Accessing GPT-4 level Mathematical Olympiad Solutions via Monte Carlo Tree Self-refine with LLaMa-3 8B: A Technical Report

Di Zhang
Fudan University
Shanghai Artificial Intelligence Laboratory
di.zhang@ustc.edu

Xiaoshui Huang
Shanghai Artificial Intelligence Laboratory
xiaoshuihuang2019@gmail.com

Dongzhan Zhou
Shanghai Artificial Intelligence Laboratory
zhoudongzhan@pjlab.org.cn

Yuqiang Li
Shanghai Artificial Intelligence Laboratory
liyuqiang@pjlab.org.cn

Wanli Ouyang
Shanghai Artificial Intelligence Laboratory
wanli.ouyang@sydney.edu.au

Abstract

This paper introduces the MCT Self-Refine (MCTSr) algorithm, an innovative integration of Large Language Models (LLMs) with Monte Carlo Tree Search (MCTS), designed to enhance performance in complex mathematical reasoning tasks. Addressing the challenges of accuracy and reliability in LLMs, particularly in strategic and mathematical reasoning, MCTSr leverages systematic exploration and heuristic self-refine mechanisms to improve decision-making frameworks within LLMs. The algorithm constructs a Monte Carlo search tree through iterative processes of Selection, self-refine, self-evaluation, and Backpropagation, utilizing an improved Upper Confidence Bound (UCB) formula to optimize the exploration-exploitation balance. Extensive experiments demonstrate MCTSr's efficacy in solving Olympiad-level mathematical problems, significantly improving success rates across multiple datasets, including GSM8K, GSM Hard, MATH, and Olympiad-level benchmarks, including Math Odyssey, AIME, and OlympiadBench. The study advances the application of LLMs in complex reasoning tasks and sets a foundation for future AI integration, enhancing decision-making accuracy and reliability in LLM-driven applications. Codes publicly accessible at github.com/trotsky1997/MathBlackBox.

1 Introduction

With the rapid evolution of artificial intelligence, large language models (LLMs) such as GPT-4 (Achiam et al., 2023) and LLaMA (Touvron et al., 2023) have become fundamental in advancing natural language processing (NLP) capabilities. These models, characterized by their multi-billion parameter architectures, exhibit remarkable language comprehension and generation abilities. Their emergent properties, including reasoning and in-context learning, have opened new avenues for addressing complex NLP tasks beyond traditional domains, encompassing mathematical problem-solving (Yu et al., 2023; Yuan et al., 2023), recommendation systems (Lyu et al., 2023), and even

Preprint. Under review.

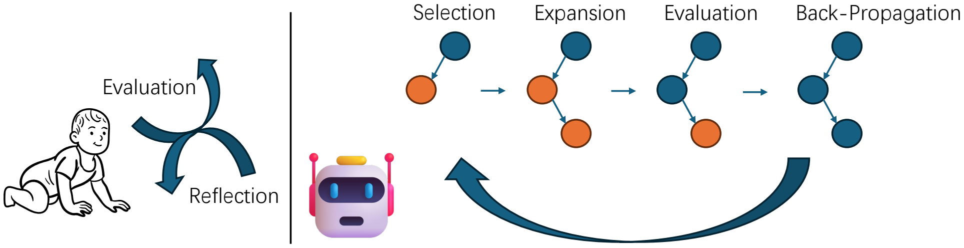

Figure 1: Agents can learn decision-making and reasoning from the trial-and-error as humans do.

molecule generation (Liang et al., 2023). However, despite these advancements, LLMs face notable challenges in areas demanding strategic and logical reasoning.

One significant hurdle is the accuracy and trustworthiness of the outputs. Especially in mathematical contexts, where precision is paramount, the reasoning capabilities of LLM always suffer from prone to producing hallucinations—outputs (Huang et al., 2023) that, while superficially plausible, but irrelevant or factually incorrect, are finally harmful to rational processes. Though rewriting techniques like Self-Refine (Madaan et al., 2023) can help relieve, this tendency can still lead to misleading or wrong outcomes in real-world complex mathematical problems.

