussions. We choose *tree-max-depth* $d$ according to the statistics of the distribution of the number of steps from the dataset sampled by the LLM policy on the training set. Specifically, we statistically analyzed the number distribution of sentences in the training set, and in our experiments, these sentences are split by '`\n`'. For GSM8K, we set the *tree-max-depth* $d$ at around the 99th percentile of the entire number distribution, to cover most query input and drop the outliers. Game24 has a fixed search depth of 4. For ProntoQA, we set the *tree-max-depth* $d$ at the upper bound of the entire number distribution. For Chess Endgame, since endgame usually ends in a small number of moves, so we set the tree-max-depth to be a maximal value of 50. For RLHF, this is not a reasoning task with CoT steps, so the depth can be flexible. We set it as the default value. In most cases, the depth of *tree-max-depth* $d$ will not be reached. Because the node expansion will be terminated when we detect our pre-defined stop words in the generation (such as '`The answer is`' or the '`<EOS>`' token).

**Specific hyperparameters for Monte Carlo Tree Search variants**: Basically we adopted the default values from Schrittwieser et al. (2020b) and Silver et al. (2017) for most of the hyperparameters, such as $c_{\text{base}} = 19652$ in Equation 4, $\tau = 1.0$ for MCTS-alpha stochastic search. And for the Dirichlet noise of MCTS-$\alpha$ stochastic search as mentioned in Appendix D.7, we adopted the default value in Silver et al. (2017) as 0.3, which is specified for chess. We do find that in MCTS-$\alpha$, MCTS-Rollout and MCTS, $c_{\text{init}}$ can affect the balance between exploration and exploitation, and we chose it by running several trials from two possible values: $\{0.3, 3.0\}$. Moreover, for MCTS-$\alpha$, the hyperparameter *num of simulation*, $n_{\text{simulation}}$ is chosen as 5 for shallow trees (tree max-depths less than of equal to 15 over GSM8k, Game24 and ProntoQA) and 10 in deep trees(a tree max-depth of 64 in RLHF), controlling the search complexity at each step.

**Specific hyperparameters for BFS-/DFS-V**: BFS-V does not have hyperparameters for single search. For DFS-V, the children of a non-leaf node is traversed in a non-decreasing order by its value. For the sake of efficient exploration, we tried 2 heuristics to prune the subtrees, (1) drop the children nodes with lower values by *prune_ratio*. (2) drop the children nodes lower than a *prune_value*. The latter is adopted from Yao et al. (2023). In our experiments, we tried possible *prune_values* from $\{0.5, 0.0, -0.5\}$ or `None`, we found that setting a high *prune_value* like 0.5 or 0.0 may introduce significant performance drop, however, setting a higher *prune_value* may introduce very closer answers. Therefore, we finally use *prune_ratio* for efficient exploration during searching on the tree with DFS-V. We set *prune_ratio* to be 0.7 (selected from $\{0.3, 0.5, 0.7\}$) for GSM8k, Game24 and PrOntoQA (tree-max-widths of 6, 6, 20), and 0.95 (selected from $\{0.5, 0.7, 0.9, 0.9\}$) for RLHF alignment task since its much wider(a tree-max-width of 50).

### D.6. Details of applying each tree search approach

We present the implementation details and hyperparameters of all tree search approaches here.

Firstly, we refer to Table 1 for basic settings of each task. We set temperature=1.0, top_p=1.0, top_k=100 when using LLM to generate tree actions. To compute logprobs when expand actions in RLHF alignment task trees, we also use a temperature of 1.0.

For MCTS variants including MCTS-$\alpha$, MCTS-Rollout and MCTS, we need to define the hyperparamter in PUCT algorithm:

$$c_{\text{puct}} = \log((\sum_b N(s,b) + c_{\text{base}} + 1)/c_{\text{base}}) + c_{\text{init}} \tag{4}$$

In this paper, we fixed $c_{\text{base}}$ with 19652 and set $c_{\text{init}} = 3$ for GSM8k, Game24 and RLHF alignment tasks, set $c_{\text{init}} = 0.3$ for PrOntoQA tasks. For Chess Endgame, we manually set $c_{\text{puct}} = 0$. Specifically, in MCTS-$\alpha$, we set the number of simulations before making an action to 5 for GSM8k, Game24 and PrOntoQA, 10 for RLHF alignment and Chess Endgame. During evaluation, we deterministically sampled actions with the largest visit count. In MCTS-Rollout, we set an computation upperbound as number of generated tokens or number of model forwards, which is 51200 in GSM8k and Game24, 1800 in PrOntoQA and 5000 in RLHF alignment, 10000 in Chess Endgame.

