Toward Self-Improvement of LLMs via Imagination, Searching, and Criticizing

Ye Tian[*], **Baolin Peng**[*], **Linfeng Song**[*], **Lifeng Jin**, **Dian Yu**, **Haitao Mi**[†], **Dong Yu**
Tencent AI Lab, Bellevue, WA
{yaptian,baolinpeng,lfsong,lifengjin,yudian,haitaomi}@global.tencent.com

Abstract

Despite the impressive capabilities of Large Language Models (LLMs) on various tasks, they still struggle with scenarios that involves complex reasoning and planning. Recent work proposed advanced prompting techniques and the necessity of fine-tuning with high-quality data to augment LLMs' reasoning abilities. However, these approaches are inherently constrained by data availability and quality. In light of this, self-correction and self-learning emerge as viable solutions, employing strategies that allow LLMs to refine their outputs and learn from self-assessed rewards. Yet, the efficacy of LLMs in self-refining its response, particularly in complex reasoning and planning task, remains dubious. In this paper, we introduce ALPHALLM for the self-improvements of LLMs, which integrates Monte Carlo Tree Search (MCTS) with LLMs to establish a self-improving loop, thereby enhancing the capabilities of LLMs without additional annotations. Drawing inspiration from the success of AlphaGo, ALPHALLM addresses the unique challenges of combining MCTS with LLM for self-improvement, including data scarcity, the vastness search spaces of language tasks, and the subjective nature of feedback in language tasks. ALPHALLM is comprised of prompt synthesis component, an efficient MCTS approach tailored for language tasks, and a trio of critic models for precise feedback. Our experimental results in mathematical reasoning tasks demonstrate that ALPHALLM significantly enhances the performance of LLMs without additional annotations, showing the potential for self-improvement in LLMs.

1 Introduction

LLMs, trained on trillions of tokens with billions of parameters have shown unparalleled capabilities in a wide range of natural language processing tasks (Touvron et al., 2023b; Team et al., 2023; OpenAI, 2023). Nevertheless, they continue to face challenges in scenarios requiring complex reasoning and strategic planning (Valmeekam et al., 2022; Stechly et al., 2024). While advanced prompting approaches such as Chain, Tree, Graph-of-Thought (Wei et al., 2022; Yao et al., 2024; Besta et al., 2024; Ding et al., 2023), which generate intermediate steps in the reasoning process demonstrate large improvements on reasoning capability of LLMs, it remains essential to fine-tune LLMs using a substantial volume of high-quality, supervised data to fundamentally improve the model performance (Nye et al., 2021; Lewkowycz et al., 2022; Chung et al., 2022). This methodology is inherently limited by the scope and quality of data that humans can provide.

Considering existing challenges, the concept of self-correction and self-learning have been proposed as promising solutions (Madaan et al., 2024; Saunders et al., 2022; Chen et al., 2024). Within these framework, LLMs typically operate by employing two main strategies: 1) they continuously refine their responses based on the feedback of their past responses, and 2) they extensively sample

[*]Equal Contribution; †Corresponding Author

Work in progress.

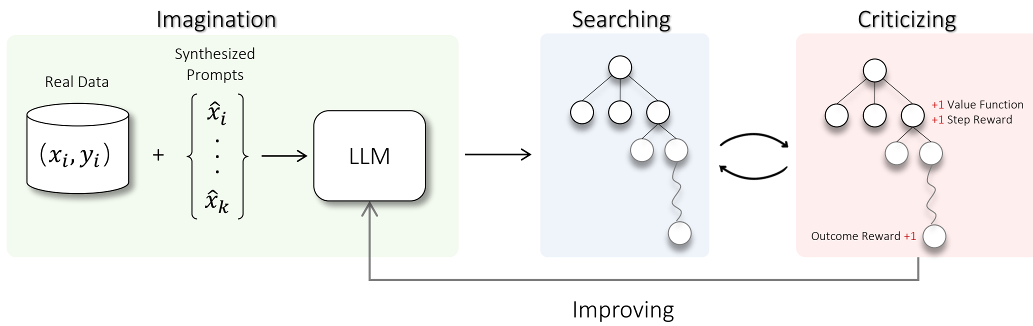

Figure 1: Imagination-Searching-Criticizing self-improvement loop: Imagination component synthesizes prompts as new learning examples, with MCTS searching better trajectories guided by signals from critics for policy improving.

responses then learn from preferences judged by itself as reward models with PPO or DPO (Yuan et al., 2024a,b; Chen et al., 2024). However, it remains a matter of ongoing research whether LLMs can effectively critique their own outputs to either enhance response quality or apply a scalar reward to indicate the quality of responses, especially in contexts demanding intricate planning and reasoning (Valmeekam et al., 2022; Stechly et al., 2024; Huang et al., 2023; Hong et al., 2023). On the other hand, advanced search algorithms such as Monte Carlo Tree Search (MCTS), combined with reinforcement learning, have enabled models to learn from self-play and achieve human parity or even surpass human performance in complex tasks such as the game of Go (Silver et al., 2016, 2017). This naturally raises a question: is it viable to leverage the strengths of MCTS alongside LLMs to inaugurate a novel paradigm of self-improving? More precisely, could the assimilation of MCTS empower LLMs to more effectively explore better responses, guided by strategic signals, and subsequently optimize these responses to enhance overall performance?

To answer this question, we begin with a systematic examination of AlphaGo, identifying three critical aspects for its success: (*i*) The large volume of expert and self-play data; imitation learning on expert data enables it to simulate human-like strategies, and the reinforcement learning on self-play data fosters the emergence of novel tactics that surpass human capabilities (Clark & Storkey, 2015). (*ii*) The use of tree search, which facilitates the exploration of potential moves through statistical sampling of the large search space. This approach allows AlphaGo to effectively identify and simulate the most promising strategies, thereby making highly informed decisions in the complex and vast decision space (Silver et al., 2016). (*iii*) Accurate and unambiguous environment feedback; the direct and accurate feedback (win or loss) provided by the game of Go offers a clear and unequivocal learning signal (Silver et al., 2017). The integration of MCTS with LLMs for self-improvement has several challenges: (*i*) Limited Data: High-quality annotated data for LLMs is generally scarce. Furthermore, how to construct of synthetic data for LLMs training, similar to AlphaGo's self-play data, remains unclear. (*ii*) Search Efficiency: The vast number of potential token combinations in natural language tasks results in an exponentially large search space, posing a significant challenge to the efficiency of MCTS (Ramamurthy et al., 2022). (*iii*) Imperfect Feedback: In contrast to the clear win/loss feedback in Go, feedback in natural language tasks is often subjective and nuanced, without a straightforward measure of success.