To address these challenges, this paper proposes MCT Self-Refine (MCTSr), an integration of LLMs with a Monte Carlo Tree Search (MCTS) algorithm (Chaslot et al., 2008), focusing on enhancing LLMs' performance in complex mathematical reasoning tasks, such as those encountered in mathematical Olympiads. MCTS, a decision-making tool widely used in AI for scenarios requiring strategic planning, is typically employed in gaming and complex problem-solving environments. By combining MCTS's systematic exploration capabilities with LLMs' capabilities of Self-Refine and Self-Evaluation, we aim to create a more robust framework for tackling intricate reasoning tasks that current LLMs struggle with.

Several technical challenges exist in adapting MCTS for LLM integration. Traditional MCTS strategies may not align well with the stochastic and generative nature of LLM outputs, which often involve an infinite, continuous space of potential actions. This misalignment necessitates a tailored approach to expectation calculation and Backpropagation within the MCTS framework to better suit the unique characteristics of LLMs. Furthermore, we introduce a dynamic pruning strategy incorporating an improved upper confidence bound (Srinivas et al., 2009) (UCB) formula to optimize the exploration-exploitation balance essential for effective decision-making in high-stakes tasks.

Our primary contributions are as follows:

We develop and validate a novel reasoning algorithm by integrating LLMs with UCT-MCTS. We enhance the algorithm's key components to accommodate the integration with LLMs better and demonstrate its effectiveness on Olympic-level mathematical problems.

We propose a dynamic pruning module that refines decision-making processes within the MCTS framework, facilitating more efficient and accurate problem-solving capabilities.

Through extensive experimentation, we provide insights into the synergistic potential of LLMs and MCTS, showcasing improved performance in complex reasoning tasks.

This research advances the application of LLMs in sophisticated reasoning challenges. It sets the stage for future innovations in integrating AI technologies for enhanced decision-making, reasoning accuracy, and reliability of LLM-driven applications.

2 Preliminary

This section introduces the preliminary and notations used in this work. We will first detail the mechanism of Monte Carlo Tree Search (MCTS) and, essential for understanding the novel dynamic Monte Carlo Tree Self-refine

Monte Carlo Tree Search (MCTS) is a decision-making algorithm widely used in games and complex decision processes, which operates by building a search tree and simulating outcomes to estimate the value of actions. It involves four key phases (Browne et al., 2012): Selection, based on the UCT strategy to maximize the potential; expansion, where new nodes are added; simulation, to foresee possible outcomes; and Backpropagation, updating the node values based on simulation results. Typically, the MCTS algorithm comprises four distinct phases:

- **Selection:** Starting from the root, the algorithm navigates through promising child nodes based on specific strategies (e.g., UCT), continuing until a leaf node is reached.
- **Expansion:** At the leaf node, unless it represents a terminal state of the game, one or more feasible new child nodes are added to illustrate potential future moves.
- **Simulation or Evaluation:** From the newly added node, the algorithm conducts random simulations—often termed "rollouts"—by selecting moves arbitrarily until a game's conclusion is reached, thereby evaluating the node's potential.
- **Backpropagation:** Post-simulation, the outcome (win, loss, or draw) is propagated back to the root, updating the statistical data (e.g., wins, losses) of each traversed node to inform future decisions.

Repeatedly iterating through these stages, MCTS incrementally constructs a decision tree, refining strategies for optimal decision-making in scenarios where direct calculation of the best strategy is infeasible due to the vastness of the state space.

Upper Confidence Bound applied on Trees Algorithm is crucial for the selection phase in MCTS, balancing exploration and exploitation by choosing actions that maximize:

$$UCT_j = \bar{X}_j + C\sqrt{\frac{2\ln N_C}{N_j}} \tag{1}$$

Where [redacted], N_C is the total visited times of the father node, and n_j is the number of times node j has been visited for simulation, C is a constant to balancing exploitation and exploration.

