

MASTER: A Multi-Agent System with LLM Specialized MCTS

Bingzheng Gan¹, Yufan Zhao¹, Tianyi Zhang¹, Jing Huang¹, Yusu Li¹
Shu Xian Teo¹, Changwang Zhang¹, Wei Shi¹

¹Huawei Technologies, Co., Ltd.

{gan.bingzheng,zhao.yufan1,zhang.tianyi,huangjing114,liyusu,
teo.shu.xian,zhangchangwang,w.shi}@huawei.com

Abstract

Large Language Models (LLM) are increasingly being explored for problem-solving tasks. However, their strategic planning capability is often viewed with skepticism. Recent studies have incorporated the Monte Carlo Tree Search (MCTS) algorithm to augment the planning capacity of LLM. Despite its potential, MCTS relies on extensive sampling simulations to approximate the true reward distribution, which leads to two primary issues. Firstly, MCTS is effective for tasks like the Game of Go, where simulation results can yield objective rewards (e.g., 1 for a win and 0 for a loss). However, for tasks such as question answering, the result of a simulation is the answer to the question, which cannot yield an objective reward without the ground truth. Secondly, obtaining statistically significant reward estimations typically requires a sample size exceeding 30 simulations, resulting in excessive token usage and time consumption. To address these challenges, we present the **Multi-Agent System with Tactical Execution and Reasoning using LLM Specialized MCTS (MASTER)**, a novel framework that coordinates agent recruitment and communication through LLM specialized MCTS. This system autonomously adjusts the number of agents based on task complexity and ensures focused communication among them. Comprehensive experiments across various tasks demonstrate the effectiveness of the proposed framework. It achieves 76% accuracy on HotpotQA and 80% on WebShop, setting new state-of-the-art performance on these datasets.

1 Introduction

LLM represent a significant milestone in artificial intelligence and are increasingly employed in problem-solving tasks (Xi et al., 2023). However, their application to complex problems is often limited due to concerns about their planning capabilities. LLM generate text based on "next token probability" conditioned on the context (Vaswani

et al., 2017; Huang et al., 2023), but effective reasoning requires rigorous deduction rooted in the first principles of logic, grounded in data and reality (Valmeekam et al., 2023).

To enhance the planning capabilities of LLM, recent studies have incorporated the MCTS algorithm (Hao et al., 2023; Zhou et al., 2024; Wang et al., 2024). MCTS uses simulations to evaluate the long-term consequences of actions, backpropagating aggregated rewards up the tree. This allows for backtracking to previous states based on estimated future potential, striking a balance between exploitation and exploration. However, the use of MCTS in existing works presents two significant challenges:

1. It relies on objective criteria from the external environment to obtain simulation rewards, which are not always available (e.g., in question-answering tasks where the correctness of an answer cannot be determined without ground truth);
2. It requires numerous simulations to obtain statistically significant rewards, which can be unsustainable due to the time and token costs.

To tackle the first challenge, certain approach compares the outcomes of simulations with the ground truth to derive objective rewards (Zhou et al., 2024) which is flawed because revealing the ground truth during problem-solving is inappropriate. In addressing the second challenge, they limit the number of simulations (Hao et al., 2023) or terminate the process once a correct answer is identified (Zhou et al., 2024). Yet, this early termination approach is not feasible if the ground truth remains undisclosed.

These issues highlight a critical step of MCTS: simulation. Due to these limitations, MCTS is constrained to a narrow scope of application and is not

fully compatible with LLM. Consequently, we propose an adaptation of MCTS tailored to LLM scenarios. Instead of performing limited simulations with uncertain rewards, we eliminate the simulation process, relying on the LLM’s self-evaluation capabilities to allocate rewards. Additionally, we propose several methods to enhance the objectivity of rewards: 1). An additional step is introduced for the LLM to provide more context before self-evaluation; 2). The LLM’s confidence is incorporated as a weight for the reward to regulate its influence; 3). The backpropagation mechanism is retained, allowing for rewards updates if initially misallocated. While traditional MCTS focuses resources on simulations for an approximate reflection of reality, our approach distributes resources across multiple steps to jointly ensure the accuracy and objectivity of rewards. This is why we named our project **MASTER**, as it replaces simulation, the core procedure of MCTS, by **mastering** a series of refined designs.