For DFS-V, the children of a non-leaf node is traversed in a non-decreasing order by value. For efficient exploration, we tried 2 heuristics to prune the subtrees, (1) drop the children nodes with low value by *prune_ratio*. (2) drop the children nodes lower than a *prune_value*. We set *prune_ratio* to be 0.7 for GSM8k, Game24 and PrOntoQA, and 0.95 for RLHF alignment task, 0.6 for Chess Endgame.

All Path@1 results for each tree search approach is conducted with 3 seeds and show the mean and standard deviation. Note that for Path@1 results of tree search approaches with sentence-level nodes, the randomness comes from the node expansion process where we use an LLM to sample candidate actions. While for CoT-SC results, the randomness comes from sampling during direct decoding.

### D.7. Details of aggregation experiments

Another alternative setting for conducting multiple searches in **Inter-tree Search**. Inter-tree Search builds a new tree for each new search, which increases the diversity of the search space, with extra computation burdens proportional to the search times. Thus, the intra-setting will have a larger state space compared with intra-tree setting. Our experiment results shown in Fig 2 (comparing MCTS-$\alpha$ intra-tree and inter-tree settings) also verify the performance gain brought by the larger search space.

We also present the details of how we sample multiple answers with tree search approaches and aggregate them into a final answer.

For the results of CoT-SC-Tree on Table 3 and Table 6, they can be viewed as **intra-tree** searches. For the results in Figure 2, only *MCTS-$\alpha$ inter-trees* were conducted with **inter-tree** searches, other tree-search algorithms (MCTS, MCTS-Rollout, BFS-V, DFS-V) were all conducted with **intra-tree** searches

Note that for all tree search algorithms except BFS-V, multiple searches are conducted in a sequential manner, while for BFS-V which can actually be regarded as Beam-Search, the number of searches means the number of beam size.

When sampling multiple intra-tree answers with MCTS-$\alpha$, we use a stochastic sampling setting. To ensure MCTS-$\alpha$ to explore suffciently, when selecting action of the current node, before doing several times of simulation, we add Dirichlet noise into the language model's prior probability of the current root node $s_0$, i.e. $\pi'_\theta(s_0, a) = (1-\epsilon)\pi_\theta(s_0, a) + \epsilon\eta$, where $\eta \sim \text{Dir}(0.3)$, and we set $\epsilon = 0.25$ for both tasks. Actions are sample based on visit count, i.e. $a \sim \frac{N(s_t,a)^{1/\tau}}{\sum_b N(s_t,b)^{1/\tau}}$, where we set $\tau = 1$. After returning with a complete path, we clear the node statistics ($Q(s_t, a_t)$ and $N(s_t, a_t)$) on the tree to eliminate the influence of previous searches, while the tree structure is maintained. This setting is denoted as clear-tree when presented in the table. **In summary, when we only measure path@1 performance, we adopt MCTS-$\alpha$ (no sampling). But when we measure the aggregation performance, we use MCTS-$\alpha$-intra tree or MCTS-$\alpha$-inter tree. In MCTS-$\alpha$-intra tree we will activate the clear-tree and stochastic sampling setting.**

When sampling multiple answers with other tree-search methods, we only utilize the intra-tree aggregation variant, without stochastic sampling and clear-tree setting. This is because only MCTS-$\alpha$ and MCTS-rollout can conduct the above sampling and clear-tree operation. And we temporarily only apply such setting on MCTS-$\alpha$.

In ORM-vote, since we train the ORM with the reward signal -1 and 1, given a list of $N$ answers to be aggregated, we first normalize its values $\{\hat{r}(y^j)\}_j$ with min-max normalization to make them in $[0, 1]$.

## D.8. sampling details of iterative update

We verify the idea of iteratively enhancing language model policy and value function model on the GSM8k and RLHF datasets.

**Sampling in GSM8k**: When sampling from the 7.5k problems in the GSM8k training dataset, we sample 12 sequences per problem in one sentence-level expanded tree, after deduplication, this results in 78.7k distinct answers, and 73.2% are correct answers. The sample parameters are listed in Table 17.