MCT Self-Refine algorithm represents an integration of Monte Carlo Tree Search (MCTS) with large language models, abstracting the iterative refinement process of mathematical problem solutions into a search tree structure. Nodes on this tree represent different versions of answers, while edges denote attempts at improvement. This algorithm's operational workflow adheres to the MCTS algorithm's general pattern. Detailly, We employ self-reflective driven self-improvement for refining answers; rewards for different answer versions are sampled using the model's self-reward capability.

To facilitate understanding of the MCTSr algorithm, the following symbols and functions are defined:

- P: The problem instance being addressed.
- A: The set of nodes, each representing a potential answer to P.
- M: The set of actions available at each node, representing possible self-refine modifications to an answer.
- R [redacted] of th
- R_a: A Set that stores all self-rewards sampling results of node a with self-rewards function R.
- T: A function determining the termination of the search process based on criteria such as reaching a maximum number of iterations or achieving satisfactory answer quality.

- $Q(a)$: A value function estimating the worth of an answer node a, derived from accumulated rewards R_a and backpropagations from children nodes.
- $U(a)$: The Upper Confidence Bound for the Q value of node a to balance between exploitation and exploration.
- Father(a): A function returning the parent node of a given node a. If a is a root node, this function returns null or a specific identifier.
- [redacted]
- $N(a)$: The total number of visits to node a, used to calculate its UCB value and assess exploration and exploitation status. Since we will sample a reward for each visit, this value equals $|R_a|$.

3 Methodology

In this section, we will first demonstrate the main structure of MCTSr, shown in Figure 1. Then, we will detail each component The main workflow of MCTSr is structured as follows:

- **Initialization**: A root node is established using either a naive model-generated answer and a dummy response (e.g., 'I don't know.') to minimize model overfitting tendencies.
- **Selection**: The algorithm employs a value function Q to rank all answers that were not fully expanded and selects the highest-valued node for further exploration and refinement using a greedy strategy.
- **Self-Refine**: The selected answer a undergoes optimization using the Self-Refine framework (Madaan et al., 2023). Initially, the model generates a feedback m, guiding the refining process to produce an enhanced answer a'.
- **Self-Evaluation**: The refined answer is scored to sample a reward value and compute its Q value. This involves model self-reward feedback and constraints such as strict scoring standards and suppression of perfect scores to ensure reliability and fairness in scoring.
- **Backpropagation**: The value of the refined answer is propagated backward to its parent node and other related nodes to update the tree's value information. If the Q value of any child node changes, the parent node's Q is updated.
- **UCT update**: After the Q values of all nodes are updated, we identify a collection **C** of candidate nodes for further expansion or Selection, then use the UCT update formula to update the UCT values of all nodes for the next **Selection** stage.

The algorithm iterates through these stages until a termination condition T is met, including rollout constraints or maximum exploration depth, continuously refining the quality of answers, and exploring new possibilities.

3.1 Self-Refine

In the self-refine process, the model is guided by a multi-turn dialogue refine prompt to optimize an answer a to problem P. Initially, the model generates a reflective or critical comment m regarding a. Subsequently, guided by m, the model modifies a to produce an improved version a'. This iterative refinement enhances the quality of the response, leveraging structured feedback to drive the evolution of the answer.

3.2 Self-Evaluation

In the refining process for mathematical problem P, the Q value of an answer a is defined as the expected quality of further refining a into a superior answer, owing to the Markovian nature of the transition from a to its rewritten forms. Unlike traditional MCTS where $Q(s, a)$ estimates the value of action a in state s, $Q(a)$ here derives from multiple samplings of the reward function values attributed to a.

