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

Paper

- Accessing GPT-4 Level Mathematical Olympiad Solutions via Monte Carlo Tree Self-Refine with Llama-3 8B

Summary

- Introduces MCT Self-Refine, an LLM with Monte carlo tree search, to enhance mathematical reasoning
- Tree is constructed through selection, self-refine, self-evaluation, and backpropagation
- Utilizes Upper Confidence Bound

Introduction

- Llms still face challenges in logical reasoning in areas
- Llms still hallucinate in math
    - The paper introduces rewriting techniques like Self-refine
- Self-Refine integrates LLms with monte carlo search
- There are existing challenges in combining mcts + llm
    - Traditional mcts strats might not align well with stochastic nature of llm's → a infinite continuous space of potential actions
    - This is solved with backpropagation and a pruning strategy with upper confidence bound
- Primary contributions

- 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.

Preliminary

Monte carlo tree search

- Decision making algorithm used in games
- Uses 4 key 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 constructs a decision tree

Upper confidence bound on trees

- This algorithm is in the selection phase in MCTS, balancing exploration/exploitation

$$UTC_j = \bar{X}_j + C\sqrt{\frac{2ln\,N_c}{N_j}}$$

- $\bar{X}_j$ is the average reward of action j
- N_c is the total visited times of father node
- N_j is the number of times node j has been visited for simulation
- C is the constant to balancing exploitation and exploration

MCT Self-refine

- The part where they integrate MCTS with llms
- They turn iterative refinement process of mathematical problem into search tree
- 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.
    - We employ self-reflective driven self-improvement for refining answers; rewards for different answer versions are sampled using the model's self-reward capability.
- Symbols are defined
    - P - problem instance being addressed
    - A - set of nodes, each is a potential answer to P
    - M - set of actions at each nodes, ie possible self-refine modifications to an answer
    - R - samples self-rewards for nodes based on modification quality
    - R_a - set that stores all self rewards of node a with above function
    - T - function determining termination of search process based on criteria like reaching max number of iterations
    - Q(a) = a value function estimating worth of answer node a, derived from rewards set R_a
    - U(a) - upper confidence for Q value of node a to balance exploitation and exploration
    - Fater(a) - function reeturning parent node of given node
    - Children(a) - function return child nodes
    - N(a) - total number of visits to node, used to calculate UCB value

Methodology

- Main flow of MCTSr goes like this
    - Initialization - root node created using model generated answer or dummy response (ex: 'i don't know')
    - Selection - uses a value function Q to rank all answers and selects highest valued node (greedy approach)

    - Self-refine - selected answer goes through self refinement. Model generates feedback m so model can produce a better answer a'
    - Self-evaluation - refined answer a' is scored to get a reward value and to compute its Q value.
    - Backpropagation - value of the refined answer is propagated backward to parent node to update related nodes' information
    - UCT update - after the Q values of all nodes are updated, we identify a collection C of candidate nodes for further selection and use the UCT update formula to update the UCT values of all nodes and repeat the selection stage
- This flow happens until termination condition T is met

Self-Refine

- Here, the model is guided by a prompt to optimize an answer *a* to a problem *P*

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.

- Initially, the model generates a reflective or critical comment *m* regarding the answer *a*.
- Then, using *m,* the modell modifies *a* to make a better answer *a'*

Self-Evaluation

- Here, we get the Q value of an answer a which is the expected quality or score of refining *a* into a better answer.
- Is different from traditional MCTS here. In traditional MCTS, Q(s,a) estimates value of action a in state s. Q(a) here derives from multiple samplings of the reward function values attributed to *a*
- Model uses self reward method to estimate rewards for *a* (ranging from -100 to 100)
- The self reward output is too smooth, so these constraints are applied to it
    - 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.

- After sampling, the Q value of *a* is calculated and we do another thing to combat the smoothing of the reward function→ use a min value

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

- Basically Q(a) is quality value of answer *a*
- R_a is set of reward samples for *a*
- Min R_a is the minimum reward
- |R_a| is the number of samples
- $\sum_{i=1}^{|R_a|} R_a^i$ is the sum of all rewards in R_a
- Formula averages rewards's minimum and mean

Backpropagation

- Happens after leaf node reward value sampling and Q values are updated
- Propagates this change to parent and ancestors
- Formula works by averaging current value and best possible outcome from children nodes

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

Update UCT and Selection

- After updating Q values, we go into selection phase for next set of nodes

Candidate node selection

- They focus on selecting all leaf nodes and those that are not fully expanded, disregarding the history of refined paths is feasible.
    - Can do this because path-independent property helps simplify our problem. → the idea that all that matters is in the current state signal which is seen in other rl problems
- We no longer need to start from the root node when selecting nodes but traverse the nodes in the tree in hierarchical order.
- 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.
- We identify a collection 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

- Markovian means "It means **the current state has everything you need to know what's going on and the best way** to proceed"
- Inspired from AlphaGo
- UCT with UCB-1 method
- Calculated for a node *a* in candidate set **C**
- We select an optimal node to explore based on the UCT value

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

Termination Function

Termination function criteria T can be

- **Early stopping** - stop when you get diminishing search results or multiple searches give you the same results
- **Search constraints** - search terminates once rollouts reach a limit or nodes reach a depth
- **Advanced criteria based on language model logits** - predefined metrics from language model's logits

Evaluation

- Used llama 3-8b as the foundational model and enhanced it with MCTSr
- Performs better than gemini 1.5 pro and claude 3 opus, and gpt4 turbo

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).
- 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.
- 8 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,
- 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.

Limitations

- Still preliminary research
- MCTSr needs to be explored in this scenarios
    - black-box optimization problems and self-driven alignment for large language models.
    - the components of MCTSr are highly scalable so more research needed to make it scale

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.