In this paper, we introduce ALPHALLM, an imagination-searching-criticizing framework designed for the self-improvement of LLMs . ALPHALLM consists of three key components, as illustrated in Figure 1. First, an imagination component is designed to synthesize prompts, alleviating the issues of data scarcity. Second, we propose ηMCTS tailored for efficient searching in language tasks. Particularly, it has been show that planning at multiple levels of temporal abstraction is critical for RL problems with a long horizon and large action space (Sutton et al., 1999b; Peng et al., 2017; Luketina et al., 2019). As such, we propose formulating the text generation process as options over a Markov Decision Process (MDP) problem, where each option represents the generation of a collection of tokens for a specific subtask, similar to the concept of chains in chain-of-thought prompting. This formulation improves search efficiency by substantially reducing the search depth. Additionally, we propose the use of state fusion and adaptive branching factors to further enhance search efficiency by balancing the trade-off between search width and depth. Lastly, since accurate feedback is crucial to the success of MCTS, we introduce a trio of critic models to guide ηMCTS, including a value function for estimating future rewards, a process reward model for assessing node correctness, and an outcome reward model for evaluating the overall trajectory. For complex tasks with which LLMs struggle assessing such as arithmetic computation and code execution, to ensure the accuracy of feedback, we augment the critics with the capacity to make dynamic decisions on which tools to use, when to use them, and how to use them effectively. After ηMCTS stage, we collect the trajectory with the largest reward from the critic models as the training examples to improve LLMs.

The experimental results on mathematical reasoning tasks demonstrate that ALPHALLM can efficiently search for better responses and use them to improve LLMs' performance, forming an effective self-improving loop. Notably, based on LLaMA-2 70b, ALPHALLM can improve its performance from 57.8 to 92.0 on GSM8K and from 20.7 to 51.0 on MATH, performing comparably to GPT-4. In summary, our contributions are threefold:

- We examine the inherent challenges in harnessing AlphaGo's self-learning algorithms for LLMs, which are data scarcity, the complexity of search spaces, and the nuanced nature of feedback.
- We introduce ALPHALLM, an imagination-searching-criticizing framework that integrates MCTS with LLMs, enabling them to self-improve without the need for additional annotations
- Experiments on mathematical reasoning problems show that, by employing ALPHALLM, we can significantly enhance the performance of LLaMA-2 70B, elevating it to levels comparable with GPT-4 on the GSM8K and MATH datasets when ηMCTS decoding is utilized.

2 Related Work

Search with LLM Effective search strategy has been shown crucial for tasks that involve complex reasoning and planning, such as go (Silver et al., 2016) and math reasoning (Cobbe et al., 2021; Hendrycks et al., 2021). For math reasoning tasks, various search methods have been studied. One direction of research (Zhu et al., 2024; Xie et al., 2024) designed beam search with dynamic pruning, where beam items of low quality are pruned. Another line of work (Yao et al., 2024; Long, 2023; Besta et al., 2024; Hao et al., 2023; Feng et al., 2023) maintains a tree or a graph that represents the current progress of solving the input question where potential branches are iteratively expanded. Both our approach and Feng et al. (2023) are based on the MCTS algorithm, while one main difference is how to define a search step: Feng et al. (2023) fix a search step to be either a token or a sentence, while our approach is more flexible on deciding steps. More importantly, we also study how to leverage MCTS for effective self-improve. We also design the MCTS process more carefully, such as we merge multiple critique signals to effectively guide the search process. As the result, our approach achieves much better performances than Feng et al. (2023).

LLM Self-improving Being a key to the success of scalable oversight (Bowman et al., 2022), self-improving for LLM aims to align the LLM to human preference and values mainly using the supervision from the knowledge inside the LLM. One crucial part of self-improving is how to obtain reliable signal of critique to distinguish between good responses from the LLM and bad ones. Initial work (Bai et al., 2022; Wang et al., 2022) first asks the LLM to generate input queries of diverse tasks and the corresponding outputs. They then rely on hand-crafted heuristic rules to filter out redundant or low-quality data pairs (e.g. the query is too long or too short). Since it is non-trivial to compose effective heuristic rule, later work (Sun et al., 2023; Li et al., 2023; Guo et al., 2024) proposes a few general principles or judging criteria and ask the LLM itself to evaluate the quality its responses based on these guidance. They hope that the LLM can automatically designate these principles into each data point to better guide data filtering. However, this requires the LLM to have strong abilities to apply these principles for each specific case and make correct judgements. Different from previous work, we propose to leverage the supervision from MCTS for LLM self-improvement: taking the outputs of MCTS to continue train the LLM. This is because the outputs from MCTS are usually in much better quality then standard nucleus sampling, and the large gap ensure that the LLM can self improve.

Another line of research explores cheaply available knowledge. Some (Saunders et al., 2022; Wang et al., 2023b) collects large-scale critique data from question-and-answer websites (e.g., stack exchange) for continue pretraining, while others (Gou et al., 2023a) utilize external tools to provide more fine-grained guidance. The goal of both directions is to enhance critique ability of the LLM for self-improving. Our approach based on MCTS is intuitively orthogonal to this line of research.

3 Preliminaries

3.1 Problem Formulation

In this paper, we consider a LLM characterized by probability p_θ and denoted as policy π_θ. It takes a sequence $\boldsymbol{x} = [x_1, \cdots, x_n]$ as input, which is typically referred as prompt, to generate the response $\boldsymbol{y} = [y_1, \cdots, y_m]$. The response $\boldsymbol{y}$ can be viewed as samples from the conditional probability distribution $p_\theta(\cdot|\boldsymbol{x})$. In the context of LLMs, each x_i and y_i represents a token from a pre-defined vocabulary. The policy π_θ operates in an autoregressive manner, where each token is generated sequentially, relying solely on the context provided by the previously generated tokens. The policy therefore constitutes a Markov process in which the conditional probability distribution $p_\theta(\boldsymbol{y}|\boldsymbol{x})$ can be decomposed and expressed with the chain rule:

$$p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{m} p_\theta(y_i|\boldsymbol{x}, \boldsymbol{y}_{<i})$$

With this property, the text generation task can be formulated as an Markov Decision Process (MDP) problem consisting of $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ in which:

- **State** $\boldsymbol{s}_t \in \mathcal{S}$: Represents the context information of current trajectory, *i.e.,* current status of the generation process, *e.g.,* a partial response to a prompt. The initial state s_0 corresponds to the original prompt.
- **Action** $a_t \in \mathcal{A}$: Denotes a single action or sampled token from the vocabulary, leading to a transition to a new state $\boldsymbol{s}_{t+1}$, by concatenating $\boldsymbol{s}_t$ and a_t.
- **Reward** $r_t = R(\boldsymbol{s}_t, a_t)$: Manifest the evaluation of the generation to the prompt, reflecting the desirability or preferences of each state-action pair, such as whether the actions follow instructions in the prompt.

γ denotes the discount factor, while T here signifies the transition probability function. We omit its detailed description as in text generation environment the transition is deterministic.

