Empowering Large Language Model Agent through Step-Level Self-Critique and Self-Training

Yuanzhao Zhai*
yuanzhaozhai@nudt.edu.cn
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Huanxi Liu*
liuhuanxi18@nudt.edu.cn
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Zhuo Zhang
iezhuo17@gmail.com
Harbin Institute of Technology (Shenzhen); Peng Cheng Laboratory
Shenzhen, China

Tong Lin
lintong23@nudt.edu.cn
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Kele Xu
kele.xu@ieee.org
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Cheng Yang
delpiero710@126.com
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Dawei Feng†
davyfeng.c@qq.com
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Bo Ding
dingbo@nudt.edu.cn
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Huaimin Wang
hmwang@nudt.edu.cn
National University of Defense Technology; State Key Laboratory of Complex & Critical Software Environment
Changsha, China

Abstract

Large Language Model (LLM) agents frequently produce sub-optimal actions when tackling complex, multi-step decision-making tasks. Employing self-critique to identify flaws and suggest enhancements is an effective strategy for refining actions. Although trajectory-level critique is commonly employed, it often fails to identify flawed steps accurately. In this paper, we introduce **SLSC-MCTS**, a method that integrates Monte Carlo Tree Search with Step-Level Self-Critique to enhance LLM agents during both testing and self-training phases. During decision tree expansion with SLSC-MCTS, the LLM agent initially generates an action, receives environmental feedback, and subsequently generates further actions via self-critique and refinement. Through multiple episodes of SLSC-MCTS, LLM agents can effectively utilize step-level critiques while disregarding ineffective ones based on node values, thereby incorporating the critiques more robustly. Additionally, our method further empowers LLM agents in a self-training manner, collecting training data from the constructed decision tree to iteratively fine-tune the LLM agents. The self-training data gathered via SLSC-MCTS is diverse and high-quality, which further enhances the reasoning, critiquing, and refining abilities of LLM agents. Experimental results demonstrate that SLSC-MCTS significantly improves LLM agents during testing, surpassing state-of-the-art baselines and achieving shorter task completion trajectories across information retrieval benchmarks such as WebShop and HotPotQA. After three iterations of self-training, LLM agents established by Llama-3.1-8B-Instruct show substantial improvement, even surpassing human experts in WebShop.

CCS Concepts

• **Information systems** → **Retrieval tasks and goals**; • **Computing methodologies** → **Natural language processing**.

Keywords

Large Language Models; Intelligent Agent; Conversational Information Retrieval; Multi-Step Reasoning

ACM Reference Format:
Yuanzhao Zhai, Huanxi Liu, Zhuo Zhang, Tong Lin, Kele Xu, Cheng Yang, Dawei Feng, Bo Ding, and Huaimin Wang. 2025. Empowering Large Language Model Agent through Step-Level Self-Critique and Self-Training. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25), July 13–18, 2025, Padua, Italy.* ACM, Padua, Italy, 11 pages. https://doi.org/10.1145/3726302.3729965

*These authors contributed equally.
†Corresponding author.

SIGIR '25, July 13–18, 2025, Padua, Italy

ACM ISBN 979-8-4007-1592-1/2025/07
https://doi.org/10.1145/3726302.3729965

1 Introduction

Large Language Model (LLM) agents [1] have recently shown considerable promise in tasks that directly benefit users, such as retrieval-enhanced question-answering [2] and web applications [3, 4], thereby opening new avenues for conversational information retrieval [5]. In these tasks, LLM agents engage with environments over multiple steps [6] by calling retrieval tools, and receive a reward upon task completion. Despite notable advancements, even advanced LLM agents encounter difficulties in managing complex multi-step decision-making tasks [4], as suboptimal actions at individual steps can accumulate errors, ultimately leading to task failure [7].

Existing step-level methods for LLM agents [8–10] primarily focus on selecting the optimal action at each step from multiple candidates. These candidate actions are typically generated in parallel through temperature-based sampling, leading to a tradeoff between quality and diversity. Since LLMs are expected to identify mistakes and provide remediation automatically, LLM agents can generate natural language critiques to refine their actions based on feedback from environments. Prior self-reflection methods [11, 12] generate critiques of the LLM itself based on previous trial trajectories, leveraging past mistakes to guide the next trial. However, trajectory-level critiques can be inefficient because they may fail to pinpoint the exact step where the issue occurred (see Figure 1). This highlights the necessity of step-level self-critique for LLM agents.

While recent works such as OpenAI o1 [13] and Deepseek R1 [14] has shown great success in incorporating step-level self-critiques for LLM reasoning, our preliminary experiments indicate that directly applying them to LLM agents can be counterproductive. This is because self-critique tasks are inherently difficult [15] and plenty of critiques are ineffective, which can degrade the overall performance [16]. This challenge is further exacerbated by the limited critique capabilities of LLM agents in complex scenarios. However, existing works primarily focus on enhancing the reasoning ability of LLM agents [17–19]. Improving the self-critique ability of LLM agents is challenging since the sparse rewards of agent tasks make it difficult to determine the effectiveness of step-level critiques.

To address the challenges mentioned above, we propose a framework combining Step-Level Self-Critique and Monte Carlo Tree Search (SLSC-MCTS). The core idea of SLSC-MCTS is utilizing MCTS to balance the exploration and exploitation of LLM agents with step-level self-critique, and automatically generating self-training data for enhancing both the reasoning and critique abilities of LLM agents. Specifically, we leverage critique and refinement mechanisms to generate refined actions based on previous actions and environmental feedback, which promotes diversity while preserving quality when generating multiple candidate actions. With multiple episodes of SLSC-MCTS, step-level critique will be exploited if it is effective and neglected otherwise. In addition, after constructing the decision tree, we collect the self-training dataset to enhance LLM agents' abilities of both reasoning, step-level critiquing and refining. The data-collection and self-training processes can be alternated over multiple iterations, leading to continuous improvements in the performance of LLM agents with SLSC-MCTS.

We perform experiments on two retrieve-based agent tasks. In the web-shopping task, LLM agents retrieve and navigate detailed product listings across multiple web pages, selecting products based on relevance to user queries. In question-answering tasks, LLM agents answer complex questions by retrieving and integrating information from large databases Wikipedia. Extensive experiments suggest that SLSC-MCTS can significantly boost the task completion rate of LLM agents by effectively leveraging step-level self-critique, achieving state-of-the-art performances on both tasks. Besides, LLM agents with SLSC-MCTS can find shorter successful trajectories, demonstrating the improvement of planning efficiency. After three self-training iterations with collected self-training data with high quality and diversity, we observe significant performance gains. Notably, LLM agents established by Llama3.1-8B-Instruct show impressive capabilities, achieving a remarkable success rate of 63.5% in WebShop, which even surpasses the performance of human experts of 59.6%.