The model utilizes a self-reward method to estimate rewards for a, where it is required to provide a reward score ranging from -100 to 100. We find that without constraints, the model's reward tendency is overly smooth, leading to a lack of comparative distinction between answers in practice. To address this, three constraints are designed:

- **Prompt Constraint**: The model must adhere to the strictest standards during reward scoring.
- **Full Score Suppression**: The model is instructed not to provide full feedback scores; any reward above 95 is reduced by a constant amount to curb excessive scores.
- **Repeated Sampling**: Each visit to a search tree node involves the repeated sampling of the node's rewards to enhance the reliability of the Self-Evaluation. It should be noted that when reward sampling is performed on the child nodes of a node, we will also perform reward sampling on its parent node to increase the sample size of reward sampling.

Post sampling, the Q value of a is calculated. To counteract the smoothing tendency of the self-reward function, a minimum value constraint is added to the expected reward, further refining the estimation of answer quality,

$$Q(a) = \frac{1}{2}\left(\min R_a + \frac{1}{|R_a|}\Sigma_{i=1}^{|R_a|} R_a^i\right) \tag{2}$$

where $Q(a)$ is the quality value of answer a, R_a is the set of reward samples for a, $\min R_a$ is the minimum reward in R_a, $|R_a|$ is the number of samples, and $\Sigma_{i=1}^{|R_a|} R_a^i$ is the sum of all rewards in R_a. This formula calculates $Q(a)$ by averaging the rewards' minimum and mean, balancing worst-case and average outcomes.

3.3 Backpropagation

After all leaf nodes' reward value sampling and Q value update are completed, we will propagate this change to its parent and ancestor nodes. During this update process, if the Q function value of any element in the child node set Children(a) of a node a changes, the Q function value of the node is updated to

$$Q'(a) = \frac{1}{2}\left(Q(a) + \max_{i \in \text{Children}(a)} Q(i)\right) \tag{3}$$

Where $Q'(a)$ is the updated quality value of answer a that consider the impact from its children nodes, $Q(a)$ is the naive quality value only consider its reward samplings, and $\max_{i \in \text{Children}(a)} Q(i)$ represents the highest quality value among the children of a. This formula refines $Q(a)$ by averaging the current value and the best possible outcome from its subsequent Children nodes.

3.4 Update UCT and Selection

After updating the Q values across all nodes in the tree, we proceed to the selection phase for the next round of choices. This process includes the following steps:

Candidate Node Selection: Leveraging the Markovian nature of the mathematical problem refine process, we focus on selecting all leaf nodes and those that are not fully expanded, disregarding the history of refine paths is feasible. This path-independent property helps simplify our problem. We no longer need to start from the root node when selecting nodes but traverse the nodes in the tree in hierarchical order.

But given that Large Language Models (LLMs), which play as policy in this task, can generate an infinite number of refine actions m for any answer state a, each node potentially faces an unbounded set of actions for expansion. Thus, drawing from the concept of Expectation Improvement in Bayesian optimization, we propose two criteria for determining "full expansion":

- The node's children count reaches a predefined limit. And,
- At least one child node's Q value exceeds the node's.

This one's a good idea because you keep on trying until unless there is a better answer

We identify a collection $\mathbf{C}$ of candidate nodes based on these criteria for further expansion or Selection. This strategy helps accurately define which nodes might yield higher-value answers in subsequent searches, enhancing overall search efficiency and outcome quality.

UCT Update: Drawing from AlphaGo, we use UCT with the UCB-1 method to balance the exploration and exploitation of nodes; for node a in the candidate set $\mathbf{C}$, its UCT_a value is,

UCT(a) is dependent on Q(a) which in turn is dependent on sampling of Reward_Score(a)

$$UCT_a = Q(a) + c\sqrt{\frac{\ln N(\text{Father}(a)) + 1}{N(a) + \epsilon}} \tag{4}$$

where $Q(a)$ is the Q value of answer a, $N(\cdot)$ is the total visited times of given nodes, c is a constant to balancing exploitation and exploration, ϵ is a small constant for avoid devided-by-zero.

Sorting and Selection: According to the UCT value of the candidate set $\mathbf{C}$, we can select an optimal node to explore the refining process through greedy sampling or importance sampling.