Another contribution of this paper is the introduction of a novel multi-agent system. Current multi-agent systems feature two prominent frameworks: the first allows agents to be independently created and to share ideas freely. While versatile, this open communication can lead to off-topic discussions due to hallucinations (Lin et al., 2024; Hong et al., 2023; Xi et al., 2023; Zhang et al., 2024a), diverting focus from the main task and depleting the token window length for extended conversation history. The second approach involves human-created agents, where communication is predefined. Although more controlled, this approach lacks code reusability (Chu et al., 2023). Additionally, since the procedures are established, it cannot adapt to tasks of varying difficulty. It struggles with unanticipated complex tasks on the one hand and spends unnecessary resources on easy tasks on the other.

Our system, MASTER, addresses these limitations by employing LLM-specialized MCTS to guide the creation and interaction of agents. In this system, child agents respond to and build upon the outputs of the parent agent, making recruitment and communication more manageable and efficient. The system dynamically scales the number of agents based on task complexity, ensuring flexibility. Although the agents share similar profiles, they assume different roles by taking distinct actions. Unlike the nodes in LATS (Zhou et al., 2024) and RAP (Hao et al., 2023), which represent states on a reasoning tree and are closely tied to

specific tasks, our agents are task-agnostic and do not require reconfiguration when the task changes.

In summary, our key contributions are:

1. We propose a novel **Multi-Agent System with Tactical Execution and Reasoning using LLM Specialized MCTS (MASTER)**, a novel multi-agent framework that employs a new agent recruitment process and communication protocol based on the MCTS algorithm. The system autonomously adjusts the number of agents according to task complexity and mitigates distractions and shortage of token window during agent communication.
2. We introduce a modified version of MCTS tailored to LLM. This adaptation is suitable for tasks where the environment does not provide objective feedback, addressing a limitation of the original MCTS. This revised MCTS is implemented within our MASTER framework.
3. We conduct comprehensive experiments across diverse tasks, including Question Answering (HotpotQA), Decision Making (Web-Shop), and Programming (MBPP). It achieves 76% accuracy on HotpotQA and 80% on Web-Shop, setting new SOTA on these datasets.

2 Related Work

Many studies have been done to enhance the planning capabilities of LLM. Among these efforts, two primary planning approaches have emerged for agents: Single-Path Planning and Tree-Based Planning. In the context of multi-agent systems, current frameworks predominantly employ either Predefined Frameworks or Open Frameworks depending on their agent communication patterns. We discuss some of the relevant works in this section.

2.1 Planning Processes

2.1.1 Single-Path Planning

In single-path planning, the LLM follows one trajectory at a time, without branching into multiple possibilities. Early examples include Few-Shot Prompting (Brown et al., 2020), where the LLM is guided by examples of completed tasks, and Chain of Thought approaches (Wei et al., 2022; Kojima et al., 2023; Ning et al., 2024), which require the LLM to reason step-by-step, maintaining a linear trajectory throughout the process. Zhang et al. introduce a structured meta-prompt with placeholders for the LLM to complete (Zhang et al.,