In summary, our main contributions are as follows:

- We propose SLSC-MCTS, which combines step-level self-critique with MCTS, enabling LLM agents to perform effective step-level reasoning, critique and refining. To the best of our knowledge, we are the first to introduce step-level critiques to empower LLM agents.
- Based on the high-quality and diverse data collected by SLSC-MCTS, we propose a multi-round iterative self-training paradigm, which enhances the reasoning, step-level critiquing and refining abilities of LLM agents.
- Extensive experiments demonstrate the effectiveness of SLSC-MCTS, which outperforms the state-of-the-art in terms of task completion and successful trajectory length. Furthermore, the collected high-quality and diverse step-level annotated data can continuously improve LLM agents through multiple rounds of iterative self-training.

2 Related Work

2.1 Large Language Model Agents

As LLMs continue to evolve, there is growing research interest in LLM agents capable of interacting with their environment to perform a wide range of tasks. In many information retrieval (IR) scenarios, such agents interact with external tools like search engines [2, 3] and carry out multi-step actions to address complex tasks. Recently, numerous studies have explored methods to enhance LLMs' outputs through test-time computation, particularly after the release of the OpenAI O1 series of models [13]. Most of the work has focused on mathematical reasoning tasks. Snell et al. [20] categorizes the test-time computation for LLM reasoning into sequential revisions and parallel sampling. We mainly summarize the multi-step decision-making work in the LLM agent scenarios.

Developing mechanisms for critique, reflection, and correction is essential for addressing complex, multi-step decision-making tasks in LLM agents. Reflection-based methods, such as Reflexion [11] and Self-Refine [12], provide critical feedback that allows agents to learn from previous mistakes. Self-verification [21, 22] employs prompting techniques to generate self-diagnosed feedback, while rStar [23] introduces self-play mutual reasoning method, utilizing models to assess and refine each other's results iteratively and ultimately selecting the most consistent answers. However, the correction performance of these methods is often limited by the LLM's inherent capabilities.

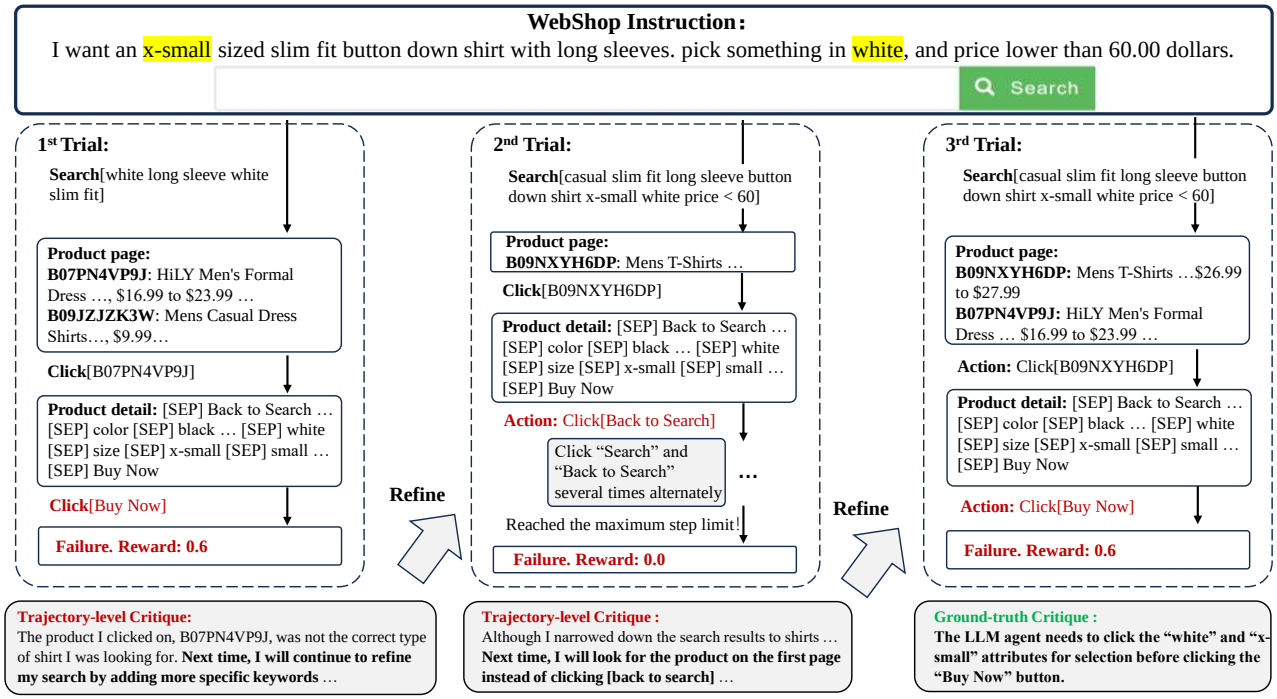

Figure 1: An example of inefficient trajectory-level self-critiques and refinements when employing the Llama-3.1-8B-Instruct LLM agent in the WebShop task [3]. In the 1st Trial, the LLM agent failed to select the options "white" and "x-small" on the product detail page to meet the constraints in the instruction, resulting in a final reward of 0.6 out of 1. However, the trajectory-level critique for the 1st Trial failed to point out these missteps, leading to a new dead-loop problem in the 2nd Trial. In the 2nd Trial, the critique shifted focus to the newly introduced dead-loop problem, but the original problem of unmet constraints remained unresolved in the 3rd Trial.

Recently, Monte Carlo Tree Search (MCTS) [24] and other tree-based search algorithms has been demonstrated effective for LLM to improve planning and reasoning [25–29]. Through simulation sampling, MCTS constructs a decision tree and evaluates each possible action, thereby selecting the optimal next move. For example, Zhou et al. [8] integrate agents with MCTS, using LLM-powered value functions and prompt mechanisms like reflection.

2.2 Self Training of Large Language Model

General LLMs can be fine-tuned on task-specific agent data before being encapsulated as agents [17–19, 30]. However, this process often depends on high-quality expert trajectories, which are costly and limited in quantity. To address the problem, recent studies [10, 31–33] have used self-training techniques, where LLMs are trained on the data collected by themselves. When combining with reject sampling fine-tuning (RFT) [34], these methods can significantly improve performance.

