Improving LLM Tool Using Planning Based on

Reinforcement Learning

Final Report

Pinqi FENG\*, Yuan CHEN\*, Yingbei ZHONG\*  
 DSA Thrust of Info Hub  
 The Hong Kong University of Science and Technology (Guangzhou), China  
{pfeng915, ychen151, yzhong632}@connect.hkust-gz.edu.cn

ABSTRACT

Currently, a crucial research direction for LLM Agent is to enable large models to utilize tools to execute tasks like humans do. This capability involves expecting large models to autonomously use numerous tools defined by natural language and automatically plan their usage behavior paths to achieve various task objectives. However, the further development of this capability is limited by the problem of finding reasonable behavior paths. Existing methods typically rely on state space search to find paths that achieve the target. But, for the vastness of the state space, naive or heuristic searches lack a global perspective, leading to suboptimal solutions or planning failures.

In this report, we attempt to introduce the Monte Carlo Tree Search (MCTS) algorithm in reinforcement learning into the decision-making process of the Agent, so that it can optimize its decision-making through the learnt experience in its execution and ensure its completeness when considering various tools and strategies. We refer to the framework of LATS, which employs LLMs as agents, value functions, and optimizers, and builds a tree structure cyclically through four steps: node selection, extended simulation, reflective evaluation, and backpropagation, and retains the optimal paths after backpropagation. This framework, which enhances its own decision-making process through the use of external feedback, achieves good performance in our tests, and further improves on the DFSDT (Depth-First Search-based Decision Tree) algorithm in the ToolBench framework.

KEYWORDS

Large Language Models, Reinforcement Learning, Tool Learning

1  INTRODUCTION

The LLM Agent aims to construct a system with complex reasoning capabilities, memory, and execution means. Significantly, enabling large models to autonomously select and use tools is a key to achieving this goal, a research area commonly referred to as Tool Learning. Tool Learning primarily focuses on enabling models to understand the concept of tools and to apply them flexibly to solve problems. Models trained in tool learning can identify tools, understand when and how to use specific tools, and, when necessary, combine multiple tools to accomplish more complex tasks.

Three primary research directions in tool learning can be identified: 1) Enabling models to understand the concept of tools (creating a reasonable dataset structure); 2) Enabling models to identify and correctly use specific tools (training for the selection and use of specific tools); 3) Behavior path planning for using multiple tools in complex scenarios.

Current research in tool learning mainly focuses on the first two directions, i.e., enabling models to understand the functions of tools and acquiring the ability to use them. However, for behavior path planning in complex scenarios, which is crucial for practical applications, the current approach still largely relies on the LLM's capabilities combined with simple or heuristic search methods. This approach often leads models to choose suboptimal solutions or results in planning failures. This paper will concentrate on the optimization of this aspect of the capability. We propose introducing reinforcement learning into the decision-making training process to train models' dynamic planning abilities.

In the tool learning framework, the model's use of tools to complete tasks is a recursive interaction process involving humans, the large model, a toolset, tool executors, and environmental sensors. This recursive process continues until the instructions are satisfied and the final answer is produced. The model needs to make next-step decisions based on feedback, a mechanism very similar to that in reinforcement learning, where continuous trials are conducted to achieve goals. Through ongoing optimization of the decision-making process, the model gradually discovers and implements the best processing path to achieve the final result, which might be an effective strategy.

2  LITERATURE REVIEW

2.1 Tool Learning

Tool learning is a critical aspect of LLMs, enabling them to interact with and manipulate external information sources and environments. Recent research [1] has explored the concept of foundation models, which are pre-trained models capable of general tasks, and their ability to learn and utilize tools effectively. These models are seen as a steppingstone towards more autonomous and capable AI systems.

图示

描述已自动生成

**Figure 1: Description of tool learning**

Research [10] highlights the importance of aligning AI systems with human feedback to improve tool learning. This collaboration is essential for creating systems that can generalize human-like decision-making [3]. The ability to scale tool learning algorithms to handle a large number of real-world APIs [7] and tasks is a significant area [6] of research. This scalability is crucial for the practical application of LLMs in various industries.