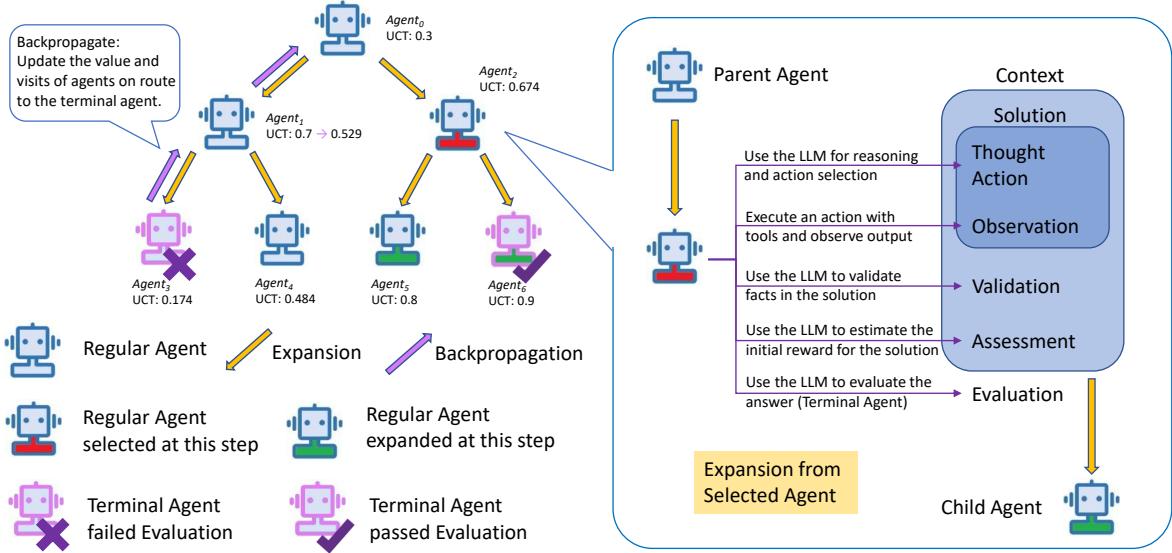


Figure 1: Reasoning Tree of MASTER. Starting from Agent₀, Agent₁ and Agent₂ are created in the first expansion. Then the system first selects Agent₁ for expansion due to its higher UCT. Its child agent Agent₃ is a terminal agent that failed evaluation which triggers a backpropagation and lowers the UCT of Agent₁. Now Agent₂ has the highest UCT and is selected for next expansion. Its child agent, Agent₆ is a terminal agent and passes evaluation. The answer in it is the final answer.

2024c) while Suzgun and Kala provide task-related information to guide the model’s path (Suzgun and Kalai, 2024). Single-path planning also benefits from external feedback to refine solutions. ReAct (Yao et al., 2023b) integrates feedback from the environment, and Reflexion (Shinn et al., 2023) supplements this with verbal reasoning based on the received feedback. Chen et al. use outputs and error messages from a code interpreter to assist the LLM in debugging (Chen et al., 2024c), while Qiu et al. leverage outputs from a symbolic interpreter to enhance the LLM’s inductive reasoning capabilities (Qiu et al., 2024).

2.1.2 Tree-Based Planning

In complex problem-solving, it is often beneficial to explore multiple thought trajectories and backtrack as needed. Tree-based planning organizes these thoughts into a tree structure, on which a search algorithm is applied. For example, BFS/DFS is employed in the Tree of Thoughts (Yao et al., 2023a). RAP (Hao et al., 2023) and LATS (Zhou et al., 2024) utilize MCTS to approximate reality and support the LLM’s reasoning processes. However, how much the simulation process contributes to their success remains uncertain due to those two issues mentioned in Introduction section. To address the first challenge, RAP prompts the LLM with “Is this reasoning step correct?” and

uses the next-word probability of “yes” as a reward, thus leveraging the LLM’s evaluation capabilities without relying on external criteria. For the second challenge, RAP reduces costs by performing only one simulation. From a mathematical perspective, it is questionable that one simulation can accurately approximate the real reward. Some methods stop simulations once a correct answer is found. However, this early termination mechanism is unavailable if the ground truth is not revealed.

2.2 Multi-Agent Systems

2.2.1 Predefined Framework

In predefined frameworks, the recruitment and communication of agents are structured in advance. For instance, ChatDev (Qian et al., 2023) and MetaGPT (Hong et al., 2023), both tailored for software development, rely on predefined workflows. Similarly, AutoAgents (Chen et al., 2024a), a framework designed for automatic agent generation, follows a predefined structure. These frameworks have been criticized for their heavy dependence on upfront planning and their lack of flexibility in responding to changing requirements (Pargaonkar, 2023).