This MDP framework sets the stage for applying Reinforcement Learning (RL) methods to optimize the policy $\pi_{\boldsymbol{\theta}}$ aiming to maximize the expected cumulative reward R. Base on these setups, we describe the self-improving problem. Given a LLM $\pi_{\boldsymbol{\theta}}$ and an initial dataset $\mathcal{D}^0$, which consists of N expert-generated prompt-response pairs $\{(\boldsymbol{x}_i^0, \boldsymbol{y}_i^0) \mid i \in [N]\}$, the goal of self-improving is to iteratively refine π_θ to maximize the reward. The refinement process includes learning from synthesized prompts and corresponding responses. These responses are obtained using an advanced search algorithm that navigates the space of possible responses to maximize the expected reward. The detailed process is described in Algorithm 1. The primary challenges in forming an effective self-improving loop lie in synthesizing suitable prompts, efficiently searching over a vast action space, and obtaining precise feedback, which will be discussed in §4.

3.2 Monte Carlo Tree Search

MCTS is a sampling-based search algorithm for policy optimization in decision-making problems. It would iteratively build a search tree, by repeating four phases: selection, expansion, evaluation, and backpropagation. In the selection phase, it would recursively select the children from the root node by Upper Confidence Bound (UCB) bandit Auer et al. (2002), which is

$$UCB(i) = w_i + C * \sqrt{2 * \ln \frac{N_i}{n_i}} \tag{1}$$

where n_i and N_i are the visit counts for the node i and its parent respectively, C represents a hyperparameter balancing exploration and exploitation, and the w_i is the average value of all descendant nodes of i. Following selection, the tree undergoes expansion according to the defined policy in the expansion phase. Then in the evaluation phase, the value of the newly expanded node is estimated, by sampling or model-based methods. Finally, in the backpropagation phase, the estimated value is backpropagated to all ancestor nodes of the newly expanded node.

Algorithm 1: LLM self-improving loop

Input Initial dataset $\mathcal{D}^0 = \{(\boldsymbol{x}_i^0, \boldsymbol{y}_i^0) \mid i \in [N]\}$, policy model π_θ^0, reward model R, number of self-improving training loop K

Output θ^k

for $k \leftarrow 1, \ldots, K$ **do**

  Generate synthetic prompts $[\boldsymbol{x}^k] = \texttt{SYN}(\pi_\theta^{k-1}, \mathcal{D}^{k-1})$

  Collect trajectories with search algorithm, *e.g.,* MCTS guided by R.

  $[\hat{\boldsymbol{y}}^k] = \texttt{MCTS}(\pi_\theta^{k-1}, [\boldsymbol{x}^k])$

  Construct dataset $\mathcal{D}^k = \{(\boldsymbol{x}^k, \hat{\boldsymbol{y}}^k)\}$

  Update policy $\theta^k = \arg\min_\theta L(\pi_\theta^{k-1}, \mathcal{D}^k)$

end

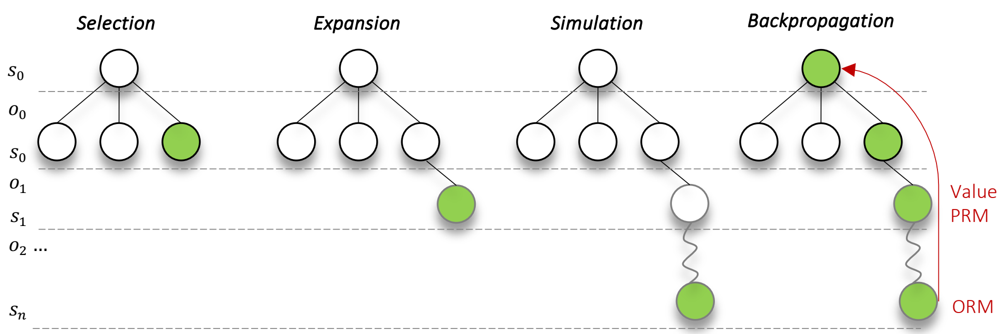

Figure 2: An overview of the four operations of ηMCTS. A node is selected, expanded, simulated with fast rollout policy until a terminal node is reached, then the signals from value function, PRM and ORM are backpropagated.

4 AlphaLLM

4.1 Overview

The architecture of AlphaLLM is depicted in Figure 1, comprising three key components. Firstly, the imagination component is tasked with synthesizing prompts as learning examples. Secondly, an efficient search component, named ηMCTS, is proposed to search high-quality trajectories for optimizing the policy. Lastly, the search process is guided by critics specifically designed to provide reliable signals.

4.2 Data Synthesizing

Let $\mathcal{D}^0 = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i \in [N]\}$ denote the initial dataset consisting of N expert-generated prompt-response pairs. The data synthesizing process aims to expand this dataset by generating a set of synthesized prompts $\mathcal{D}^1 = \{(\boldsymbol{x}_i^1, \cdots) \mid i \in [N]\}$. The generation of each synthesized prompt $\boldsymbol{x}_i^1$ can be mathematically described as a transformation g applied to one or more examples from $\mathcal{D}^0$:

$$\boldsymbol{x}_i^1 = g(\boldsymbol{x}_{i_1}^0, \cdots, \boldsymbol{x}_{i_m}^0, \pi^0)$$

where $\boldsymbol{x}_{i_1}^0, \cdots, \boldsymbol{x}_{i_m}^0$ are selected examples from $\mathcal{D}^0$. The transformation function g controls the synthesis process, which can be a learnable function, manually defined heuristic rules, a strong LLM or the policy model itself π^0 equipped with data synthesis instructions. The data synthesizing process aims to enrich the diversity and complexity presented for the training of the policy model. Among various strategies, such as Self-instruct (Wang et al., 2022), Evol-instruct (Xu et al., 2023), we opt for a method akin to that described in Yu et al. (2023).

4.3 ηMCTS

4.3.1 Option-level MCTS

Search Node	Example	Termination
Token-level	$y_0 \rightarrow y_1 \rightarrow y_2 \rightarrow y_3 \rightarrow y_5 \rightarrow y_6 \rightarrow y_7 \rightarrow y_8$	token
Sentence-level	$y_0y_1y_2$ ⏎ $\rightarrow y_4y_5y_6$ ⏎ $\rightarrow y_7y_8y_9y_{10}$	new line
Option-level	$y_0 \rightarrow y_1y_2$ ⏎ $\rightarrow y_4y_5y_6$ ⏎ $y_7y_8y_9$ ⏎ $\rightarrow y_{10}$	termination function

Table 1: Comparative illustration of token-level, sentence-level, and option-level MCTS search nodes. y denotes a token sampled from the policy model. The arrow $\rightarrow$ represents the transition from one search node to the subsequent node within the search process.