The development of frameworks that enable LLMs to learn a wide array of tools [6] is a current focus. These frameworks aim to increase the versatility of LLMs across different domains and tasks [7]. Despite the advancements, there are challenges that need to be addressed, such as the scalability of learning algorithms [8], the integration of human feedback [9], and the computational efficiency of the learning process.

To solve those challenges, there is a growing interest in applying reinforcement learning (RL) to guide tool learning within LLMs [2,3]. RL's ability to optimize actions through rewards and punishments makes it a suitable candidate for improving decision-making processes. Interactive systems that combine planning with tool learning have been proposed to enhance the flexibility and effectiveness of LLMs [4]. These systems allow for a more dynamic and responsive approach to problem-solving [5].

2.2 LLM Reasoning

For large language models (LLMs), reasoning typically involves decomposing complex inputs into sequential intermediate steps towards a final answer (Cobbe et al., 2021). However, these methods, which create reasoning chains autoregressively in a single step, often suffer from error propagation as the number of steps increases (Guo et al., 2018; Chen et al., 2022b) due to compound errors. The accumulation of small errors in each step can lead to significantly incorrect final results.

To address this challenge, various advancements have been proposed. The Self-Consistency (Wang et al., 2022), aim to improve robustness by employing majority voting over multiple sampled chains, thereby reducing the impact of individual erroneous chains. This method leverages the statistical properties of multiple outputs to converge on a more reliable answer.

Other approaches focus on multi-step decomposition to handle complex tasks more effectively. For example, least-to-most prompting (Zhou et al., 2022) decomposes tasks into simpler sub-tasks, solving them in a hierarchical manner. Additionally, external tools like a scratchpad (Nye et al., 2021) or compiler (Gao et al., 2022) have been used to assist the LLMs in maintaining and using intermediate states, thus reducing the cognitive load on the models and improving accuracy.

图片包含 地图

描述已自动生成

**Figure 2: Comparison of LLM reasoning methods**

Recently, enhancements to CoT have been achieved through the incorporation of sophisticated search algorithms (Yao et al., 2023a; Hao et al., 2023; Besta et al., 2023) that can sample reasoning trajectories more effectively. These algorithms aim to explore the search space more thoroughly and systematically, thereby identifying more accurate solutions. For instance, Tree-of-Thought (ToT) prompting (Yao et al., 2023a) uses depth-first search (DFS) (Qin et al., 2023) or breadth-first search (BFS) guided by heuristics generated by the language model, which helps in systematically exploring possible reasoning paths. Another approach, Reasoning via Planning (RAP) (Hao et al., 2023), employs Monte Carlo Tree Search (MCTS) with rollouts simulated by the language model, enabling it to evaluate multiple potential outcomes before making a decision.

Despite these advancements, it's important to note that many of these methods rely solely on the internal knowledge of LLMs and may not adapt well to external feedback. This limitation underscores the need for future research to integrate external information sources and feedback mechanisms, which could provide more contextual and accurate responses.

3  PRELIMINARIES

3.1 ToolBench Pipeline

Qin et al. (2023) introduced ToolBench, a framework that establishes a systematic workflow enabling large models to answer questions through the invocation of multiple APIs. The ToolBench framework involves five key steps: API Retrieval, State Initialization, Task Definition, Planning Process, and Execution.

图示

描述已自动生成

**Figure 3: The structure of ToolBench**

**API Retrieval** The first step involves understanding the user's query, which is the input to our system. Using a semantic understanding of the query, a retriever is employed to find the top K APIs that are most relevant to the user's question. This retrieval process typically involves comparing word vector similarities to match the query with the descriptions or functionalities of available APIs. The retrieved APIs are then placed into a collection for further consideration.

**State Initialization** Once the relevant APIs have been identified, the LLM agent is equipped with access to this collection of candidate API functions, denoted as {API0, API1, …, APIk}, where k represents the number of APIs that closely relate to the user's query.

**Task Define** The agent is given a natural language task description via prompt, including other prompts to instruct the LLM. The agent's goal is to translate the task description into a sequence of API function calls S = {a0, a1, …, am}, where m represents the number of API calls necessary to accomplish the users’ task.

**Planning Process** Treating the task description as the initial state, the agent generates a plan by considering the API definitions and demonstration samples. This process involves selecting API calls that form a coherent strategy to achieve the task described.