**Sampling in RLHF alignment**: We collect 10 answers for each training set problem sampled by MCTS-$\alpha$. We list the specific hyperparameters in Table 18.

To collect data for the rejection sampling baseline, we first sample 10 sequences per problem and then use the top 5 sequences for supervised fine-tuning.

Table 17: Hyperparameters of sampling in GSM8k for LLM decoding(left), tree construction setting(middle), and MCTS-$\alpha$ setting(right).

| Hyperparameter | value |
|---|---|
| temperature | 1.0 |
| top_p | 1.0 |
| top_k | 100 |

| Hyperparameter | value |
|---|---|
| Tree Max width | 6 |
| Tree Max depth | 8 |
| Node | Sentence |

| Hyperparameter | value |
|---|---|
| num simulation | 5 |
| clear tree | True |
| stochastic sampling | True |
| $c_{\text{base}}$ | 19652 |
| $c_{\text{init}}$ | 3 |
| $\tau$ | 1.0 |

Table 18: Hyperparameters of sampling in RLHF alignment for LLM decoding(left), tree construction setting(middle), and MCTS-$\alpha$ setting(right).

| Hyperparameter | value |
|---|---|
| temperature | 1.0 |
| top_p | 1.0 |
| top_k | 50 |

| Hyperparameter | value |
|---|---|
| Tree Max width | 50 |
| Tree Max depth | 64 |
| Node | Token |

| Hyperparameter | value |
|---|---|
| num simulation | 5 |
| clear tree | True |
| stochastic sampling | True |
| $c_{\text{base}}$ | 19652 |
| $c_{\text{init}}$ | 3 |
| $\tau$ | 1.0 |

## D.9. Training details of iterative update

**Policy training in GSM8k**: We construct the dataset for supervised finetuning by combining data in the training dataset with 57.6k correct answers sampled in Sec D.8 which results in 64.1k distinct correct answers. And we train the new policy model $\pi_{\theta_1}$ from the starting base model LLaMA2-7b for 3 epochs, following Yuan et al. (2023a). The training setting is the same as described in Sec D.2.

**Value training in GSM8k**: We construct the dataset for value and ORM training by combining the data used to train $\{v, \hat{r}\}_{\phi_0}$ with 78.7k answers sampled by MCTS-$\alpha$ in Sec D.8. To fairly compare $\{v, \hat{r}\}_{\phi_1}$ with $\{v, \hat{r}\}_{\phi_0}$, we drop samples in the former dataset to keep at most $51 - 12 = 39$ answers per problem resulting in 359k distinct answers. And we train the new value function $\{v, \hat{r}\}_{\phi_1}$ from the value model with its initial weight(before being updated on any data) for 3 epochs. The training setting is the same as described in Sec D.2.

**Policy training in RLHF alignment**: For the MCTS-$\alpha$'s training, we subsample the top 5 answers from the full 10 candidates (mentioned in Appendix D.8) to serve as the SFT dataset. For the RFT n=5 baseline, we subsample the top 5 answers from 50 direct decodings as the SFT dataset. For the training of the PPO algorithm, we adopt the implementation from trlx[5]. We sample 20 answers for each question in total, which maintains the same level of token consumption during the PPO rollouts as that of MCTS-$\alpha$.

**Value training in RLHF alignment**: We construct the value and ORM dataset by mixing data from SFT-policy direct decoding and from MCTS-$\alpha$. To make the comparison fair, the new value function's training utilizes the same amount of data as the old one by subsampling 40 answers (from 50 shown in Appendix D.2) from direct decoding data and all 10 answers (shown in Appendix D.8) generated by MCTS-$\alpha$. We train our value function with learning rate 2e-5 and cosine scheduler from the initial model (instead of continuing training from the old value function) for 2 epochs.

## E. Qualitative Results

Here we present some qualitative results of generation for different methods in Table 19 (for Game24), Table 20 (for GSM8k), Table 21 (for ProntoQA) and 22 (for the RLHF alignment task).