When applying MCTS to LLMs, it is natural to perform token-level search , where each token is considered as an action (Liu et al., 2023). However, the substantial vocabulary size typical of LLMs presents a significant challenge *i.e.,* conducting a deep search in such a vast space becomes increasingly complex as the search space expands exponentially. To mitigate this, some paper proposed a sentence-level search, treating each sentence or step as a search node (Feng et al., 2023). While this method reduces the search space, it might compromise the flexibility and effectiveness of applying MCTS to LLMs, which is particularly true for tasks where subtle variations in token can dramatically impact the outcome, or where a more comprehensive search beyond a sentence is necessary.

Inspired by Sutton et al. (1999a); De Waard et al. (2016), we use the term option as a search node and propose option-level MCTS where each option represents a sequence of tokens, which can range from multiple tokens to several sentences. A comparisons of different levels search is listed in Table 1. Mathematically, an option $o = \langle \mathcal{I}, \pi, \beta \rangle$, where $\mathcal{I} \subseteq \mathcal{S}$ is a set of initial states for the option; $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is a policy to generate actions, which in our case is a LLM; and $\beta : \mathcal{S}^+ \rightarrow [0, 1]$ is the termination function. Starting from a state s_t, we can choose all the options for which $s_t \in \mathcal{I}$. Once an option is chosen, the policy π will generate actions for several steps until the option terminates according to the termination function β. As illustrated in Figure 2, option-level MCTS consists of the following operations:

- **Selection** Starting from the root node, we iteratively select the child node based on Equation 1.
- **Expansion** Once an expandable leaf node is selected, a new node is generated by starting with the previous state of the parent node as the initial option state. The option is then sampled using the policy π, and its completion is determined by the termination function β.
- **Simulation** The scaled reward of the newly expanded node, as well as some simulated future trajectories are evaluated using the feedback functions, which will be discussed in §4.4.
- **Backpropagation** The average value of the newly generated node and all its ancestors is updated using the scaled reward from the evaluation step. Meanwhile, the visit counts for these nodes are also increased by one.

Employing an option to substitute a single token within each node could reduces search space, as the number of options in a trajectory is much smaller than the number of tokens. This facilitates a deeper search, broader coverage of the search space, and minimizes the frequency of requesting feedback from functions such as the value model. Moreover, the option-level offers more flexibility compared to the sentence-level, as a new line can be treated as a special case of the termination function, as demonstrated in Table 1.

4.3.2 Importance Weighted Expansion

In previous work related to option/sentence level tree search Feng et al. (2023); Yao et al. (2024), it has been a common practice to assume that each node in the tree has the same predefined width *i.e.,* branching factor. This is due to the fact that unlike token-level MCTS with a limited action space, the sample space at the option-level is exceedingly large, with an unlimited number of token combinations. Consequently, it is necessary to set a predefined maximum width. However, this assumption can often result in an inefficient search space, as it may be either too large or too small.

A more effective and efficient way to determine the branching factor for each node is to dynamically adjust it based on the importance of each node. This approach allows us to allocate a larger child budget to nodes of higher importance, thereby preventing inefficient exploration of these nodes and ensuring that we do not miss promising solutions. Meanwhile, by reducing the number of children for less important nodes, we can perform deeper searches at various levels of the tree, rather than considering all possible options at each node. Inspired by Taylor et al. (2014); Clouse (1996), we define the importance of a node $\boldsymbol{s}_t$ as:

$$I(\boldsymbol{s}_t) = \max_{\boldsymbol{o}_t} |v^{\pi}([\boldsymbol{s}_t, \boldsymbol{o}_t]) - v^{\pi}(\boldsymbol{s}_t)|$$

where v^{π} is the value function which will be detailed in §4.4. $I(\boldsymbol{s}_t)$ captures the maximum value deviation from the current state. When this value is small, there is no need to explore further on this node, as there will not be a significant difference by rolling out on this node. Conversely, if the value is large, it is worth trying different children. We set the number of children allowed for a node $n(\boldsymbol{s}_t)$ to be linear with this importance, using a factor α. In practice, to avoid extreme cases, we bound the number of children by depth-dependent constants $c_{\mathtt{min}}(t)$ and $c_{\mathtt{max}}(t)$:

$$n(\boldsymbol{s}_t) = \max\left(c_{\mathtt{min}}(t), \min\left(\lfloor \alpha I(\boldsymbol{s}_t) \rfloor, c_{\mathtt{max}}(t)\right)\right).$$

4.3.3 State Merge

With $n(\boldsymbol{s}_t)$ determined, another issue is that states under the same node can be very similar, causing many unnecessary sub-trees. To maximize diversity among states and cover as much space as possible with limited rollouts, we utilize the concept of move groups Van Eyck & Müller (2012). By partitioning available options into distinct groups based on their similarities, with the maximum number of groups equal to the branching factor, we enhance diversity among groups. This strategy allows us to cover a larger problem space with limited search rollouts, making the search process more efficient.

In practice, each time we generate a new option from the policy, we use some heuristic functions to measure its similarity with existing options. The heuristic function can either be a faster rule-based measurement (*e.g.,* edit distance) or a model-based method (*e.g.,* prompting a LLM). Based on this, we decide whether to merge this option with a previous one or create a new group. This process is repeated until a maximum number of repetitions is reached. The details of this process are outlined in Algorithm 2.

Algorithm 2: Find Action with Minimum Distance Larger Than Threshold

Input max number of trails max_trials, threshold $thres$
Output pool of children nodes
$n \leftarrow 0$
$min\_dist \leftarrow 0$
while $n < max\_trials$ *and* $min\_d \leq thres$ **do**
    $\boldsymbol{o}_t \sim \pi(s_t)$
    $min\_d \leftarrow \min_{\boldsymbol{o} \in A_{t,\mathtt{pool}}} \mathtt{Dist}(\boldsymbol{o}_t, \boldsymbol{o})$
    $n \leftarrow n + 1$
end
Add $\boldsymbol{s}_{t+1} = [\boldsymbol{s}_t, \boldsymbol{o}_t]$ to the pool of children nodes

In Algorithm 2, we iteratively sample an option $\boldsymbol{o}_t$ from the policy $\pi(\boldsymbol{s}_t)$ and compute the minimum distance min_d between $\boldsymbol{o}_t$ and the actions in the pool $A_{t,\mathtt{pool}}$ measured by distance function `Dist`. If min_d is larger than a predefined threshold $thres$ or the maximum number of trials max_trials is reached, the loop terminates and the resulting state $\boldsymbol{s}_{t+1}$ is added to the pool of children nodes.

4.3.4 Fast Rollout with Specialized LM

The simulation operation which employs a rollout policy to project future trajectories from a given state, is crucial for an effective MCTS. This process significantly improves the efficiency of exploration and exploitation, and enhances the accuracy of reward estimation[2]. By simulating numerous

[2]Typically, the closer the simulation is to the termination state, the more accurate the reward estimation becomes.

potential trajectories, MCTS can better approximate the likely outcomes of various actions, thereby facilitating a more informed and search process.