3.5 Termination Function

In MCTSr algorithms, search termination function criteria T can derive from several conditions:

Early Stopping: Termination occurs when improvements in search results diminish or when consecutive searches yield repetitive outcomes.

Search Constraints: The search terminates once the number of rollouts reaches a predetermined limit or when one or more nodes in the tree satisfy the maximum depth constraint.

Advanced Criteria Based on Language Model Logits: The search concludes based on predefined metrics derived from the language model's logits.

Once the Termination Function condition T is satisfied, we can gather the best answers from tree nodes according to Q values or other conditions.

4 Evaluation

4.1 Experiment Settings

To assess the MCTSr algorithm's effectiveness in solving mathematical problems, we employed LLaMA3-8B (Meta AI, 2024) as the foundational model, enhanced with MCTSr. Detailed prompt settings are provided in the appendix. We compared LLaMA3-8 B's performance across several configurations—Zero-Shot CoT (Wei et al., 2022), Self-Refine, 4-rollouts MCTSr, and 8-rollouts MCTSr—against the performances of GPT-4 (Achiam et al., 2023), Claude 3 (Anthropic, 2024) and Gemini 1.5-Pro (Reid et al., 2024), which are the latest state-of-the-art closed-source models. These comparisons were conducted on various datasets, including GSM8K (Cobbe et al., 2021), GSM Hard (Gao et al., 2022), MATH (Hendrycks et al., 2021), AIME (AIME, 2024), Math Odyssey (AGI Odyssey, 2024), and OlympiadBench(pure-text subset) (He et al., 2024).

4.2 GSM Benchmarks

We evaluated the above methods on the test sets of GSM8K and GSM-hard, which involved typical and challenging mathematical problems, respectively. The results are shown in Table 1.

We can find that results reveal a direct correlation between the number of MCTSr rollouts and success rates, significantly improving as iterations increase, especially in the less complex GSM8K. However, the more intricate GSM-Hard set showcased a performance ceiling even at higher rollouts, indicating the limits of current strategies against complex problems.

These insights underscore the MCT-Self-refine algorithm's robustness and potential boundaries, highlighting the necessity for ongoing enhancements to tackle more complex challenges effectively. This work demonstrates the algorithm's capacity to enhance problem-solving performance and its varying efficacy across problem complexities, suggesting areas for future refinement in educational technology and automated reasoning.

Datasets	Zero-Shot CoT	One-turn Self-refine	4-rollouts MCTSr	8-rollouts MCTSr	Example Nums
GSM8K	977	1147	1227	1275	1319
	74.07%	86.96%	93.03%	96.66%	
GSM-Hard	336	440	526	600	1319
	25.47%	33.36%	39.88%	45.49%	

Table 1: Performance of MCTSr on the GSM Dataset

Level	Zero-Shot CoT	One-turn Self-refine	4-rollouts MCTSr	8-rollouts MCTSr	Example Nums
level-1	250	314	365	394	437
	57.21%	71.85%	83.52%	90.16%	
level-2	363	474	594	692	894
	40.60%	53.02%	66.44%	77.40%	
level-3	309	454	585	719	1131
	27.32%	40.14%	51.72%	63.57%	
level-4	202	368	523	656	1214
	16.64%	30.31%	43.08%	54.04%	
level-5	94	177	290	451	1324
	7.10%	13.37%	21.90%	34.06%	
Overall	1218	1787	2357	2912	5000
	24.36%	35.74%	47.14%	58.24%	

Table 2: Performance of MCTSr on the MATH Dataset

4.3 MATH Benchamark

This section presents the outcomes of applying the MCT-Self-refine (MCTSr) algorithm across various complexity levels on the MATH dataset. The dataset is stratified into five levels of difficulty, ranging from level 1 (easiest) to level 5 (most challenging). The algorithm's performance is evaluated using four distinct configurations: Zero-Shot CoT, One-turn Self-refine, 4-rollouts MCTSr, and 8-rollouts MCTSr. Each configuration's efficacy is measured by the number of successfully solved problems and the corresponding success rates, with a total of 5000 examples across all levels.