Model collapse is one of the critical challenges when training models on self-generated data [35, 36], especially in multi-round iterations due to the limited quality and diversity of self-generated data. This issue has been observed in various settings including RFT and direct policy optimization (DPO), where it manifests as reduced output diversity [37, 38]. Previous studies [31, 39] have constructed Chain of Thought (CoT) data, which enhance diversity by enriching the reasoning content of model. Recent works [10, 32, 40] have broadened the scope of exploration by integrating MCTS and other search algorithms. However, these search algorithms, using temperature-based sampling, have difficulties in maintaining a balance between data quality and diversity (as illustrated in Figure 4). Our method introduces a self-critique mechanism in MCTS to enhance the diversity of training data, while maintaining high quality.

2.3 Model-Based Feedback

In order to address the inefficiencies and hard-to-reach features of real feedback, model-based feedback is widely used to pursue efficient feedback. Notably, outcome reward models (ORMs) can be trained on the preference dataset [41], which is more efficient than human feedback. The recently advanced reward models can be utilized to distinguish between desirable and undesirable responses, particularly in complex multi-step reasoning contexts. Compared to outcome reward models (ORMs) that are trained based on the final results of responses, process reward models (PRMs) [42, 43] are often deemed more effective, as they offer step-wise reward signals for process supervision.

Existing reward models usually directly produce a numerical continuous-valued score without a detailed explanation for each step. As LLMs advance in their capabilities across diverse domains, numerous studies have explored model-based feedback by having LLMs critique their generated responses with verbal feedback [44, 45]. These approaches highlight the potential of LLMs in critiquing and refining ability. To the best of our knowledge, we are the first to apply step-level self-critique to LLM agents.

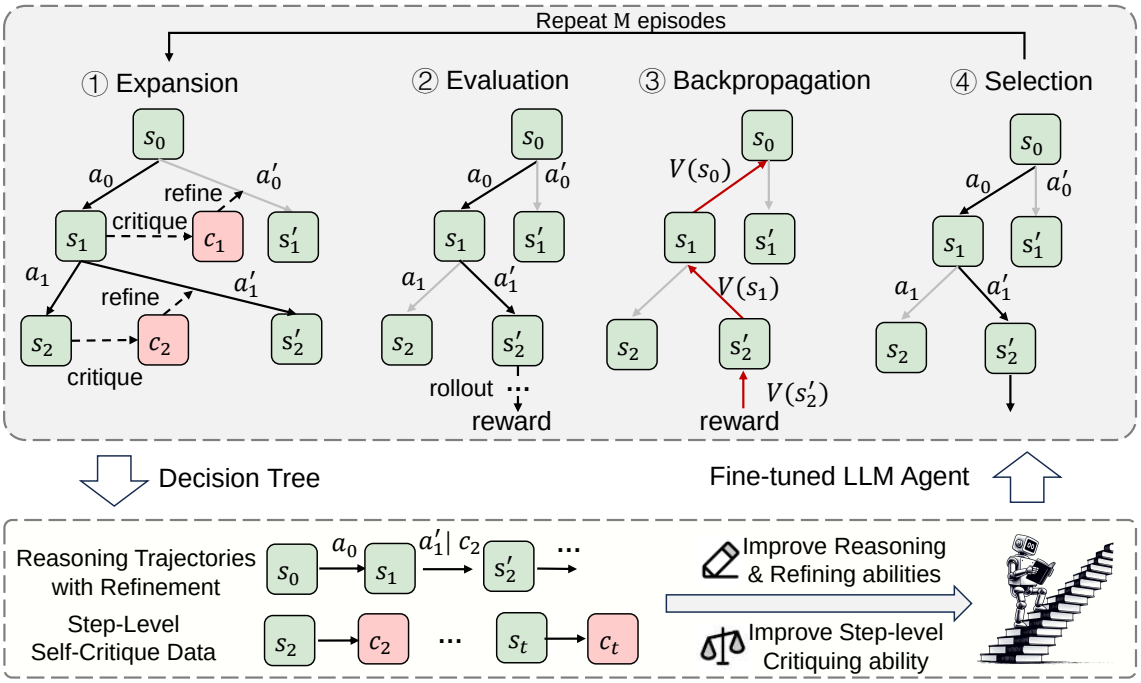

Figure 2: Illustration of SLSC-MCTS enhancing LLM agents through two alternating phases performed over multiple iterations: Data collection (Top) and Self-training (Bottom). In the top part, an example iteration of constructing the decision tree is presented. For simplicity, the diagram displays the expansion of only two nodes at each level of the tree. In the bottom part, the LLM agents are fine-tuned to learn from the reasoning trees constructed by SLSC-MCTS.

3 Preliminaries

The agent task with environment feedback is formalized as a partially observable Markov decision process (POMDP) $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, r)$, where $\mathcal{U}$ represents the instruction space, $\mathcal{S}$ the state space, $\mathcal{A}$ the action space, $\mathcal{O}$ the observation space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the state transition function, and r the reward function. Since the environment is partially observable, we can estimate the current state s_t utilizing the concatenation of the task instruction u, the interactive history, and other available information that can represent the state (such as step-level critique feedback c_t if exists).

Given a task instruction $u \in \mathcal{U}$, the LLM agent generates an initial action $a_0 \sim \pi(\cdot|u)$ based on its policy π. The state transitions to $s_1 \in \mathcal{S}$, and the agent receives an observation $o_1 \in \mathcal{O}$. This process continues as the agent interacts with the environment until either the task is successfully completed or the maximum number of steps is reached.

At each time step t, given the historical interactions $\tau_t = (a_0, o_1, ..., a_{t-1}, o_t)$, the agent produces the next action $a_t \sim \pi(\cdot|u, \tau_t)$. The multi-step decision-making task can therefore be defined as:

$$\pi(\tau|u) = \prod_{t=0}^{T} \pi(a_t|u, \tau_t) \tag{1}$$

where we denote τ as the whole trajectory, T as the total interaction steps. Note that the environment only provides the outcome reward $r(u, \tau) \in [0, 1]$.

The objective of LLM agents is to maximize rewards from the environment:

$$\max_{\pi} \mathbb{E}_{u \sim \mathcal{D}, \tau \sim \pi(\cdot|u)} \left[r(u, \tau) \right], \tag{2}$$

where $\mathcal{D}$ represents the dataset containing task instructions.

4 Methodology

We first present our proposed SLSC-MCTS that introduces the step-level self-critique mechanism into "tree search" for effective and efficient reasoning-path searching. We then describe the iterative self-training of LLM agents with SLSC-MCTS in terms of data construction and model training. Further details will be elaborated in the following subsections.