Ideally, π_θ would serve as the rollout policy, yet its computational demands render it impractical for the rapid simulations required by MCTS. To address this challenge, we propose the use of a smaller, specialized LM as the fast rollout policy $\pi^{\texttt{fast}}$. Given a state $\boldsymbol{s}_t$, the fast rollout policy $\pi^{\texttt{fast}}$ efficiently continues generation until it reaches a termination condition, denoted as $\pi^{\texttt{fast}}(\boldsymbol{s}_t)$.

4.4 Critic

It is crucial for searching algorithms to have reliable guidance signals towards achieving the end goal. In ALPHALLM, we design three types of critic models to guide the search process, *i.e.,* value function v^π predicting the future reward, process reward models PRM estimating node quality, and outcome reward model ORM assessing the overall trajectory quality.

Value Function The value function, denoted as $v^\pi(\boldsymbol{s})$, is the expected reward starting from state $\boldsymbol{s}_t$ following the policy π thereafter. To train a value function $v^\pi_\phi(\boldsymbol{s})$ parameterized by ϕ, we use the Monte Carlo (MC) estimate to empirically approximate the expected reward by averaging the rewards observed after many samplings starting from state s and following policy π. The reward from a state is the sum of rewards obtained in the future, discounted by a factor γ at each time step. Thus, the MC estimate of $v^\pi_\phi(\boldsymbol{s})$ can be written as $v^\pi_\phi(\boldsymbol{s}) \approx \frac{1}{J}\sum_{j=1}^{J} G^{(j)}(\boldsymbol{s})$ where J is the number of trajectory starting from state $\boldsymbol{s}$, $G^{(j)}(\boldsymbol{s})$ is the total discounted reward from state s in the j-th trajectory. Particularly, given the expert demonstration dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, for each prompt $\boldsymbol{x}_i$, we generate trajectories $\boldsymbol{\tau}_i^j = \{\boldsymbol{x}_i, \boldsymbol{o}_{i1}^j, \boldsymbol{o}_{i2}^j, \cdots, \boldsymbol{o}_{iT}^j\}$ by following policy π. A reward r_i^j is assigned to indicate whether $\boldsymbol{\tau}_i^j$ aligns with $\boldsymbol{y}_i$, *e.g.,* rewarding trajectories that contains correct answers in mathematical tasks or closely follows the instruction as the ground-truth. We then construct a dataset $\mathcal{D}_{\texttt{value}} = \{(\boldsymbol{s}_{it}, v_{it}) | i \in [N], t \in [T]\}$ in which $\boldsymbol{s}_{it} = [\boldsymbol{x}_i \cdot \boldsymbol{o}_{<it}]$ and $v_{it} = \frac{1}{J}\sum_{j=1}^{J} r_{iT}^j$. The value function v^π_ϕ is optimized by minimizing mean squared error:

$$\mathcal{L}_\phi = -\mathbb{E}_{(\boldsymbol{s},v)\sim\mathcal{D}_{\texttt{value}}}(v^\pi_\phi(\boldsymbol{s}) - v)^2$$

We opt to initialize v^π_ϕ using the parameters from policy π_θ, incorporating an MLP layer on top of it to output a scalar on each token. The scalar prediction at the last token of each state is used as the value.

PRM The value function often struggles with credit assignment problem (Sutton, 1984) and its learning could be inefficient due to delayed and sparse rewards (Sutton & Barto, 2018). Therefore, we propose to incorporate PRM that introduces process supervision (Lightman et al., 2023) for direct option assessment. PRM generates intrinsic rewards (Chentanez et al., 2004) to encourage explorations of advantageous options, effectively mitigating issues of reward sparsity by providing immediate, action-specific rewards. Given a state $\boldsymbol{s}_t$ and an option $\boldsymbol{o}_t$ at time t, the PRM aims to predict the immediate reward $r_t^{\texttt{PRM}}$ that results from taking option $\boldsymbol{o}_t$ in state $\boldsymbol{s}_t$. Formally, the PRM is a function $R(\boldsymbol{s}_t, \boldsymbol{o}_t) \rightarrow r_t^{\texttt{PRM}}$. Instead of adding a MLP layer on top of the policy model for outputting a scalar reward (Ouyang et al., 2022), we formulate PRM as a text generation task to best leverage LLM's intrinsic knowledge for assessing the quality of an option. We use prefix sampling (Wang et al., 2023a) to estimate the quality of an option by starting from an option and exploring the final reward after reaching terminal states. The intuition is that an intermediate step can be regarded as a good if it frequently leads to achiving the goal. We adapt the dataset constructed for the value function as $\mathcal{D}_{\texttt{PRM}} = \{(\boldsymbol{s}_{it}, \boldsymbol{o}_t, r_t^{\texttt{PRM}}) | i \in [N], t \in [T]\}$ where $r_t^{\texttt{PRM}}$ is the textual description of the reward, *e.g.,* an option can be regarded as good if v_{it} is larger than certain threshold. To train PRM, we initialize it from the policy model π and use the following prompt templates and typical language model loss.

```
###[A detailed rubric that specifies how to evaluate a step of a task]\n\n###
State\n{state}\n\n###Action\n{option}\n\n###Assessment\n{textual reward}
```

ORM In additional to the value function and PRM, we introduce ORM to guide MCTS. ORM is designed to evaluate options sequences in their entirety, assessing the extent to which the complete trajectory aligns with the desired end goal. The outcome evaluation complements value function and PRM by offering a comprehensive assessment of trajectories. Crucially, ORM plays a vital role in the simulation stage of MCTS by providing more accurate signals on the terminal state, which in turn facilitates a more balance between exploration and exploitation strategies. ORM is formulated as a text generation task, similar to PRM. We leverage the same dataset for the value function training and construct $\mathcal{D}_{\texttt{ORM}} = \{(\boldsymbol{x}_i, \boldsymbol{o}^i_{1:T}, r^{\texttt{ORM}}_i)|i \in [N]\}$, where each instance includes a initial state or prompt $\boldsymbol{x}_i$, a sequence of actions or options $\boldsymbol{o}^i_{1:T}$ taken from that state, and a textual reward $r^{\texttt{ORM}}_i$ indicating the sequence's success or quality. Similarly, ORM is initialized from the policy model π and the following prompt templates and language model loss are used for training.

> ###[A detailed rubric that specifies how to evaluate a complete trajectory of a task]\n\n### Prompt\n{prompt}\n\n###Trajectory\n{trajectory}\n\n###Assessment\n{textual reward}

4.5 Policy Self-Improvement

We have discussed how ηMCTS can guide policy to find trajectories of higher quality and. In this subsection, we discuss how to leverage these trajectories to further improve the policy. It is an iterative process with each iteration containing two main steps: *data generation* and *policy finetuning*.