Level-1 results demonstrate the highest success rates, with the 8-rollouts MCTSr achieving a remarkable 90.16% success rate, solving 394 out of 437 problems. This level shows a clear progression in success rates as rollouts increase.

At the most challenging level-5 part, the 8-rollouts MCTSr configuration yields a 34.06% success rate, solving 451 out of 1324 problems. This illustrates the increasing difficulty and the algorithm's strained performance in highly complex scenarios.

Overall performance across all levels shows a cumulative success rate of 58.24% with the 8-rollouts MCTSr, solving 2912 out of 5000 problems. This rate demonstrates a substantial enhancement from the Zero-Shot CoT's initial rate of 24.36%. The data indicates a consistent trend where the increase in rollouts correlates with improved success rates, underlining the efficacy of the MCT-Self-refine algorithm in enhancing problem-solving capabilities across varying levels of mathematical complexity.

These results validate the MCT-Self-refine algorithm's potential in academic and problem-solving contexts and highlight its scalability and adaptability to different levels of problem complexity within the MATH dataset.

4.4 Olympiad-level Benchmarks

The efficacy of the MCT-Self-refine (MCTSr) algorithm was tested on three datasets from mathematical Olympiad competitions: AIME, GAIC Math Odyssey, and OlympiadBench. The GAIC Math Odyssey dataset, released in April 2024, is notable for its minimal overlap with the pre-training corpus of the LLaMa3-8B model, providing a robust test of the algorithm's ability to generalize.

Datasets	Zero-Shot CoT	One-turn Self-refine	4-rollouts MCTSr	8-rollouts MCTSr	Example Nums
AIME	22	41	70	110	933
	2.36%	4.39%	7.50%	11.79%	
Math Odyssey	67	118	156	192	389
	17.22%	30.33%	40.10%	49.36%	
OlympiadBench	16	39	67	99	1275
	1.25%	3.06%	5.25%	7.76%	

Table 3: Performance of MCTSr on Olympiad-level Datasets

	Gemini 1.5-Pro	Claude 3 Opus	GPT-4 Turbo
MATH (Reid et al., 2024)	67.7	60.1	73.4
Math Odyssey (Reid et al., 2024)	45.0	40.	49.1
GSM8K (Papers with Code, 2024)	94.4	95	97.1

Table 4: closed-source LLM performance on mathematical datasets

AIME: From Zero-Shot CoT's 2.36% (22 problems solved) to 8-rollouts MCTSr's 11.79% (110 problems solved).

GAIC Math Odyssey: Showed substantial improvement, starting at 17.22% (67 problems solved) and reaching up to 49.36% (192 problems solved) with 8-rollouts MCTSr.

OlympiadBench: Improved from 1.25% (16 problems solved) in Zero-Shot CoT to 7.76% (99 problems solved) in 8-rollouts MCTSr.

The results demonstrate a clear trend where increased rollouts correlate with higher success rates, highlighting the algorithm's potential to improve performance through iterative refinement. The GAIC Math Odyssey results mainly reflect the MCTSr's generalization capabilities in new environments.

These findings affirm the MCT-Self-refine algorithm's robustness and its utility in tackling complex, unseen mathematical problems, suggesting its applicability in educational technologies aimed at competitive academic settings like Olympiads.

4.5 Disscussion

We investigated the reported values of the current state-of-the-art closed-source large model SOTA performance on the above test benchmarks, as shown in the table 4. Comparing the performance of current closed-source large models, MCTSr can effectively enhance the mathematical reasoning capabilities of small-parameter open-source models, like LLaMa-3, to a comparable level.