4.1 Step-Level Self-Critique Monte Carlo Tree Search

We construct a decision tree in which each node represents the current state of LLM agents, and each edge corresponds to an action. Each node maintains a set of statistics:

$$V(s_t), N(s_t), \tag{3}$$

where $V(s_t)$ denotes the value function, representing the expected reward obtainable from the subtree rooted at s_t. The term $N(s_t)$ indicates the number of visits to the node s_t.

The core idea of SLSC-MCTS is to leverage step-level natural language critique during node expansion, with the goal of identifying effective reasoning paths leading to correct answers in an iterative manner. Our SLSC-MCTS process starts from a root node s_0 and progresses through four iterative stages: expansion, evaluation, backpropagation and selection, as shown in the top part of Figure 2.

Expansion. The expansion operation is the main difference between SLSC-MCTS and other MCTS variants. Due to the infinite action space of LLM agents, existing methods expand nodes based on multiple high-temperature sampling to avoid falling into local optimal solutions. However, excessive temperature will affect the quality of the generated actions, leading to a trade-off between sample efficiency and the diversity of expanded nodes.

During the expansion of SLSC-MCTS, for the first node of each expansion, the LLM agent directly generates actions $a_t = \pi(u, \tau_t)$. For subsequent sibling nodes that are expanded later, we propose a two-step approach: self-critique and refine. Given s_t, the LLM agent can output a self-critique $c_t = \pi(u, \tau_t)$ which contains two types of information: discrimination and feedback. Discrimination evaluates the effectiveness of the action, while feedback provides a detailed explanation of the underlying reasoning. The prompt is shown in Figure 3, where the red text part needs to be replaced by real-time generated content. Upon receiving the critique, the LLM agent generates a revised action $a'_t = \pi(u, \tau_t, c_{t+1})$, a process referred to as conditional refinement. This process can be repeated up to $w - 1$ times, resulting in the expansion width of w. Compared to MCTS, our critique-based expansion achieves higher diversity while also maintaining sample quality.

Evaluation. In LLM agent tasks, the decision tree's depth is typically much shallower than that of Go games, resulting in rapid expansions that frequently reach terminal nodes. Unlike AlphaGo [46], which employs a value network to evaluate state nodes, we evaluate expanded nodes using a rollout algorithm. Beginning from the expanded nodes, the LLM agent interacts with the environment until it reaches either a terminal state or the maximum rollout depth. If a terminal node is encountered, the reward provided by the environment is returned. Otherwise, nodes at the maximum depth are assigned a fixed negative reward.

Backpropagation. Backpropagation updates the tree's statistics based on the outcome rewards or fixed negative rewards determined during the evaluation stage. For each node along the trajectory τ, the visit count $N(s_t)$ is incremented by 1, and the value is propagated and updated from the terminal node s_T to the root node s_0 using the formula:

$$V(s_t) \leftarrow \frac{V(s_t)(N(s_t) - 1) + r(u, \tau)}{N(s_t)}. \tag{4}$$

The updated values are subsequently used in the Upper Confidence bounds applied to Trees (UCT) [47] to guide the selection of the next node:

$$UCT(s_t) = V(s_t) + \sqrt{\frac{\eta \ln N\big(p(s_t)\big)}{N(s_t)}}, \tag{5}$$

where η is the exploration weight, and $p(s_t)$ denotes the parent node of s_t.

Selection. The purpose of the selection operation is to identify the most promising trajectory for the next expansion. Starting from the root node, a trajectory is traced down to the current leaf node. At each level of the tree, the child node with the highest UCT value is selected, ensuring a balance between exploration and exploitation.

Through multiple episodes of expansion, evaluation, backpropagation and selection, the final decision tree can be constructed. There are three data types in the decision tree: reasoning, self-critiquing, and refining data.

4.2 Self-Training SLSC-MCTS Pipeline

We can collect self-critique data to improve the critique ability using the constructed decision tree. Additionally, the output of reasoning and refining data is action a_t, which contains thoughts and actions following the ReAct style [6]. This output can be utilized to improve the overall reasoning ability of LLM agents.

Collect reasoning trajectories with refinement. We adopt a trajectory-level approach to construct the dataset, aiming to improve the reasoning and refinement abilities of LLM. Specifically, we find a leaf node with the highest outcome reward $r(u, \tau)$ based on the decision tree. Then we trace back from the leaf node to the root node. During the backtracking process, if a node is the left-most among its sibling nodes, the critique is absent at this tree level. In this case, we retain the original path $< u, \tau_t, a_t >$. Otherwise, critique information is incorporated, resulting in $< u, \tau_t, c_{t+1}, a_t >$ in each decision-making step. In this way, the whole trajectories contain both basic reasoning and refining data. To ensure the quality of the trajectories, we follow [48] to filter the trajectories based on task completion status and only keep those trajectories with an outcome reward $r(u, \tau) > \beta$.

Collect step-level critique data. The critique data is constructed at the step level. After M episodes of SLSC-MCTS, the value of each node can represent the potential to complete the task. Therefore, we can evaluate critiques according to the value difference before and after critique and refining $\Delta V(s'_t) = V(s'_t) - V(s_t)$. We designed a two-layer filtering mechanism for collecting step-level self-critique data. Similar to collecting reasoning trajectories, we filter the critique according to the outcome reward $r(u, \tau) > \beta$. More importantly, we filter critiques that have a positive value difference $\Delta V(s'_t) > 0$, indicating that the critiquing and refinement process plays a positive role in the completion of the final task.

LLM Agents Self-training with Step-Level Self-Critique. After constructing the dataset containing trajectory-level reasoning and step-level critiquing, we fine-tune the LLM agent with the Rejection sampling Fine-Tuning (RFT) algorithm [34], which has gained notable attention for its simplicity and scalability. With the curated dataset and the LLM agent π_θ, standard supervised fine-tuning (SFT) can then be performed to improve the reasoning and refinement abilities, as follows:

$$\mathcal{L}_{\text{reasoning}} = \begin{cases} \mathbb{E}_{(u,\tau_t,a_t)} \log \pi(a_t|u, \tau_t), & \text{if } c_t \text{ does not exist,} \\ \mathbb{E}_{(u,\tau_t,c_{t+1},a_t)} \log \pi(a_t|u, \tau_t, c_{t+1}), & \text{if } c_t \text{ exists.} \end{cases} \tag{6}$$

At the same time, the self-critiquing ability can be improved via:

$$\mathcal{L}_{\text{critique}} = \mathbb{E}_{(s_t,c_t)} \log \pi(c_t|s_t). \tag{7}$$

The overall loss of self-training can be formulated as a combination of improving both policy and critique capabilities:

$$\mathcal{L} = \mathcal{L}_{\text{reasoning}} + \mathcal{L}_{\text{critique}}. \tag{8}$$

The data-collection and fine-tuning processes can be repeated across multiple iterations, leading to a progressively refined dataset and improved LLM agents. Note we only need a task instruction set

Prompt for Self-Critiquing

Below are the previous Thought and Action you generated along with their corresponding Observation:
<Previous Thought and Action>
<Previous Observation>
Review the previous Thought, Action, and Observation. Your role is to determine whether the action is effective for completing the task, and provide specific and constructive feedback. Please output feedback directly.
Feedback:[[Feedback]]

Prompt for Refining

Below are the previous Thought and Action you generated along with their corresponding Observation:
<Previous Thought and Action>
<Previous Observation>
<Critique>
Based on the critique, generate a new Thought and Action with as much distinctiveness as possible.