Data generation In this step, we assume to have the current policy π_{θ_k} and synthetic prompts $\mathcal{D}_k = \{\boldsymbol{x}^k_1, \dots\}$ at the k-th round, where each $\boldsymbol{x}^k_1$ represents a question. We obtain the corresponding training data $\mathcal{D}_k$ for policy π_{θ_k} by firstly performing ηMCTS on $\mathcal{D}_k$ (§4.3) and then sampling a trajectory $\boldsymbol{y}^k_i$ from the corresponding MCTS forest for each question $\boldsymbol{x}^k_i$. There are several ways to select a trajectory from a MCTS forest, such as taking a greedy path based on the critic score (w_i in Eq. 1). Here we choose the trajectory that yield the highest critic score on the leaf node for each input question. As the next step, we filter out instances where the corresponding trajectory is not in high quality:

$$\mathcal{D}_k = \{(\boldsymbol{x}^k_i, \boldsymbol{y}^k_i) \mid f(\boldsymbol{x}^k_i, \boldsymbol{y}^k_i) > \gamma\}$$

where f represents the quality evaluating function for quality scoring, and γ represents the threshold. There can be several ways to implement the function, and here we simply use the ORM (§4.4).

Policy finetuning With the obtained training data $\mathcal{D}_k$, we organize the data into the following prompt templates:

> A chat between a curious user and an artificial intelligence assistant.\n The assistant gives helpful, detailed, and polite answers to the user's questions.\n User: $\boldsymbol{x}_i$\n Assistant: $\boldsymbol{y}_i$

Then the policy π_{θ_k} is finetuned using target-loss SFT:

$$\mathcal{L}_{\theta_k} = \mathbb{E}_{(\boldsymbol{x}^k_i, \boldsymbol{y}^k_i)\sim\mathcal{D}_k}\big[\log \pi_{\theta_k}(\boldsymbol{y}^k_i|\boldsymbol{x}^k_i)\big]$$

This results in an updated policy $\pi_{\theta_{k+1}}$. We leave other training methods, such as DPO (Rafailov et al., 2023) or PPO (Schulman et al., 2017) in future work.

5 Experiments

5.1 Evaluation Setups

Datasets ALPHALLM is generally applicable to a wide spectrum tasks. As an early exploration, in this paper, we conduct experiments on mathematical reasoning problems where the learning signals are clear to define *i.e.,* , final answer is correct or wrong. We choose to evaluate on two widely used datasets GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). For GSM8K, we utilize the whole test set while for MATH, due to computation constraints, we utilize a subset following the same procedure of Lightman et al. (2023).

Metrics We evaluate the performance of predicting answers correctly for policy models. In the same time, we calculate the average rollouts, represented by the number of nodes in the tree, as a measure of computational efficiency.

5.2 Baseline Systems

We evaluate the performance of ALPHALLM against a suite of proprietary model, including OpenAI's GPT-4 and GPT-3.5, Anthropic's Claude-2, as well as Google's PaLM-2 and the gemini model family. To ensure a fair and consistent evaluation, we employ CoT as our primary prompting method. We additionally report PAL (Gao et al., 2023) prompting performance with GPT-4 as it demonstrates enhanced performance.

Additionally, we conduct comparisons with strong open-source models, including LLaMA-2 70B (Touvron et al., 2023a) and Wizardmath 70B (Luo et al., 2023). For LLaMA-2 70B, we present results from few-shot prompting as well as zero-shot prompting for its SFT version, which was trained using CoT rationales and final answers. Wizardmath 70B has been trained on a diverse set of mathematical data generated by ChatGPT, employing both SFT and RLHF. We provide zero-shot prompting results.

5.3 Implementation Details

We select LLaMA-2 70B as the policy model for the GSM8K dataset and Wizardmath 70B V10 for the MATH dataset. To construct the training dataset for the value function, `PRM` and `ORM`, we generate 50 trajectories for each prompt and construct the training target following Section 4.4. Both `PRM` and `ORM` are initialized using the weights from the policy model. In the design of `ORM`, tool usage is not incorporated for GSM8K. However, for MATH, we enhance `ORM` by incorporating tools like pythoin sympy to assess the quality of a trajectory, in a manner similar to that described by Gou et al. (2023b). The training employ a learning rate of 1e-6 and are trained for one epoch. For the fast rollout policy model, we opt for the Abel-002-7B model (Chern et al., 2023) for both the GSM8K and MATH tasks for its high efficiency and superior performance.

We set the MCTS parameters as follows: in GSM8K, $c = 1$ for the small scale (`#rollout`) and 1.5 for the large scale, with $\alpha = 1$. For $t = 0$, $c_{\text{min}}(0) = 10$ for the small scale and 40 for the large scale, while for the rest of t, $c_{\text{min}}(t) = 2$. We also set $c_{\text{max}}(0) = 10$ for the small scale and 40 for the large scale, and for the remaining t, $c_{\text{max}}(t) = 10$. The termination condition is based on sentence termination. In MATH, the parameters are $c = 1$, $\alpha = 1$, and for $t = 0$, $c_{\text{min}}(0) = 10$ for the small scale and 20 for the large scale, while for the rest of t, $c_{\text{min}}(t) = 3$. We set $c_{\text{max}}(0) = 10$ for the small scale and 20 for the large scale, and for the remaining t, $c_{\text{max}}(t) = 10$. The termination function is rule-based, checking if there are any formulations or calculations in the sentence. If there are, the option is terminated; otherwise, the option continues to extend.

For policy self-improving (§4.5), we train the policy model up to 3 epochs, setting batch size to 128, learning rate to 5×10^{-6} and minimal learning rate to 1×10^{-6}. Linear warm-up and decay is used with warm-up percent to be 10%. We perform early stopping based on a devset held out from the training instances. For second-round self-improving, we sample 7.9k MetaMath (Yu et al., 2023) prompts to obtain the corresponding MCTS outputs for training.

5.4 Results

Table 2 lists the performance comparisons of various methods on the GSM8K and MATH datasets. Our findings reveal that ALPHALLM, which utilizes only final answer annotations and self-improves through the training on synthetic prompts with responses from ηMCTS, outperforms both LLaMA-2 70B and WizardMath 70B V1.0—even though these models are trained on a larger set of examples that include both rationales and final answer annotations. This comparison underscores the efficacy and broad applicability of our imagination-searching-criticizing self-improving framework. Moreover, when our model is augmented with ηMCTS decoding strategy, its performance markedly improves, achieving scores of 88.9 and 48.7 on the GSM8K and MATH datasets, respectively. Following two iterations of self-improvement using synthetic prompts, ALPHALLM demonstrates performance comparable to that of GPT-4. This suggests a viable approach to improving LLMs' capabilities in complex problem-solving tasks in a self-improving fashion, leveraging a minimal amount of labeled data.