Problem: 4 8 9 13

| Methods | Response | Correctness |
|---|---|---|
| CoT | 8 + 13 = 21 (left: 4 9 21)<br>4 - 9 = -5 (left: -5 21)<br>21 - -5 = 26 (left: 26)<br>The answer is (8 + 13) - (4 - 9) = 26 | No |
| BFS-V | 9 + 3 = 12 (left: 4 8 12)<br>12 / 4 = 3 (left: 3 8)<br>3 * 8 = 24 (left: 24)<br>The answer is (9 + 3) / 4 * 8 = 24 | No |
| MCTS-$\alpha$ | 13 - 9 = 4 (left: 4 4 8)<br>4 * 4 = 16 (left: 8 16)<br>16 + 8 = 24 (left: 24)<br>The answer is 4 * (13 - 9) + 8 = 24 | Yes |
| MCTS-Rollout | 13 - 9 = 4 (left: 4 4 8)<br>4 * 4 = 16 (left: 8 16)<br>16 + 8 = 24 (left: 24)<br>The answer is (13 - 9) * 4 + 8 = 24 | Yes |

Table 19: Qualitative examples of Game24 outputs

## F. Detailed Experiment Results

### F.1. Results per task per aggregation

We show detailed results of GSM8k on Table 23, results of Game24 on Table 24, Table 25 for PrOnToQA and Table 26 for RLHF alignment. Due to the limit of computation resources, we show the results under 1 seed except for the path@1 results.

[5]https://github.com/CarperAI/trlx

| Question: A new program had 60 downloads in the first month. The number of downloads in the second month was three times as many as the downloads in the first month, but then reduced by 30% in the third month. How many downloads did the program have total over the three months? | | |
|---|---|---|
| Methods | Response | Correct |
| CoT | The number of downloads in the second month was 3*60 = 180<br>The number of downloads in the third month was 180*0.30 = 54<br>The total number of downloads for the three months was 60+180+54 = 294<br>The answer is 294 | No |
| BFS-V | In the second month, the program had 60*3=180 downloads<br>In the third month, the program had 180*30/100=54 downloads which was 30% less than the previous month<br>Total downloads for the program in the three months was 60+180+54=294<br>The answer is 294 | No |
| MCTS-$\alpha$ | The program had 60 x 3 = 180 downloads in the second month.<br>The downloads in the third month was 180 x 0.30 = 54 downloads lesser than the previous month.<br>So, the program had 180 - 54 = 126 downloads in the third month.<br>Therefore, the program had 60 + 180 + 126 = 366 downloads over the three months.<br>The answer is 366 | Yes |
| MCTS-Rollout | The number of downloads in the second month was 3 times as many as the downloads in the first month, which means the program had 3*60 = 180 downloads in the second month.<br>The total number of downloads for the two months was 180+60 = 240 In the third month, the program had 180-30/100*180 = 120 downloads, 30% of the number of downloads it had in the second month.<br>The total number of downloads for the three months was 240+120 = 360<br>The answer is 360 | No |

Table 20: Qualitative examples of GSM8k outputs

| Question: Butterflies are lepidopterans. Every arthropod is small. Whales are not small. Invertebrates are animals. Every insect is an arthropod. Lepidopterans are insects. Every insect is six-legged. Every arthropod is an invertebrate. Animals are multicellular. Polly is a lepidopteran. Is the statement "Polly is not small" true or false? | | |
|---|---|---|
| Methods | Response | Correct |
| CoT | Polly is a lepidopteran.<br>Lepidopterans are insects.<br>Polly is an insect.<br>Every insect is an arthropod.<br>Polly is an arthropod.<br>Every arthropod is small.<br>Polly is small.<br>The answer is true. | No |
| BFS-V | Polly is a lepidopteran.<br>Lepidopterans are insects.<br>Polly is an insect.<br>Every insect is an arthropod.<br>Polly is an arthropod.<br>Every arthropod is small.<br>Polly is small.<br>Polly is small.<br>The answer is false. | Yes |
| MCTS-$\alpha$ | Polly is a lepidopteran.<br>Lepidopterans are insects.<br>Polly is an insect.<br>Every insect is an arthropod.<br>Polly is an arthropod.<br>Every arthropod is small.<br>Polly is small.<br>The answer is false. | Yes |
| MCTS-Rollout | Polly is a lepidopteran.<br>Lepidopterans are insects.<br>Polly is an insect.<br>Every insect is an arthropod.<br>Polly is an arthropod.<br>Every arthropod is small.<br>Polly is small.<br>The answer is false. | Yes |