**Execution** Finally, the agent executes the plan to produce an outcome. The execution is guided by a plan executor. This step involves running the sequence of API calls to see if they lead to the desired task completion.

3.2  Monte Carlo Tree Search algorithm

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm used for some of the most complex decision processes, particularly those involving game playing and planning under uncertainty.

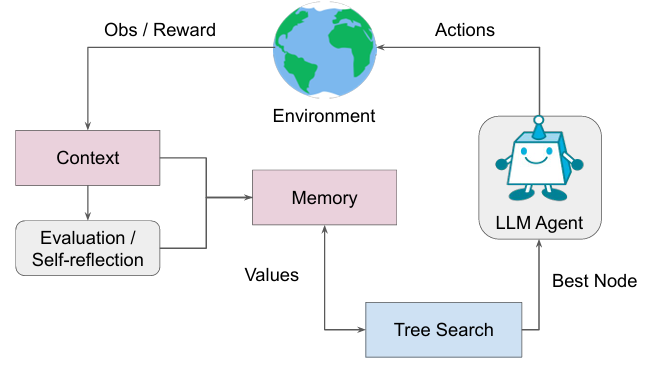
As an extension of the Monte Carlo method., MCTS builds a tree of potential futures, using random sampling to guide its expansion. Starting from the root, the algorithm selects the most promising child node to expand, typically using the Upper Confidence Bound for Trees (UCT) formula to balance exploration and exploitation. The algorithm expands the tree by adding one or more child nodes to the selected node. These nodes represent unexplored actions or states. From the newly expanded node, the algorithm simulates random play-out to a terminal state to estimate the value of the node. This is often done using a simple heuristic or random policy. The results of the simulation are used to update the values of the nodes along the path from the root to the expanded node.

To apply MCTS to tool learning in LLMs, we can define the state as the current context and the history of API calls made so far. This includes the environment status, any intermediate results, and the goal state the LLM is trying to achieve. The actions are the possible API calls that can be made. These need to be generated based on the current state and the documentation or descriptions of available APIs. The LLM acts as a transition model, predicting the next state given the current state and an action (API call). This could involve calling the API with the LLM's current understanding of the environment and parameters. Design a reward function that evaluates the desirability of a state. This could be based on how close the current state is to the goal state, the logical coherence of the API call sequence, or the immediate outcome of an API call.

For the search strategy, we can use UCT or a similar strategy to select the most promising API call based on the current state. Expand the search tree by simulating the effects of the chosen API call and adding the resulting state as a child node. For tool learning, a simulation could involve calling the actual API to observe the outcome or using the LLM to predict the outcome based on the API's documentation. Then update the value of the node and its ancestors with the reward obtained from the simulation. The search stops when a pre-set number of iterations have been completed, or when the goal state is reached.