Model	Decoding	#Annotation	RN	FA	SYN	GSM8K	MATH
GPT-3.5	Sampling	-	-	-	-	80.8	35.5
GPT-4	Sampling	-	-	-	-	92.0	42.5
GPT-4 (PAL)	Sampling	-	-	-	-	94.2	51.8
Gemini 1.0 Pro	Sampling	-	-	-	-	77.9	32.6
Gemini 1.0 Ultra	Sampling	-	-	-	-	88.9	53.2
Gemini 1.5 Pro	Sampling	-	-	-	-	92.5	58.5
Claude-2	Sampling	-	-	-	-	85.2	32.5
PaLM-2 540B	Sampling	-	-	-	-	80.7	34.3
LLaMA-2 70B	Greedy	0	×	×	×	57.8	-
LLaMA-2 70B SFT	Greedy	7.5k	✓	✓	×	69.3	-
WizardMath 70B V1.0	Greedy	96k	✓	✓	×	-	20.7
ALPHALLM	Greedy	7.5k/3k	×	✓	✓	73.7	23.6
ALPHALLM	ηMCTS	7.5k/3k	×	✓	×	88.9	48.7
ALPHALLM	ηMCTS	7.5k/3k	×	✓	✓	92.0	51.0

Table 2: Comparison results of ALPHALLM on the GSM8K and MATH datasets, utilizing LLaMA-2 70B and WizardMath 70B V1.0 as base models for GSM8K and MATH datasets, respectively. #Annotation indicates the quantity of labeled data employed for fine-tuning each base model. The annotation used for training are noted as RN for rationales and FA for final answers. SYN means models trained on synthetic prompts, where trajectories were generated using ηMCTS.

Method	#Responses	GSM8K		MATH	
		#Rollouts	Accuracy	#Rollouts	Accuracy
Greedy	1	4.6	57.8	9.9	20.7
Self-consistency	10	46	67.4	99	22.5
	30	137	74.2	299	27.3
	50	229	75.4	499	28.8
Re-ranking	10	46	80.8	99	34.1
	30	137	86.3	299	39.0
	50	229	87.7	499	42.0
ηMCTS	-	55	87.0	223	45.4
	-	230	88.9	341	48.7

Table 3: Comparative results of various searching method on GSM8K and MATH.

In addition, table 3 presents the performance of various methods applied to different number of responses, from 10 to 50. Our analysis confirms several key findings: 1) Reranking utilizing ORM consistently outperforms self-consistency techniques, indicating that ORM is capable of generating meaningful signals for searching. 2) ηMCTS demonstrates superior performance while requiring significantly fewer rollouts. For instance, on the MATH dataset, ηMCTS achieves better results with only half the number of rollouts compared to reranking. These results suggest that our design of an efficient MCTS in ALPHALLM can serve as an effective policy improvement operation, enabling the search for high-quality trajectories with reduced computational cost.

5.5 Ablation Study

We assess the effectiveness of each component in ALPHALLM and report the results on GSM8K in Table 4(a). Vanilla MCTS, that is coupled with only value function, yields an accuracy of 84.9%, which is used as a reference point to assess the incremental benefit provided by each subsequent component. The addition of PRM improves the accuracy modestly to 85.9%, showing the effectivenss of process supervision for searching. A more significant improvement is observed

PRM	FR-ORM	SM	LG-#Rollout	Acc
×	×	×	×	84.9
✓	×	×	×	85.9
✓	✓	×	×	86.5
✓	✓	✓	×	87.0
✓	✓	✓	✓	88.9

(a) Ablation study on GSM8K

TA-ORM	Option	Acc	#Rollout
×	×	38.8	201
✓	×	44.1	198
✓	✓	45.4	148

(b) Ablation study on MATH

Table 4: **(a)**: Ablation studies on the GSM8K test set of various components of ηMCTS, including PRM, fast-rollout with ORM, state merge, and large number of rollouts. **(b)**: Ablation studies of the impacts of tool-augmented ORM and option-level formulation on MATH.

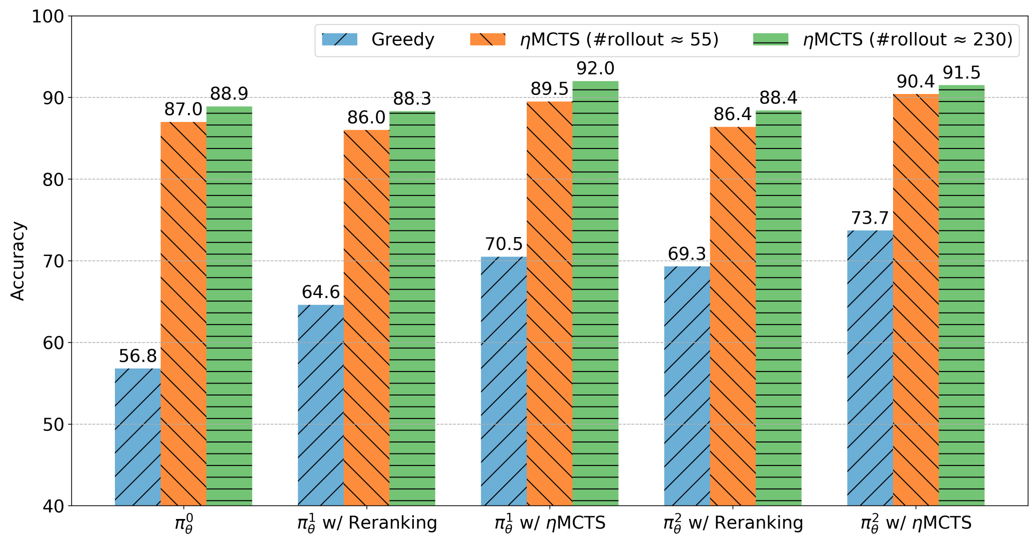

Figure 3: Empirical analysis on GSM8K of different self-improving data collection methods and number of iterations. Models are evaluated with greedy decoding, ηMCTS with small #rollout and large #rollout. Two iterations of self-improvement are conducted using data from reranking and ηMCTS

with the introduction of ORM with fast rollout, which boosts the accuracy to 86.5%. Integrating state merging results in a further increase in accuracy, reaching 87.0%. Finally the combined of increasing the number of rollouts with the other components yields the best performance on this task.

Table 4(b) presents the ablation study of option formulation and the tool-augmented critic on the MATH dataset. Our proposed ηMCTS achieves an accuracy of 45.4 with 148 rollouts. When options are excluded, reverting to essentially sentence-level MCTS, the performance decreases to 44.1 with a noticeable increase in the number of rollouts to 198. This demonstrates that option formulation introduces enhanced flexibility to MCTS, enabling better performance with fewer search efforts. Furthermore, the most significant decrease in performance is observed when only intrinsic knowledge is utilized for ORM, which drops to an accuracy of 38.8. This suggests that the absence of an external tool critically impedes the ORM's capability to effectively assess challenging math problems.