2.2.2 Open Framework

Conversely, open frameworks offer greater flexibility, allowing agents to interact more dynamically. For example, AgentVerse (Chen et al., 2024b) re-

cruits agents with the freedom to communicate openly, while CAMEL (Li et al., 2023) explores a two-agent system. Additionally, AutoGen (Wu et al., 2023) facilitates next-generation LLM applications by enabling multi-agent conversations, allowing for adaptive and context-aware interactions. As noted by Wei et al., effective multi-agent collaboration in these open frameworks requires autonomous systems to regulate communication, ensuring that agents know when and with whom to interact to avoid the exchange of irrelevant information which is challenging for these frameworks (Wei et al., 2023).

3 Methodology

3.1 Preliminaries

Before introducing our framework, we present MCTS to clarify the motivation behind our work. MCTS (Coulom, 2006) is a widely used planning algorithm and famously employed in AlphaGo (Silver et al., 2016). Taking the Game of Go as an example, the algorithm assists in selecting the best possible action in the current state of the board based on their average rewards. These rewards are obtained through numerous simulations. For instance, consider the current state where action a_1 is chosen as the next move. The game then proceeds to completion, with all subsequent actions, whether by our side or the opponent, determined by a policy model rather than a real player. The entire game sequence constitutes one simulation of action a_1 . If we win, the reward is 1; otherwise, it is 0. Specifically, if we simulate 10 games from the current state with action a_1 and win 9 of them, the average reward for a_1 would be 0.9. However, due to the vast action space in the Game of Go, it is impractical to simulate every possible action. The Upper Confidence Bound 1 applied to Trees (UCT) identifies actions with higher potential to win, allocating more simulations to these actions rather than distributing simulations equally among all actions. Once an action is decided based on this process and physically executed, leading to a new game state, the same procedure is then applied to select actions from this new state, and the planning continues until the actual end of the Game of Go.

MCTS typically involves four key procedures: **Selection:** Traverse the current reasoning tree to select the node with the highest UCT for simulation. **Expansion:** Extend the reasoning tree by adding new child nodes from the selected node. **Simula-**

tion: Continue expanding from a child node until the task is completed. **Backpropagation:** After each simulation, update the average reward of corresponding node with the newly obtained simulation reward.

3.2 Framework of MASTER

In our framework, MASTER, the concepts of reasoning tree, selection, expansion, and backpropagation are analogous to those used in MCTS. However, we eliminate the simulation step due to those two issues highlighted in the Introduction section. Instead, our framework incorporates three special mechanisms to derive rewards: providing more context before self-evaluation, incorporating LLM’s confidence in our novel UCT formula, and updating rewards in backpropagation. The details of our framework are demonstrated in Figure 1 and further explained in following paragraphs.

When our framework is presented with a task, it initializes a root agent with the problem. This agent then prompts the LLM to generate text that reflects on the current reasoning trace (**Thought**) and proposes an action to solve the problem (**Action**). The model uses a temperature of 0.6 to encourage more diverse thoughts and actions in the reasoning tree. The agent then executes this action using tools that interact with the external environment, producing feedback (**Observation**). The texts, including Thought, Action, and Observation, form the agent’s **Solution**, as illustrated in Figure 1. Subsequently, the agent prompts the LLM to generate a textual output that verifies key facts in the current Solution (**Validation**). Finally, based on the Solution and Validation, the agent prompts the LLM to evaluate progress toward solving the problem, generating both a score and the LLM’s confidence in this score (**Assessment**). These two values (score and confidence) are then extracted. The model uses a temperature of 0.0 for Validation and Assessment to ensure more stable decisions. All texts (Solution, Validation, and Assessment) form the agent’s **Context** and are stored in its memory (Figure 1). The score serves as the agent’s initial reward.