Table 21: Qualitative examples of ProntoQA outputs

| Prompt: What careers am I good at. | | |
|---|---|---|
| Methods | Response | Reward |
| CoT | You can be a great career at any career you choose. Depending on your interests and skills, you may be able to find a variety of jobs that align with your interests and skills. | -2.625 |
| BFS-V | Some careers great careers to do professionally are entrepreneur in healthcare policy policy advocacy consulting public diplomacy leadership policy innovation in global issues managing information theory. Some great careers to achieve excellence in healthcare management include doctor psychologist | -2.03 |
| MCTS-$\alpha$ | You may be good at a variety of careers, depending on your interests and skills. Some of the most popular careers include software engineer, data scientist. You may also be good at a variety of fields such as finance, marketing, and business administration. | -1.67 |

Table 22: Qualitative examples of RLHF outputs

Table 23: Detailed Results in GSM8k

| Method | N | Majority-vote | ORM-vote | ORM-max | #Token |
|---|---|---|---|---|---|
| CoT | - | 41.4 | 41.4 | 41.4 | 0.1k |
| CoT-SC | 1 | 38.21 | 38.21 | 38.21 | 0.1k |
| CoT-SC | 10 | 51.93 | 57.47 | 53.83 | 1k |
| CoT-SC | 20 | 54.44 | 59.44 | 54.74 | 2k |
| CoT-SC | 50 | 56.79 | 61.03 | 54.44 | 5k |
| CoT-SC | 100 | 58.15 | 62.70 | 53.68 | 10k |
| CoT-SC-Tree | 1 | 37.91 | 37.91 | 37.91 | 0.1k |
| CoT-SC-Tree | 10 | 50.19 | 53.15 | 50.95 | 0.8k |
| CoT-SC-Tree | 20 | 52.69 | 55.12 | 52.84 | 1.3k |
| CoT-SC-Tree | 50 | 54.51 | 57.16 | 53.45 | 2.7k |
| BFS-V | 1 | 52.5 | 52.5 | 52.5 | 0.5k |
| BFS-V | 10 | 58.98 | 56.25 | 54.97 | 3.1k |
| BFS-V | 20 | 58.91 | 56.79 | 52.39 | 5.3k |
| BFS-V | 50 | 59.29 | 59.36 | 53.22 | 10.1k |
| DFS-V | 1 | 51.8 | 51.8 | 51.8 | 0.5k |
| DFS-V | 10 | 57.09 | 56.18 | 54.89 | 1.2k |
| DFS-V | 20 | 58.23 | 58.38 | 55.19 | 1.6k |
| DFS-V | 50 | 58.98 | 58.98 | 55.35 | 2.1k |
| MCTS-$\alpha$ (no sampling) | 1 | 51.9 | 51.9 | 51.9 | 0.5k |
| MCTS-$\alpha$-intra tree | 1 | 46.78 | 46.78 | 46.78 | 0.7k |
| MCTS-$\alpha$-intra tree | 10 | 57.85 | 56.86 | 54.36 | 3.4k |
| MCTS-$\alpha$-intra tree | 20 | 58.83 | 58.23 | 55.19 | 5.3k |
| MCTS-$\alpha$-inter trees | 1 | 51.9 | 51.9 | 51.9 | 0.5k |
| MCTS-$\alpha$-inter trees | 10 | 57.92 | 58.53 | 55.34 | 5.5k |
| MCTS-$\alpha$-inter trees | 20 | 58.83 | 59.06 | 54.97 | 11.1k |
| MCTS-$\alpha$-inter trees | 50 | 58.76 | 61.26 | 53.98 | 27.8k |
| MCTS | 1 | 52.2 | 52.2 | 52.2 | 0.5k |
| MCTS | 10 | 57.92 | 55.72 | 53.75 | 2.4k |
| MCTS | 20 | 58.61 | 56.79 | 54.74 | 4.0k |
| MCTS | 50 | 59.36 | 58.23 | 53.75 | 7.5k |
| MCTS-Rollout | 1 | 47.8 | 47.8 | 47.8 | 3.4k |
| MCTS-Rollout | 10 | 51.10 | 50.49 | 49.81, | 5.4k |
| MCTS-Rollout | 20 | 51.86 | 51.25 | 50.19 | 6.1k |
| MCTS-Rollout | 50 | 52.69 | 52.24 | 50.49 | 7.2k |