Figure 3 depicts a comparative results on GSM8K of two rounds of self-improving trained on trajectories collected using reranking and ηMCTS. We report the performance of greedy decoding, ηMCTS with a moderate number of rollouts (55), and ηMCTS with a large number of rollouts (230) for each model. We observe that 1) Models trained on the trajectories from reranking or ηMCTS outperform the initial policy by a significant margin. In addition, the performance can be iteratively improved with training suggesting that self-improving has the potential to achieve continual performance gain. 2) While both reranking and ηMCTS can generate high-quality trajectories for self-improving , ηMCTS is performant with high efficiency and better accuracy. Models trained on trajectories generated by it not only exceed the performance of those trained on reranked trajectories but also, when decoded with ηMCTS, demonstrate on par performance with GPT-4, revealing that ALPHALLM is an effective self-improving framework.

6 Limitations and Future Work

Despite the promising results demonstrated by ALPHALLM in this study, there are several limitations that requires further exploration. (*i*) Our current implementation employs relatively simple methods for generating synthetic prompts. Future iterations of ALPHALLM should explore advanced techniques, such as Self-Instruct, to create both diverse and model capability-awared prompts. (*ii*) Although ALPHALLM demonstrates improvements over base models, its performance in greedy sampling is substantially inferior to that observed when decoded with ηMCTS. This indicates that the full potential of MCTS for self-improvement in LLMs has not yet been fully realized. Two potential factors contributing to this issue have been identified: a) the self-improvement loop may not be leveraging sufficient data; and b) the base model may be limited in its capacity for rapid learning. Addressing these concerns could lead to more significant improvemens. (*iii*) In our existing framework, the critic models remain static. We will explore mechanisms to continually update critic models to adapt to new policy models. This will help ensure the discriminator-generator gap and improve the overall training dynamics. (*iv*) The evaluation of ALPHALLM has been limited to mathematical reasoning tasks. To verify the generalizability and broader applicability of the framework, future research will need to extend its application to other domains.

7 Conclusion

In this paper, we introduce ALPHALLM, an imagination-searching-criticizing framework designed for the self-improvement of LLMs without the necessity of additional annotations. At the heart of it is the integration of MCTS with LLMs. To tackle the inherent challenges associated with this integration, including data scarcity, the vastness of search spaces, and the subjective nature of feedback in language tasks, we introduce a data synthesizer for strategic prompt synthesis, an optimized MCTS tailored for efficient search in language tasks, and a trio of critic models to provide precise feedback. Our experimental findings on mathematical reasoning tasks reveal that ALPHALLM significantly boosts the performance of LLMs without requiring extra data annotations. Moreover, when decoded with ηMCTS, ALPHALLM performs comparably to GPT-4, highlighting the potential for self-improvement in LLMs.

References

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 17682–17690, 2024.

Samuel R Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, et al. Measuring progress on scalable oversight for large language models. *arXiv preprint arXiv:2211.03540*, 2022.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *arXiv preprint arXiv:2401.01335*, 2024.

Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

Ethan Chern, Haoyang Zou, Xuefeng Li, Jiewen Hu, Kehua Feng, Junlong Li, and Pengfei Liu. Generative ai for math: Abel. `https://github.com/GAIR-NLP/abel`, 2023.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play go. In *International conference on machine learning*, pp. 1766–1774. PMLR, 2015.

Jeffery Allen Clouse. *On integrating apprentice learning and reinforcement learning*. University of Massachusetts Amherst, 1996.

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

Maarten De Waard, Diederik M Roijers, and Sander CJ Bakkes. Monte carlo tree search with options for general video game playing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. IEEE, 2016.

Ruomeng Ding, Chaoyun Zhang, Lu Wang, Yong Xu, Minghua Ma, Wei Zhang, Si Qin, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Everything of thoughts: Defying the law of penrose triangle for thought generation. *arXiv preprint arXiv:2311.04254*, 2023.

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. Critic: Large language models can self-correct with tool-interactive critiquing. In *Second Agent Learning in Open-Endedness Workshop*, 2023a.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023b.

Hongyi Guo, Yuanshun Yao, Wei Shen, Jiaheng Wei, Xiaoying Zhang, Zhaoran Wang, and Yang Liu. Human-instruction-free llm self-alignment with limited samples. *arXiv preprint arXiv:2401.06785*, 2024.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

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

Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. A closer look at the self-verification abilities of large language models in logical reasoning. *arXiv preprint arXiv:2311.07954*, 2023.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259*, 2023.

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

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. Making ppo even better: Value-guided monte-carlo tree search decoding. *arXiv preprint arXiv:2309.15028*, 2023.

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

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. *ArXiv*, abs/1906.03926, 2019. URL `https://api.semanticscholar.org/CorpusID:182952502`.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

R OpenAI. Gpt-4 technical report. *arXiv*, pp. 2303–08774, 2023.

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

Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2017.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Rajkumar Ramamurthy, Prithviraj Ammanabrolu, Kianté Brantley, Jack Hessel, Rafet Sifa, Christian Bauckhage, Hannaneh Hajishirzi, and Yejin Choi. Is reinforcement learning (not) for natural language processing?: Benchmarks, baselines, and building blocks for natural language policy optimization. *ArXiv*, abs/2210.01241, 2022. URL `https://api.semanticscholar.org/CorpusID:252693405`.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

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

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. *arXiv preprint arXiv:2402.08115*, 2024.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. Principle-driven self-alignment of language models from scratch with minimal human supervision. *arXiv preprint arXiv:2305.03047*, 2023.

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

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999a. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL `https://www.sciencedirect.com/science/article/pii/S0004370299000521`.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999b.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

Matthew E Taylor, Nicholas Carboni, Anestis Fachantidis, Ioannis Vlahavas, and Lisa Torrey. Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1): 45–63, 2014.

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

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

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

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). *arXiv preprint arXiv:2206.10498*, 2022.

Gabriel Van Eyck and Martin Müller. Revisiting move groups in monte-carlo tree search. In *Advances in Computer Games: 13th International Conference, ACG 2011, Tilburg, The Netherlands, November 20-22, 2011, Revised Selected Papers 13*, pp. 13–23. Springer, 2012.

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

Tianlu Wang, Ping Yu, Xiaoqing Ellen Tan, Sean O'Brien, Ramakanth Pasunuru, Jane Dwivedi-Yu, Olga Golovneva, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. Shepherd: A critic for language model generation. *arXiv preprint arXiv:2308.04592*, 2023b.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36, 2024.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

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

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024b.

Tinghui Zhu, Kai Zhang, Jian Xie, and Yu Su. Deductive beam search: Decoding deducible rationale for chain-of-thought reasoning. *arXiv preprint arXiv:2401.17686*, 2024.