The previous paragraph describes the generation of a single agent in our system. After the root agent is generated, child agents are created following the same procedure, with the root agent’s Context appended to the prompts of these child agents, as they need to continue solving the problem. As illustrated in Figure 1, two child agents ($Agent_1$ and $Agent_2$) are generated using exactly the same

procedure and prompt to explore diverse reasoning paths from the same state (parent agent). The number of child agents is a hyperparameter, *Number of Branches*, which varies depending on the task. The creation of these child agents is termed **Expansion** from the parent agent.

Further expansion can be carried out from any existing agent using the same procedure. The UCT of each agent is calculated, and the agent with the highest UCT is selected for further expansion. Another hyperparameter, *Maximum of Expansion*, represents the approximate number of steps required to solve the problem, allowing users to set it based on their understanding of the task. If this limit is reached without finding a satisfactory solution, the solution from the terminal agent with the highest reward is submitted as the final answer.

During the expansion of the reasoning tree, agents that generate a final answer rather than an intermediate step in their Solution, are called **Terminal Agents**. For instance, in the HotpotQA task, if an agent’s Action is ‘Finish[]’, it is identified as a terminal agent, as this Action indicates a final answer. Similar indicators exist in the Solution of other tasks. During the **Evaluation** (Figure 1) which applies only to Terminal Agents, the LLM assesses the correctness of the Solution. If the solution is deemed correct, it is submitted as the final answer, concluding the task. If not, **Backpropagation** is triggered, using the reward from this terminal agent to update the rewards of all agents on the path up to the root agent. Pseudo-code can be found in Appendix A.

3.3 Formula of Modified UCT

Recent works RAP and LATS directly apply the original UCT formula. To better suit our design, we propose a modified UCT formula.

The original UCT formula is derived from Hoeffding’s Inequality (Lattimore T, 2020) which can be found in Appendix B. It is typically applied in the following scenario: Given a node representing a state (referred to as node_h), there are multiple subsequent actions to choose from (e.g., a_i, a_j, a_k). To determine the Q value of these actions, multiple simulations are conducted, and UCT is employed to decide which action should be simulated. Instead of simply selecting the node with the highest Q value (pure exploitation, the first term in Eqn 1), UCT balances exploitation and exploration by incorporating an exploration term (the second term in Eqn 1) that favors nodes with fewer simulations.

The node_{*i*}, representing the state resulting from action a_i , is one of the child nodes of node_{*h*}. The UCT formula for node_{*i*} is:

$$UCT = Q_i + \sqrt{\frac{\ln(N_i)}{2n_i}} \quad (1)$$

$$Q_i = \frac{\sum_{n=1}^{n_i} r_n}{n_i} \quad (2)$$

Where n_i is the number of backpropagations applied to the node_{*i*}. r_n is the reward of the n-th backpropagation. Q_i is the estimation of Q value calculated by Eqn 2. It represents the average reward from simulations. N_i is the total number of simulations by the parent node of the current node_{*i*}, which is node_{*h*}. In other words, N_i is the sum of n_i, n_j and n_k in above example although n_j and n_k are not explicitly shown in the UCT formula.

In our system, Q_i , as the estimation of Q value, consists of two components: **Initial Reward** is extracted from Assessment when this agent is generated. **Updating Reward** is the mean of rewards from backpropagation, similar to Q_i in Eqn 2. Inspired by the research on auxiliary information about rewards in the form of control variables (Verma and Hanawal, 2021), we modify the reward estimation in our system as shown in Eqn 3:

$$Q_i = c_0 \cdot r_0 + (1 - c_0) \cdot \frac{\sum_{n=1}^{n_i} r_n}{n_i} \quad (3)$$

Where r_0 is the initial reward given by LLM. c_0 is the confidence of the LLM to this initial reward. r_n and n_i are the same with Eqn 2.