Figure 3: Prompts of LLM agents for self-critiquing and refining.

to collect the self-training dataset. We utilize the same task instruction set for different iterations. In each iteration, the self-training data, containing reasoning, self-critique and refining data, are generated with iteratively improved LLM agents. To prevent overfitting and ensure the quality of iteratively collected self-training data, we only retain the sample with the highest outcome reward.

5 Experimental Results

5.1 Experimental Setup Details

WebShop. WebShop tasks the agent with solving shopping tasks by navigating websites with detailed product descriptions and specifications. The available action APIs include Search[QUERY] for using the search bar and Click[BUTTON] for clicking buttons on web pages. Clickable buttons include product titles, options, buy, back-to-search, and previous/next page, among others. When the agent selects the "Buy Now" action, the environment provides an outcome reward ranging from 0 to 1 determined by matching heuristics of the attributes and price of the product.

HotPotQA. HotPotQA is a question-answering task that requires retrieval across multiple Wikipedia passages. Following the setup in [6], LLM agents are equipped with API calls for searching (Search[INF-ORMATION]) and retrieving (Lookup[INFORMATION]). Upon receiving an answer, the environment provides a binary outcome reward of 0 or 1 based on its correctness according to the ground truth. Consistent with previous work [6, 8, 11], we adopt an oracle setup for HotPotQA, where the environment provides feedback on the correctness of the answer as soon as it is received.

Evaluation Metrics. In WebShop, we use *Average Reward* and *Success Rate* to measure the performance of agents, and the task is considered solved only when the outcome reward is 1. For HotPotQA, the evaluation is confined to the Exact Match metric with a binary reward. To measure the agent's efficiency in both scenarios, we use the *Trajectory Length*, which is the length of the maximum reward trajectory searched.

Implementation Details. In our experiments, we use Llama3.1-8B-Instruct[1] as our backbone model. When collecting data and evaluating, we utilize the model from the final epoch in each iteration and set the sampling temperature to 1.0 unless explicitly stated. During the self-training stage, we set the base learning rate to $5e-6$ with a cosine scheduler and the number of epochs to 3. For WebShop, we follow the dataset split used in Song et al. [48] and filter out samples from the training set that are low-quality and repeated in the test set, resulting in 1800 and 200 samples in the training and test set, respectively. For HotPotQA, we randomly choose 1000 unique question indices for training and 100 for testing. For both tasks, another 100 samples are selected for validation.

[1]https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

5.2 Baselines

We perform comprehensive evaluations against several strong baselines, including: 1) **ReAct** [6] is an interactive style that allows LLM to explicitly construct a complete series of actions based on logical reasoning to achieve expected goals. This interaction style is also adopted by all baselines involved in our experiments. 2) **Step-Level Self-Critique** generates the critique for each step and then subsequently refines, without integrating MCTS. 3) **Reflexion** [11] agents verbally reflect on trajectory-level feedback signals, then generate critiques in an episodic memory buffer to guide better decision-making in subsequent trials. 4) **MCTS** [49] involves directly applying the traditional heuristic search algorithm MCTS to LLM agents, balancing the exploration and exploitation, where the nodes are expanded based on the temperature-based sampling. 5) **LATS** [8] is an MCTS-based method, and additionally incorporates trajectory-level reflection on the failure trajectory into the interaction history, guiding the subsequent episode of MCTS. 6) **LLM Agent with Q** [9] introduces a process reward model, i.e., Q-value model, that scores LLM's multiple candidate actions in each step and selects the highest-scoring one.

When introducing self-training, we filter samples with high outcome rewards as high-quality data for fine-tuning LLMs, which can be combined with MCTS and SLSC-MCTS.

Hyperparameters. For RFT, we set the filtering threshold $r(u, \tau) > 0.7$. For reflection-based methods, i.e., Reflexion, Step-Level Self-Critique and LATS, we set the maximum number of reflections to 3. For MCTS-based methods (MCTS, LATS, and SLSC-MCTS), we set the expansion width to 3, the maximum MCTS episodes $M = 30$. For LLM Agent with Q, the width of beam search is also set to 3 and the lookahead ahead is set to 1, following Zhai et al. [9].

5.3 Main Results

Comparison with Baselines. We present the main results in Table 1, where self-training is performed for up to three iterations, and we choose the best-performing checkpoint on the valid set. As shown, SLSC-MCTS outperforms all baselines in both datasets by introducing more test-time computation. Overall, our method

Table 1: Performance comparisons across two LLM agent datasets. Compared to existing works, we additionally record the trajectory length metric to evaluate the search efficiency, which indicates the length of successful trajectories.

Method	WebShop			HotPotQA	
	Average Reward ↑	Success Rate ↑	Trajectory Length ↓	Exact Match ↑	Trajectory Length ↓
Test-time Computation Only (Llama-3.1-8B-Instruct)					
ReAct [6]	0.42	10%	4.07	30%	3.47
Step-Level Self-Critique	0.12	3%	5.60	12%	3.79
Reflexion [11]	0.48	18%	4.47	49%	3.76
MCTS	0.71	27%	4.52	70%	2.94
LATS [8]	0.69	27%	5.08	73%	3.21
LLM Agent with Q [9]	0.70	30%	4.63	57%	3.26
SLSC-MCTS (Ours)	0.74	29%	**4.15**	72%	**2.43**
Self-Training + Test-time Computation (Fine-tuned Llama-3.1-8B-Instruct)					
MCTS + Self-Training	0.79	46%	4.42	70%	2.86
SLSC-MCTS + Self-Training (Ours)	**0.87**	**64%**	4.34	**78%**	2.47
Reference Results					
GPT-4-Turbo	0.46	29%	4.62	44%	3.48
Human Expert [3]	0.82	60%	-	-	-