5 Related works

Monte Carlo Tree Search (MCTS) has been widely utilized in various fields to solve complex problems efficiently. Pitanov et al. (2023) explored the application of MCTS in Multi-agent Pathfinding, demonstrating its superiority over heuristic search algorithms like A*. Additionally, Yang (2023) integrated MCTS with heuristic, unsupervised, and supervised learning methods to efficiently solve the Train Timetabling Problem (TTP). Furthermore, Li et al. (2023) introduced a general method for solving various types of SAT problems using a unified framework incorporating MCTS. Vagadia et al. (2024) developed PhyPlan, a physics-informed planning framework that combines physics-informed neural networks with modified MCTS to enable robots to perform dynamic physical tasks effectively. In conclusion, MCTS has proven to be a versatile and effective mathematical solution for solving various complex problems in different domains, including robotics, game solving, and optimization. Researchers continue to explore and enhance the capabilities of MCTS by integrating it with other algorithms and frameworks to tackle increasingly challenging tasks.

Recent research has made notable strides in enhancing mathematical reasoning in large language models (LLMs). Du et al. (2023) introduced a method where multiple LLMs discuss and refine answers collectively, significantly boosting reasoning and factual accuracy. Luo et al. (2023) developed WizardMath, which leverages Reinforcement Learning from Evol-Instruct Feedback to surpass existing LLMs in mathematical benchmarks. Meanwhile, Lu et al. (2023) created MathVista, a visual, mathematical benchmark, with GPT-4V achieving a 49.9% accuracy, highlighting gaps that persist relative to human performance. Yu et al. (2023) introduced MetaMath, a fine-tuned model that excels in mathematical challenges, and Yuan et al. (2023) demonstrated that pre-training loss and Rejection sampling Fine-Tuning can optimize LLM performance, particularly in less advanced models. These studies suggest significant progress yet underline the necessity for ongoing research in LLM mathematical reasoning.

Recent advancements in large language models (LLMs) have significantly improved their mathematical reasoning abilities. Yet, they still face complex problems that require multiple reasoning steps, leading to logical or numerical errors. To address this limitation, Chen et al. (2024) proposed incorporating Monte Carlo Tree Search (MCTS) to enhance the mathematical reasoning capabilities of fine-tuned LLMs without additional fine-tuning steps . Xu (2023) utilized MCTS and a lightweight energy function, the models can rank decision steps and enable immediate reaction and precise reasoning, leading to improved performance on mathematical reasoning benchmarks. However, it still lacks a framework that combines the self-refine capabilities and self-reward evaluation method of LLMs to refine the model's response iteratively with the Monte Carlo Tree Search Algorithm.

6 Limitations

Although the MCTSr algorithm has demonstrated certain advantages in mathematical tasks, our research is still in its preliminary stages. As a general decision-making framework, the potential applications of MCTSr in various scenarios remain to be explored further, such as in black-box optimization problems and self-driven alignment for large language models. Additionally, the components of MCTSr are highly scalable, necessitating ongoing development to identify and compare a broader range of component algorithms, thereby enhancing the practical potential and effectiveness of the MCTSr algorithm.

7 Conclusion

This paper demonstrates the effectiveness of the MCT Self-Refine (MCTSr) algorithm in enhancing the capability of Large Language Models (LLMs) to solve complex mathematical problems. By integrating Monte Carlo Tree Search (MCTS) with LLMs, MCTSr addresses critical challenges in accuracy and reliability, particularly within mathematical reasoning tasks. Experimental results confirm significant improvements in problem-solving success rates across multiple datasets, including notable performance in Olympic-level mathematical challenges.

Moreover, the research advances the application of LLMs in sophisticated reasoning tasks and lays the groundwork for future integration of AI technologies to enhance decision-making and reasoning accuracy. Despite MCMCTSr'semonstrated potential in mathematical problem-solving, its applicability in broader contexts, such as black-box optimization and self-driven alignment, remains to be explored. Future work will optimize algorithmic components and test their performance across various problems and settings to achieve broader practicality and effectiveness.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AGI Odyssey (2024). AGI odyssey.

AIME (2024). AIME problem set: 1983-2024.

Anthropic, A. (2024). The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card.*

Browne, C., Powley, E. J., Whitehouse, D., Lucas, S. M. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Liebana, D. P., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43.