The original MCTS relies heavily on numerous simulations to make this estimation accurate. Consequently, it becomes unreliable when the number of n_i is small. On the other hand, our Q value has an extra component (initial reward), as our Q value is the weighted sum of initial reward and updating reward with the confidence as weight. When LLM has high confidence to the initial reward it assigns, the influence of rewards from backpropagation (updating reward) is reduced due to its lower weight ($1 - c_0$). Conversely, when the LLM has low confidence, updating reward is required to dominate the Q value estimation while the weight of updating reward ($1 - c_0$) is higher. Notably, the number of backpropagation n_i is automatically adapted to each question rather than being manually set by users. For complex tasks, the model requires more attempts to obtain an acceptable answer, with each

failed attempt triggering a backpropagation. For simple question, the model may produce an acceptable result in their first attempt, eliminating the need for backpropagation. This early termination mechanism completes the task while reducing token consumption.

In the formula used by RAP and LATS, an exploration constant λ replaces the fixed $1/\sqrt{2}$ as the weight of the exploration term as following:

$$UCT = Q_i + \lambda \cdot \sqrt{\frac{\ln(N_i)}{n_i}} \quad (4)$$

In our approach, we use $1/(10\sqrt{2}c_0)$ as the exploration weight. The significance of this adjustment is twofold: 1). The exploration term reflects the uncertainty associated with this agent. When the LLM has low confidence in the initial reward, the agent's uncertainty is relatively high, necessitating more exploration. In such cases, a higher exploration weight increases the UCT, guiding the algorithm to select this agent for further exploration; 2). The number of backpropagations in our system is significantly lower than in the original MCTS while the slope of the logarithmic function is relatively steep when the variable is small, so the value of the exploration term tends to play a dominant role. Therefore, this exploration weight should be used to control the influence of this term. Moreover, when the minimum confidence of 0.1 is used, the exploration weight equals $1/\sqrt{2}$, the same as the weight in Eqn 1.

In summary, the revised formula in our system is:

$$UCT = \begin{cases} r_0 & \text{if } n_i = 0 \\ c_0 \cdot r_0 + (1 - c_0) \cdot \frac{\sum_{n=1}^{n_i} r_n}{n_i} \\ + \frac{1}{10\sqrt{2}c_0} \cdot \sqrt{\frac{\ln N_i}{n_i}} & \text{otherwise} \end{cases} \quad (5)$$

When n_i is 0, indicating that no backpropagation has been applied to the node, the UCT of this node is set to be its initial reward r_0 .

3.4 Strategies of Reward Assignment

To conclude, the three special mechanisms we implement to ensure the reliability of rewards in our framework are:

1. Before assigning a reward in the Assessment phase, an additional Validation step is con-

ducted, where the LLM comments on the correctness of the facts in the current solution. These comments are added to the prompt for the Assessment step, guiding the LLM toward a more reliable reward. This design is motivated by the observation that the LLM performs better when it is asked to address one problem at a time. Separately verifying correctness and progress can lead to stable and reliable scoring.

2. In the Assessment phase, the LLM is asked to provide both a score and its confidence in that score, rather than just the score alone. The confidence value plays two roles in our modified UCT formula, which is detailed in the Formula of Modified UCT subsection. If the LLM has low confidence in the score it provides, the influence of this score is reduced and the likelihood to select this agent for further exploration is increased.
3. Backpropagation occurs after each simulation in the original MCTS. Although we have removed simulations, backpropagation is retained in our framework. It is triggered whenever a terminal agent produces a Solution that fails Evaluation. A failed Evaluation indicates that the reasoning steps leading to this terminal agent may be flawed, and their Q value should be reduced accordingly. This mechanism allows for the adjustment of rewards if they are inaccurate at the outset.

4 Experiment Setup

To demonstrate the generalizability of our framework, we conduct experiments across a diverse set of tasks, including question answering (HotpotQA), decision making (WebShop), and coding (MBPP). These datasets are widely recognized benchmarks in their respective domains.

In addition to assessing effectiveness and efficiency, we also performed ablation and parameter studies to investigate the contributions of each mechanism in our framework and the impact of hyperparameters.