Table 24: Detailed Results in Game24

| Method | N | Majority-vote | ORM-vote | ORM-max | #Token |
|---|---|---|---|---|---|
| CoT | - | 12.7 | 12.7 | 12.7 | 0.1k |
| CoT-SC | 1 | 9.94 | 9.94 | 9.94 | 0.1k |
| CoT-SC | 10 | 13.54 | 50.83 | 50.83 | 0.8k |
| CoT-SC | 20 | 14.36 | 65.75 | 65.47 | 1.6k |
| CoT-SC | 50 | 16.30 | 78.45 | 78.45 | 4.0k |
| CoT-SC | 100 | 18.23 | 84.25 | 84.53 | 7.9k |
| CoT-SC-Tree | 1 | 9.67 | 9.67 | 9.67 | 0.1k |
| CoT-SC-Tree | 10 | 11.33 | 48.34 | 48.34 | 0.7k |
| CoT-SC-Tree | 20 | 13.26 | 61.60 | 62.15 | 1.1k |
| CoT-SC-Tree | 50 | 16.57 | 69.61 | 69.89 | 2.0k |
| BFS-V | 1 | 64.8 | 64.8 | 64.8 | 0.4k |
| BFS-V | 10 | 47.79 | 70.72 | 70.99 | 1.6k |
| BFS-V | 20 | 27.62 | 69.34 | 69.34 | 2.3k |
| BFS-V | 50 | 7.18 | 70.17 | 70.72 | 3.7k |
| DFS-V | 1 | 66.3 | 66.3 | 66.3 | 0.4k |
| DFS-V | 10 | 55.25 | 69.06 | 69.34 | 0.9k |
| DFS-V | 20 | 54.14 | 69.34 | 69.61 | 1.0k |
| MCTS-$\alpha$ (no sampling) | 1 | 63.3 | 63.3 | 63.3 | 0.4k |
| MCTS-$\alpha$-intra tree | 1 | 64.36 | 64.36 | 64.36 | 0.4k |
| MCTS-$\alpha$-intra tree | 10 | 66.85 | 67.68 | 63.90 | 0.9k |
| MCTS-$\alpha$-intra tree | 20 | 67.13 | 69.34 | 68.78 | 1.1k |
| MCTS-$\alpha$-intra tree | 50 | 67.96 | 69.89 | 69.34 | 1.4k |
| MCTS-$\alpha$-inter trees | 1 | 63.3 | 63.3 | 63.3 | 0.4k |
| MCTS-$\alpha$-inter trees | 10 | 72.65 | 82.87 | 82.32 | 4.1k |
| MCTS-$\alpha$-inter trees | 20 | 72.93 | 84.25 | 83.15 | 8.3k |
| MCTS | 1 | 64.0 | 64.0 | 64.0 | 0.4k |
| MCTS | 10 | 70.44 | 70.72 | 70.17 | 0.8k |
| MCTS | 20 | 72.10 | 72.10 | 71.27 | 1.1k |
| MCTS | 50 | 72.38 | 72.38 | 71.55 | 1.6k |
| MCTS-Rollout | 1 | 71.3 | 71.3 | 71.3 | 0.7k |
| MCTS-Rollout | 10 | 73.48 | 73.20 | 72.65 | 0.9k |
| MCTS-Rollout | 20 | 73.48 | 73.48 | 72.38 | 1.0k |
| MCTS-Rollout | 50 | 73.48 | 73.48 | 72.38 | 1.1k |