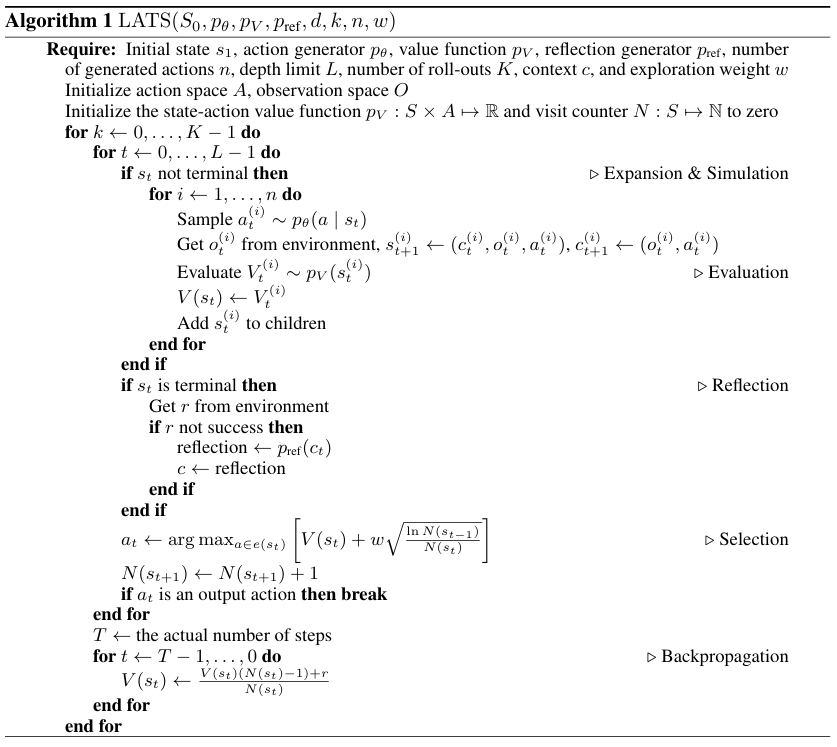
3.3 Language Agent Tree Search

While extensive research has focused on incorporating external feedback to enhance Large Language Models' (LLMs) reasoning capabilities, there has been limited application to the optimization of the reasoning process itself. To address this gap, Zhou et al. (2024) proposed the LATS framework, which leverages Monte Carlo Tree Search (MCTS) from model-based reinforcement learning. This framework transforms a pre-trained LLM into an agent, a value function, and an optimizer that constructs optimal paths through carefully designed action samples. Utilizing the advanced natural language understanding and contextual learning abilities of modern LMs, LATS uses text as an interface between its components, enabling it to adapt to environmental changes without additional training. Furthermore, LATS incorporates internal introspection, allowing agents to reflectively learn from practice, thereby enhancing the model's self-awareness and sensitivity to its own behavior.



**Figure 4: An overview of LATS**

The LATS framework operates through six key steps: Selection, Expansion, Evaluation, Simulation, Backpropagation, and Reflection. Initially, a promising node is selected using the Upper Confidence Bound for Trees (UCT) algorithm. The model then expands the tree by generating (n) actions from the current node, adding new child nodes to an external long-term memory repository. Each new node is evaluated based on its progress towards the task, guiding future selection and backpropagation. The selected node is expanded until reaching a terminal state, at which point the path's correctness is assessed. If successful, the search halts; otherwise, backpropagation updates the values of nodes based on simulation outcomes. Reflection optimizes decision-making by documenting errors and proposing improved strategies for failed paths, providing rich semantic feedback for future attempts and enhancing learning through contextual adaptation without the high costs of reinforcement learning.



**Figure 5: The psuedocode of LATS**

The LATS framework, which integrates planning, action, and reasoning strategies into Language Models (LMs), has shown promising results in test validation. Based on the LATS framework, GPT-4 has achieved significant improvements in various domains, including interactive question answering (HotPotQA) and web navigation (WebShop), and even achieved state-of-the-art performance on the HumanEval programming task.

4 EXPERIMENTS

4.1 Experimental Setup

4.2 Tool-use: ToolBenchQA

4.3 Reflections

1.

2. Other Approach: The A\* (A-star) algorithm

The simulation process of Monte Carlo Tree Search (MCTS) evidently incurs significant resource consumption, which to some extent limits the practical applicability of the framework. To address this issue, Zhuang et al. (2023) introduced ToolCHAIN\*, an innovative approach that integrates the A\* algorithm into the decision-making process of Large Language Models (LLMs). This method employs a heuristic path-search function to evaluate potential actions generated by the model, selecting the next action based on the criterion of minimizing total path cost. This direct evaluation approach aims to circumvent the computational expense associated with the multiple simulations required by MCTS. The efficacy of this method has been validated on the GSM8K dataset and within SambaNova's ToolBench environment. Looking ahead, we can consider incorporating this algorithm to enhance the operational efficiency of our framework and potentially achieve performance improvements.

图示

描述已自动生成

**Figure 5: Illustration of the process using A\* Search**

5 Conclusion

In summary, the project has identified and addressed several key challenges in integrating reinforcement learning (RL) with large language models (LLMs) in tool learning. The primary issue lies in the complexity of the state space, which traditional search algorithms struggle to navigate efficiently, often leading to suboptimal solutions or planning failures. Additionally, the project highlighted the inefficiency in tool usage, the limitations of conventional training methods, and the need for algorithms that can generalize across diverse business scenarios without extensive retraining.

To overcome these challenges, the project proposed a sophisticated algorithmic structure that enhances the completeness of LLMs by considering a wide array of tools and strategies. The framework optimizes decision-making through a history of learned experiences, allowing for continuous improvement and adaptation. Efficiency is achieved by employing a targeted search strategy, reducing computational load and time investment.