demonstrates substantial advantages over MCTS and and achieves significant performance improvements. In WebShop, our method achieved a success rate of 64% (+35%) and an average reward of 0.87 (+0.13) outperforming MCTS which achieves a success rate of 46% (+19%) and an average reward of 0.79 (+0.08). SLSC-MCTS also surpasses those of the Human (expert) at 0.82 and 60%, as well as the success rate of 41.5% of the state-of-the-art (SOTA) method AgentQ [10]. This performance **not only outperforms state-of-the-art baselines such AgentQ+MCTS (with the success rate of 50.5%) with advanced LLM agents, but also exceeds human experts on this task for the first time**, to the best of our knowledge. Similarly in HotPotQA, SLSC-MCTS raises the exact match from 72% to 78%, whereas MCTS shows no significant improvement. In comparison, the performance of the SOTA model, GPT-4-Turbo, falls significantly short of our method, with a success rate of 29% in WebShop and 44% in HotpotQA.

Compared to ReAct, reflection-enhanced and other MCTS-based methods improve the performance effectively, demonstrating the effectiveness of critique and MCTS. For instance, the Reflexion agent obtains an exact match of 49% (+19%) in HotPotQA, and the LLM agent with Q obtains 0.70 (+0.28) on the average reward in Webshop. However, we can observe that there is a slight performance degradation of LATS compared to MCTS in WebShop, demonstrating that trajectory-level critique is insufficient to facilitate task completion when using tree search algorithms. Additionally, our experiments reveal that simply adopting step-level self-critiques can be counterproductive, whose success rates in WebShop and HotPotQA at only 3% and 12%, respectively, significantly lower than those of ReAct and Reflection.

Furthermore, our method improves the planning efficiency of agents, as evidenced by a decrease in trajectory length. For example, compared to MCTS, the trajectory length of SLSC-MCTS decreased by 0.37 and 0.54 in WebShop and HotPotQA, respectively. Nevertheless, after self-training, the trajectory length obtained by our method increased to 4.34. This is due to the completion of more complex tasks, which inherently require longer trajectories.

Table 2: Statistical information of the collected self-training data in different iterations.

	Total Node Number ↑	Retention Rate ↑	Self-BLEU ↓
MCTS			
1st Iteration	32.05	28%	0.52
2nd Iteration	27.16	41%	0.51
3rd Iteration	24.40	44%	0.54
SLSC-MCTS			
1st Iteration	51.71	37%	0.21
2nd Iteration	50.05	48%	0.21
3rd Iteration	46.76	52%	0.25

5.4 Results of Multi-round Self-Training

We analyze the efficiency of our approach relative to MCTS in terms of data collection and self-training across multiple iterations in WebShop.

Quantity and quality of self-training data. For collected training data, we analyze the fundamental information of the decision tree with regard to the total node number and the retention rate after filtering by outcome rewards greater than 0.7, which can indicate the quantity and quality of the self-training data. In addition, we utilize Self-BLEU [50] to assess the diversity among expanded nodes, where a lower Self-BLEU score means higher diversity. The statistical information of self-training data is shown in Table 2. We found that the total node number in MCTS was much lower than in our method, as its most extended nodes are deduplicated due to the same parsed actions. We further calculate the averaged node number during the expansion, which is 2.01, 1.94, and 1.89 with MCTS and 2.69, 2.69, and 2.60 with SLSC-MCTS. These results explain why the decision tree constructed by SLSC-MCTS contains a greater number of extracted nodes.

Moreover, the nodes generated by SLSC-MCTS are not only more numerous but also of higher quality and diversity, as evidenced by the higher retention rate and lower Self-BLEU scores. As mentioned in Section 4.1, collecting self-training data with MCTS involves a trade-off between diversity and quality, as demonstrated in Figure 4 (where values on the x-axis represent 1- Self-BLEU). In contrast,

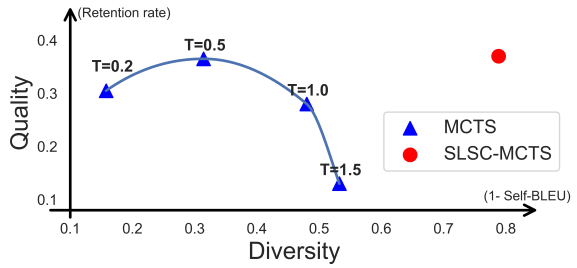

Figure 4: The trade-off between quality and diversity of collected self-training data.

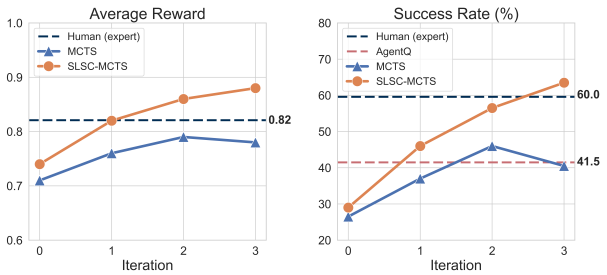

Figure 5: Performance improvement across three self-training iterations.

the data collected by SLSC-MCTS can achieve higher diversity and quality simultaneously.

Performance of multiple self-training iterations. The performance across multiple self-training iterations is presented in Figure 5. The observations are as follows: (1) In comparison with MCTS, our method achieved significant performance improvements, with an average reward increase of 0.09 and a 17.5% gain in success rate. This highlights the effectiveness of incorporating step-level self-critique into MCTS. (2) While MCTS initially shows performance improvements in the early iterations, it quickly plateaus or even begins to decline by the third iteration. In contrast, our method shows consistent improvement across three rounds of iterative self-training, without overfitting.

5.5 Ablation Studies

We conduct additional experiments on WebShop to demonstrate the effect of each component of SLSC-MCTS.

Ablation Study on Numbers of Self-training Data. To evaluate the effectiveness of our method under limited self-training samples, we reduced the sample size to 1000 and constructed a training dataset to fine-tune the agents. Besides, we added two scenarios for comparison: without training and 1800 samples. The evaluation results shown in Table 3 reveal that: (1) our method consistently outperforms MCTS when using the same number of samples for self-training. Notably, MCTS uses 1000 samples for sampling and self-training, but negatively impacts the test set performance, with a 6% decrease in success rate. This is attributed to the fact that collecting too few nodes can lead to overfitting during self-training, as we demonstrated in Table 2. In comparison, collecting data with 1000 samples using SLSC-MCTS demonstrates a stable enhancement, progressing from 29% to 39% in success rate. (2) Compared

Table 3: Effect of the sample numbers on self-training.