Table 25: Detailed Results in PrOntoQA

| Method | N | Majority-vote | ORM-vote | ORM-max | #Token |
|---|---|---|---|---|---|
| CoT | - | 48.8 | 48.8 | 48.8 | 92 |
| CoT-SC | 1 | 54.40 | 54.40 | 54.40 | 91.25 |
| CoT-SC | 3 | 63.60 | 82.40 | 82.40 | 273.75 |
| CoT-SC | 10 | 58.40 | 97.80 | 97.80 | 912.55 |
| CoT-SC | 20 | 57.00 | 99.80 | 99.80 | 1.8k |
| CoT-SC-Tree | 1 | 50.20 | 50.20 | 50.20 | 82.02 |
| CoT-SC-Tree | 10 | 62.40 | 98.40 | 98.40 | 413.58 |
| CoT-SC-Tree | 20 | 61.00 | 99.40 | 99.40 | 632.91 |
| BFS-V | 1 | 94.40 | 94.40 | 94.40 | 125.52 |
| BFS-V | 10 | 99.00 | 100.00 | 100.00 | 837.78 |
| BFS-V | 20 | 98.60 | 99.80 | 99.80 | 1.5k |
| DFS-V | 1 | 93.30 | 93.30 | 93.30 | 124.46 |
| DFS-V | 10 | 95.60 | 96.40 | 96.40 | 187.59 |
| DFS-V | 20 | 95.60 | 96.40 | 96.40 | 193.91 |
| MCTS-$\alpha$ (no sampling) | 1 | 99.40 | 99.40 | 99.40 | 183.66 |
| MCTS-$\alpha$-intra tree | 1 | 97.20 | 97.20 | 97.20 | 208.68 |
| MCTS-$\alpha$-intra tree | 10 | 99.80 | 99.80 | 99.80 | 364.96 |
| MCTS-$\alpha$-intra tree | 20 | 99.80 | 99.80 | 99.80 | 441.31 |
| MCTS-$\alpha$-inter trees | 1 | 99.40 | 99.40 | 99.40 | 183.66 |
| MCTS-$\alpha$-inter trees | 10 | 100.00 | 100.00 | 100.00 | 1.9k |
| MCTS-$\alpha$-inter trees | 20 | 100.00 | 100.00 | 100.00 | 3.8k |
| MCTS | 1 | 94.20 | 94.20 | 94.20 | 126.65 |
| MCTS | 10 | 99.60 | 99.60 | 99.60 | 182.88 |
| MCTS | 20 | 100.00 | 100.00 | 100.00 | 240.16 |
| MCTS-Rollout | 1 | 96.90 | 96.90 | 96.90 | 210.41 |
| MCTS-Rollout | 10 | 99.20 | 99.20 | 99.20 | 220.16 |
| MCTS-Rollout | 20 | 99.20 | 99.20 | 99.20 | 224.16 |

Table 26: Detailed Results in RLHF alignment

| Method | N | Mean | Best | #Forward |
|---|---|---|---|---|
| CoT | 1 | 0.387 | 0.387 | 57.8 |
| CoT-SC | 1 | -0.164 | -0.164 | 58 |
| CoT-SC | 10 | -0.182 | 1.592 | 0.6k |
| CoT-SC | 20 | -0.175 | 1.972 | 1.2k |
| CoT-SC | 50 | -0.176 | 2.411 | 2.9k |
| BFS-V | 1 | -1.295 | -1.295 | 61.8 |
| BFS-V | 10 | -1.523 | -1.065 | 0.6k |
| BFS-V | 20 | -1.520 | -0.948 | 1.2k |
| BFS-V | 50 | -1.474 | -0.813 | 3.1k |
| DFS-V | 1 | -1.295 | -1.295 | 61.8 |
| DFS-V | 10 | -1.498 | -1.067 | 67.8 |
| DFS-V | 20 | -1.507 | -0.985 | 71.8 |
| DFS-V | 50 | -1.503 | -0.86 | 85.8 |
| MCTS-$\alpha$ (no sampling) | 1 | 2.221 | 2.221 | 186 |
| MCTS-$\alpha$-intra tree | 1 | 1.538 | 1.538 | 198.50 |
| MCTS-$\alpha$-intra tree | 10 | 1.527 | 3.052 | 1.6k |
| MCTS-$\alpha$-intra tree | 20 | 1.533 | 3.311 | 3.1k |
| MCTS | 1 | -1.295 | -1.295 | 61.8 |
| MCTS | 10 | -1.146 | 0.160 | 0.6k |
| MCTS | 20 | -1.08 | 0.528 | 1.2k |
| MCTS | 50 | -0.961 | 0.981 | 2.8k |
| MCTS-Rollout | 1 | 1.925 | 1.925 | 0.8k |
| MCTS-Rollout | 10 | 2.278 | 2.540 | 1.1k |
| MCTS-Rollout | 20 | 2.376 | 2.643 | 1.2k |
| MCTS-Rollout | 50 | 2.491 | 2.746 | 1.3k |