Chaslot, G., Bakkes, S. C. J., Szita, I., and Spronck, P. (2008). Monte-carlo tree search: A new framework for game ai. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Chen, G., Liao, M., Li, C., and Fan, K. (2024). Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*.

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

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. (2023). Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. (2022). Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

He, C., Luo, R., Bai, Y., Hu, S., Thai, Z. L., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., Liu, J., Qi, L., Liu, Z., and Sun, M. (2024). Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. (2021). Measuring mathematical problem solving with the MATH dataset. *CoRR*, abs/2103.03874.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., and Liu, T. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ArXiv*, abs/2311.05232.

Li, A., Han, C., Guo, T., Li, H., and Li, B. (2023). General method for solving four types of sat problems. *arXiv preprint arXiv:2312.16423*.

Liang, Y., Zhang, R., Zhang, L., and Xie, P. (2023). Drugchat: Towards enabling chatgpt-like capabilities on drug molecule graphs. *ArXiv*, abs/2309.03907.

Lu, P., Bansal, H., Xia, T., Liu, J., Li, C., Hajishirzi, H., Cheng, H., Chang, K.-W., Galley, M., and Gao, J. (2023). Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*.

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

Lyu, H., Jiang, S., Zeng, H., Xia, Y., and Luo, J. (2023). Llm-rec: Personalized recommendation via prompting large language models. *ArXiv*, abs/2307.15780.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Welleck, S., Majumder, B. P., Gupta, S., Yazdanbakhsh, A., and Clark, P. (2023). Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Meta AI (2024). Introducing meta llama 3: The most capable openly available LLM to date.

Papers with Code (2024). Papers with code - GSM8k benchmark (arithmetic reasoning).

Pitanov, Y., Skrynnik, A., Andreychuk, A., Yakovlev, K., and Panov, A. (2023). Monte-carlo tree search for multi-agent pathfinding: Preliminary results. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 649–660. Springer.

Reid, M., Savinov, N., Teplyashin, D., Lepikhin, D., Lillicrap, T., Alayrac, J.-b., Soricut, R., Lazaridou, A., Firat, O., Schrittwieser, J., et al. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2009). Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58:3250–3265.

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

Vagadia, H., Chopra, M., Barnawal, A., Banerjee, T., Tuli, S., Chakraborty, S., and Paul, R. (2024). Phyplan: Compositional and adaptive physical task reasoning with physics-informed skill networks for robot manipulators. *arXiv preprint arXiv:2402.15767*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., hsin Chi, E. H., Xia, F., Le, Q., and Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903.

Xu, H. (2023). 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*.

Yang, F. (2023). An integrated framework integrating monte carlo tree search and supervised learning for train timetabling problem. *arXiv preprint arXiv:2311.00971*.

Yu, L., Jiang, W., Shi, H., Yu, J., Liu, Z., Zhang, Y., Kwok, J. T., Li, Z., Weller, A., and Liu, W. (2023). Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

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

A Prompts in Experiment

A.1 Self-Refine

Get Feedback:

USER: Since we have a weak Answer, could you provide me with a relection or feedback to correct this answer better? Analyze this Answer Strictly and Critic, point out every flaw for ervery possible imperfect to minus every possible score!
Let's think step by step.

Get Refined Answer:

USER: Please refine the your answer according to your Reflection or Feedback. The response should begin with [reasoning process]...[Verification]... and end with end with "[Final Answer] The answer is [answer formula]"
Let's think step by step.

A.2 Self-Reward

USER: Question: question
Answer: ans
Analyze this Answer Strictly and Critic, and point out every flaw for every possible imperfect to minus every possible score! You need to be very harsh and mean in calculating grades, and never give full marks to ensure that the marks are authoritative.
Output a score between [-100,+100], ig. from -100 to +100.
Response format:
[Analyst]...[Score]...

A.3 Dummy Answers

USER: ["I Don't Know","I can't understand this question.","I can't help with this question.","I don't know how to solve this question.","I don't know the answer to this question.","I don't know the answer to this question, sorry."]