Training Data	Average Reward ↑	Success Rate ↑	Trajectory Length↓
MCTS			
w/o training	0.71	27%	4.52
1000 samples	0.68	21%	4.35
1800 samples	0.76	37%	4.03
SLSC-MCTS			
w/o training	0.74	29%	4.15
1000 samples	0.78	39%	3.80
1800 samples	0.82	46%	3.90

Figure 6: Effects of different data types for self-training.

to vanilla MCTS, LLM agents with SLSC-MCTS can find shorter trajectories, indicating improved better planning efficiency.

Ablation Studies on Self-Training Data. In our self-training framework, reasoning, critiquing, and refining data are combined to fine-tune LLM agents. To analyze the impact of each data type, the policy model is fine-tuned with reasoning and refining data, while the critic model is fine-tuned with critiquing data. Results in Figure 6 suggest that removing the reasoning and refining data when training policy models causes the most severe performance degradation. Besides, introducing step-level self-critique to MCTS consistently improves performance, with further gains when the critic model is trained on critiquing data, except when the policy model is fine-tuned solely with reasoning data. This exception arises because the refining ability needs to be improved as the critiquing ability improves. Overall, combining reasoning, self-critiquing, and refining data for self-training achieves optimal performance.

Ablation Study on Critique Models. We compare our critique models with the SOTA open-sourced critique model, Auto-J [51], and one of the best commercial LLM, GPT-4-Turbo. The performance of different critique models is summarized in Table 4.

Without self-training, GPT-4-Turbo demonstrates the best performance across all three metrics, emphasizing the importance of inherent LLM capabilities in generating effective critiques. We observe that AUTO-J, which is fine-tuned from LLaMA-2-13B-chat for diverse alignment scenarios, performs poorly in LLM agent tasks such as WebShop. This highlights the challenge of generating critiques specific to LLM agent scenarios.

After self-training to improve the reasoning ability, the performance of using AUTO-J for critique is improved. However, the performance of using GPT-4-Turbo for critique did not improve, as even GPT-4-Turbo encounters the bottleneck of critique. In comparison, our step-level self-critique model after self-training can significantly enhances overall performance, highlighting the effectiveness of using SLSC, especially after self-training.

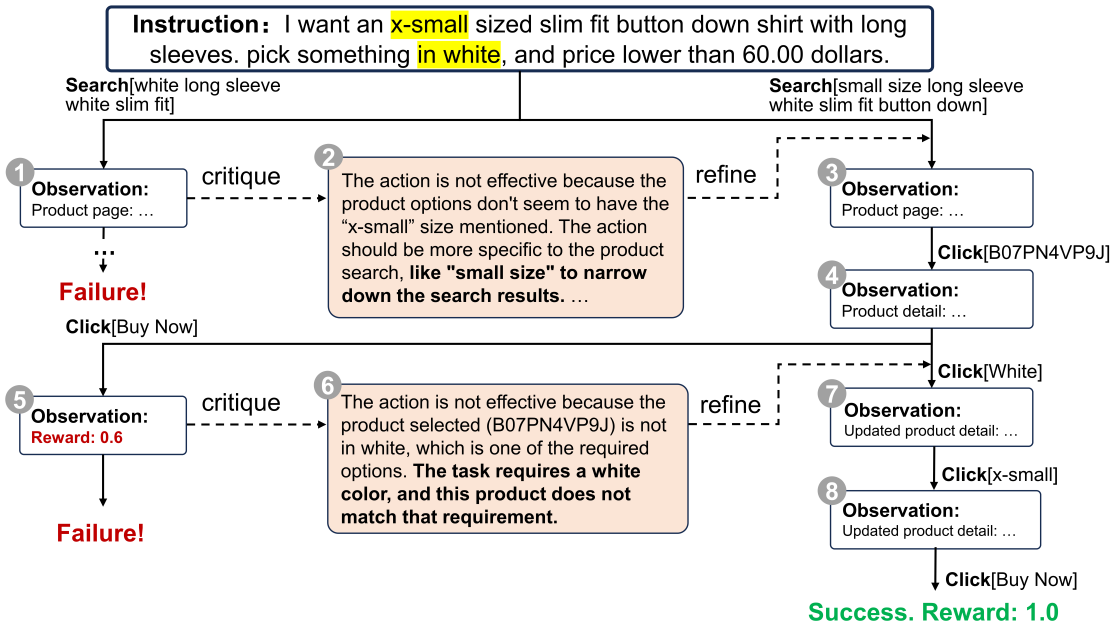

Figure 7: A successful case of LLM agents enhanced by SLSC-MCTS in WebShop.

Table 4: Effect of using different critique models.

Critique	Average Reward ↑	Success Rate ↑	Trajectory Length ↓
Utilizing Llama-3.1-8B-Instruct for reasoning and refining			
AUTO-J	0.71	0.30	4.07
GPT-4-Turbo	**0.75**	**0.38**	**3.97**
SLSC	0.74	0.31	4.15
Utilizing self-trained Llama-3.1-8B-Instruct for reasoning and refining			
AUTO-J	0.73	0.33	4.14
GPT-4-Turbo	0.74	0.35	3.94
Fine-tuned SLSC	**0.82**	**0.46**	**3.90**

Ablation Study on Model Size. We conducted experiments with the Phi-3.5-mini-instruct [52] (3.8B parameters). SLSC-MCTS achieved an average reward of 0.48, surpassing MCTS's 0.42, which validates the effectiveness of our method on the smaller LLM.

5.6 Case Study

Figure 7 illustrates the LLM agent with SLSC-MCTS successfully solving the task in Figure 1. Due to space limitations, only the correct trajectory and two step-level critiques are visualized, with indices in the upper-left corner showing execution order. In comparison to the greedy strategy with trajectory-level critique, step-level critique effectively refines the sub-optimal action "Click[Buy Now]" to "Click[White]' in **step 6**, which is critical for task completion.

Firstly, the user provided an instruction, specifying the product's size (x-small) and color (white). The agent conducted the original search but was criticized in **step 2** for not including "small size" as a keyword to narrow down the search results, leading to search failure. Next, the agent clicked on the product id to navigate to the product details page in **step 4**. Subsequently, the agent initially executed an immediate action - "Click[Buy Now]", which was critiqued for not matching the user's color specification. This critique prompted a refinement of the action to "Click[White]". Finally, after selecting the appropriate size and color, the agent clicked the purchase button in **step 8**, successfully obtaining the product that met both constraints outlined in the user's instruction.