The project compares several innovative approaches, including the DFSDT (Depth-First Search-based Decision Tree) algorithm, which enhances reasoning capabilities by generating distinct nodes to diversify the search space. The A\* Search Algorithm is adapted for efficient planning and decision-making within LLMs, balancing effectiveness with costs. Furthermore, the application of Monte Carlo Tree Search (MCTS) to tool learning in LLMs represents a significant advancement in navigating the vast space of possible actions to find a sequence that leads to a successful outcome.

Despite the advancements, there are still several limitations. The resource-intensive nature of RL poses a hurdle, necessitating further optimization for efficient function within the constraints of large-scale LLMs. Additionally, the generalization of the algorithm across different business scenarios remains a challenge, requiring ongoing research to ensure minimal retraining or fine-tuning. It also recognizes the need for further research to improve the scalability of learning algorithms, integrate human feedback more effectively, and enhance computational efficiency.

REFERENCES

[1] Qin, Y., Hu, S., Lin, Y., Chen, W., Ding, N., Cui, G., ... & Sun, M. (2023). Tool learning with foundation models. arXiv preprint arXiv:2304.08354.

[2] Zhuang, Y., Chen, X., Yu, T., Mitra, S., Bursztyn, V., Rossi, R. A., ... & Zhang, C. (2023). Toolchain\*: Efficient action space navigation in large language models with a\* search. arXiv preprint arXiv:2310.13227.

[3] Qiao, S., Gui, H., Chen, H., & Zhang, N. (2023). Making Language Models Better Tool Learners with Execution Feedback. arXiv preprint arXiv:2305.13068.

[4] Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., & Liang, Y. (2023). Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. arXiv preprint arXiv:2302.01560.

[5] Wang, B., Fang, H., Eisner, J., Van Durme, B., & Su, Y. (2024). LLMs in the Imaginarium: Tool Learning through Simulated Trial and Error. arXiv preprint arXiv:2403.04746.

[6] Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., & Sun, L. (2023). Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. arXiv preprint arXiv:2306.05301.

[7] Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., ... & Sun, M. (2023). Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789.

[8] Li, L., Chai, Y., Wang, S., Sun, Y., Tian, H., Zhang, N., & Wu, H. (2023). Tool-Augmented Reward Modeling. arXiv preprint arXiv:2310.01045.

[9] Zeng, A., Liu, M., Lu, R., Wang, B., Liu, X., Dong, Y., & Tang, J. (2023). Agenttuning: Enabling generalized agent abilities for llms. arXiv preprint arXiv:2310.12823.Conference Short Name:WOODSTOCK’18Conference Location:El Paso, Texas USA

[10] Yuan, Z., Yuan, H., Tan, C., Wang, W., Huang, S., & Huang, F. (2023). Rrhf: Rank responses to align language models with human feedback without tears. arXiv preprint arXiv:2304.05302.

[11] Hao, S., Gu, Y., Ma, H., Hong, J. J., Wang, Z., Wang, D. Z., & Hu, Z. (2023). Reasoning with language model is planning with world model. arXiv preprint arXiv:2305.14992.

[12] Tian, Y., Peng, B., Song, L., Jin, L., Yu, D., Mi, H., & Yu, D. (2024). Toward Self-Improvement of LLMs via Imagination, Searching, and Criticizing. arXiv preprint arXiv:2404.12253.

[13] Ding, R., Zhang, C., Wang, L., Xu, Y., Ma, M., Zhang, W., ... & Zhang, D. (2023). Everything of thoughts: Defying the law of penrose triangle for thought generation. arXiv preprint arXiv:2311.04254.

[14] Yang, S., Nachum, O., Du, Y., Wei, J., Abbeel, P., & Schuurmans, D. (2023). Foundation models for decision making: Problems, methods, and opportunities. arXiv preprint arXiv:2303.04129.

[15] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35, 24824-24837.

[16] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[18] Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., ... & Hadsell, R. (2015). Policy distillation. arXiv preprint arXiv:1511.06295.

[19] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. arXiv preprint arXiv:1611.05397.

[20] Finn, C., Abbeel, P., & Levine, S. (2017, July). Model-agnostic meta-learning for fast adaptation of deep networks. In International conference on machine learning (pp. 1126-1135). PMLR.