6 Conclusion and Future Work

In this work, we introduce SLSC-MCTS, the first framework to leverage step-level self-critique in multi-step decision-making agent tasks. SLSC-MCTS addresses the key limitation of lacking step-level critiques and incorporates MCTS to balance exploration and exploitation for LLM agents enhanced by step-level self-critiques. Additionally, we propose a self-training framework that efficiently collects the training data from the constructed decision trees to improve the reasoning, critiquing, and refining capabilities of LLM agents. Our evaluation demonstrates that leveraging SLSC-MCTS during test time can empower LLM agents to outperform existing baselines by introducing more test-time computation. Besides, the performance can be further improved with multiple iterations of self-training.

Despite this, there are still limitations and future directions to work on. First, self-critique introduces more computational costs, even if this can be offset by the higher search efficiency of SLSC-MCTS. Second, leveraging step-level self-critique assumes the ability to revert immediately after taking an action. Although the assumption can be satisfied in most information retrieval tasks, it may not be universally applicable in all multi-step decision-making tasks. In future work, we aim to apply SLSC-MCTS to other complex retrieval tasks and develop more general LLM agents with step-level self-critiques.

7 Acknowledgments

This work was partially supported by National Science and Technology Major Project (2023ZD0121101), the Open Fund of National Key Laboratory of Parallel and Distributed Computing (PDL) NO.2024-KJWPDL-02 and National Key Laboratory under grant 231-HF-D04-01.

References

[1] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[2] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600* (2018).

[3] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems* 35 (2022), 20744–20757. http://arxiv.org/abs/2207.01206

[4] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2024. Webarena: A realistic web environment for building autonomous agents. *The Twelfth International Conference on Learning Representations* (2024).

[5] ChengXiang Zhai. 2024. Large language models and future of information retrieval: opportunities and challenges. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 481–490.

[6] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*. https://arxiv.org/abs/2210.03629

[7] Zhiheng Xi, Wenxiang Chen, Boyang Hong, Senjie Jin, Rui Zheng, Wei He, Yiwen Ding, Shichun Liu, Xin Guo, Junzhe Wang, et al. 2024. Training Large Language Models for Reasoning through Reverse Curriculum Reinforcement Learning. *International Conference on Machine Learning* (2024).

[8] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning acting and planning in language models. In *International conference on machine learning*. PMLR.

[9] Yuanzhao Zhai, Tingkai Yang, Kele Xu, Feng Dawei, Cheng Yang, Bo Ding, and Huaimin Wang. 2025. Enhancing decision-making for LLM agents via step-level q-value models. *Proceedings of the AAAI conference on artificial intelligence* (2025).

[10] Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199* (2024).

[11] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems* 36 (2023).

[12] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2023).

[13] OpenAI. 2024. Learning to Reason with Large Language Models. https://openai.com/index/learning-to-reason-with-llms/.

[14] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948* (2025).

[15] Liangchen Luo, Zi Lin, Yinxiao Liu, Lei Shu, Yun Zhu, Jingbo Shang, and Lei Meng. 2023. Critique ability of large language models. *arXiv preprint arXiv:2310.04815* (2023).

[16] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *The Twelfth International Conference on Learning Representations*.

[17] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915* (2023).

[18] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823* (2023).

[19] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing Data and Methods of Effective Agent Tuning for Large Language Models. *arXiv preprint arXiv:2403.12881* (2024).

[20] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314* (2024).

[21] Zelalem Gero, Chandan Singh, Hao Cheng, Tristan Naumann, Michel Galley, Jianfeng Gao, and Hoifung Poon. 2023. Self-verification improves few-shot clinical information extraction. *arXiv preprint arXiv:2306.00024* (2023).

[22] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2022. Large language models are better reasoners with self-verification. *arXiv preprint arXiv:2212.09561* (2022).

[23] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual Reasoning Makes Smaller LLMs Stronger Problem-Solvers. *arXiv preprint arXiv:2408.06195* (2024).

[24] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[25] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems* 36 (2023).

[26] Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 8154–8173.

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

[28] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. AlphaMath Almost Zero: process Supervision without process. *arXiv preprint arXiv:2405.03553* (2024).

[29] Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. 2024. Improve Mathematical Reasoning in Language Models by Automated Process Supervision. *arXiv preprint arXiv:2406.06592* (2024).

[30] Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, et al. 2024. AgentGym: Evolving Large Language Model-based Agents across Diverse Environments. *arXiv preprint arXiv:2406.04151* (2024).

[31] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 15476–15488. https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf

[32] Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking. *arXiv preprint arXiv:2501.04519* (2025).

[33] Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte Carlo Tree Search Boosts Reasoning via Iterative Preference Learning. *arXiv preprint arXiv:2405.00451* (2024).

[34] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825* (2023).

[35] Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. 2023. The Curse of Recursion: Training on Generated Data Makes Models Forget. *ArXiv* abs/2305.17493 (2023). https://api.semanticscholar.org/CorpusID:258987240

[36] Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, et al. 2024. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data. *arXiv preprint arXiv:2404.01413* (2024).

[37] Ting Wu, Xuefeng Li, and Pengfei Liu. 2024. Progress or regress? self-improvement reversal in post-training. *arXiv preprint arXiv:2407.05013* (2024).

[38] Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. 2024. Understanding the Effects of RLHF on LLM Generalisation and Diversity. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=PXD3FAVHJT

[39] Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733* (2024).

[40] Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search. *arXiv preprint arXiv:2406.03816* (2024).

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

[42] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050* (2023).

[43] Peiyi Wang, Lei Li, Zhihong Shao, R.X. Xu, Damai Dai, Yifei Li, Deli Chen, Y.Wu, and Zhifang Sui. 2023. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. *arXiv preprint arXiv:2312.08935* (2023).

[44] Tian Lan, Wenwei Zhang, Chen Xu, Heyan Huang, Dahua Lin, Kai Chen, and Xian-Ling Mao. [n. d.]. CriticEval: Evaluating Large-scale Language Model as Critic. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[45] Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. CriticBench: Benchmarking LLMs for Critique-Correct Reasoning. *arXiv preprint arXiv:2402.14809* (2024).

[46] 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. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[47] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[48] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and Error: Exploration-Based Trajectory Optimization for LLM Agents. *arXiv preprint arXiv:2403.02502* (2024).

[49] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.

[50] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[51] Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, Pengfei Liu, et al. 2024. Generative Judge for Evaluating Alignment. In *The Twelfth International Conference on Learning Representations*.

[52] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219* (2024).