4.1 Datasets

HotpotQA: (Yang et al., 2018) tests multi-hop reasoning in LLM, requiring models to parse and reason across multiple paragraphs. We used the Distractor setting, where the task is to answer the

question with a mix of relevant and irrelevant paragraphs context. **WebShop**: (Yao et al., 2022) simulates an e-commerce environment to test decision-making abilities. The task involves navigating a virtual store to find products that best match a given instruction, with success measured by how closely the selected product matches the requirement. **MBPP**: (Austin et al., 2021)) assesses coding abilities. Each task includes a problem description and test cases for validation. In our system, agents generate and test complete code, with child agents iteratively improving on errors identified by their parent agents. A task is considered solved when the generated code passes all test cases.

4.2 Baselines

Since we use GPT-4, a highly capable LLM, as the base model, we take **GPT-4** itself without any agent, as a baseline. It is prompted with the task description and the same examples as our framework to solve tasks in a single call (**Few-shot Chain-of-Thought**). Notably, in this setting, GPT-4 cannot solve HotpotQA and WebShop problems because these tasks require multiple interactions with the environment. For HotpotQA, the model must generate search keywords, receive the retrieved context from the environment, and decide whether to search for additional context or answer the question. For WebShop, the model must purchase a target item on a mock website through multiple search and click actions. Expecting GPT-4 to perform these actions without environment feedback in a single call is impractical. When incorporating external environment feedback to enable multi-turn interactions, the setup becomes identical to the experimental conditions of ReAct. In other words, ReAct’s performance in Table 1 serves as an indicator of the base model’s capacity in these two tasks.

ReAct and **Reflexion** are well-known methods in the planning domain, and our work integrates some of their ideas. **LATS**, like our approach, is a tree-based method and demonstrates strong performance. Therefore, we compare against these baselines across all three datasets. **MetaGPT** and **AgentVerse**, two representative multi-agent systems, serve as our benchmarks in the multi-agent setting. We evaluate them only on MBPP because MetaGPT is specifically designed for programming tasks, while AgentVerse requires execution tools, and only the tool for programming tasks is available.

Additionally, we benchmark our framework

against the current state-of-the-art (SOTA) for each dataset: **Beam Retrieval** for HotpotQA, **AgentKit** for WebShop, and **AgentCoder** for MBPP. However, AgentKit is evaluated on two datasets in its original paper (Crafter and WebShop), while its GitHub repository provides code only for Crafter. Moreover, the available code for Crafter cannot be adapted to WebShop due to insufficient implementation details. Consequently, we rely on the original performance claims from their paper. Nevertheless, given the substantial performance gap between our results and theirs, we believe our SOTA claims remain highly plausible.

4.3 Implementation Details

Given the high cost of GPT-4, we randomly select a sample of 100 questions from each dataset, following the approach used in Reflexion (Shinn et al., 2023) and LATS (Zhou et al., 2024). To ensure fairness, the same random seed is used across all three datasets to select these 100 questions.

To mitigate the effect of LLM randomness on accuracy, we repeat each experiment three times on the same samples and report the mean accuracy in Table 1. Our framework effectively controls LLM randomness through the strategies outlined in the Methodology section, where multiple steps support and verify each other.

5 Results and Analysis

5.1 Effectiveness Analysis

We reproduce all baseline approaches, except for AgentKit due to insufficient information, with GPT-4 as base model on the same 100 questions and record the results. Additionally, we compare these results with those reported in the respective papers. The better result is used as the final value for each baseline in Table 1. This method favors baseline approaches and ensures that our framework demonstrates superior performance across various standards.

MASTER sets new SOTA performance across multiple tasks: 1). 76.0% Exact Match accuracy on HotpotQA, surpassing Beam Retrieval’s 73.3% (Zhang et al., 2024b); 2). 80.0% accuracy on WebShop, exceeding AgentKit’s 70.2% (Wu et al., 2024). On MBPP, it closely matches AgentCoder’s 91.8% pass@1 accuracy with 91.0% (Huang et al., 2024), showing competitive performance in programming task.

	HotpotQA	WebShop	MBPP
GPT-4 (CoT)	-	-	0.683
ReAct	0.420	0.320	0.710
Reflexion	0.510	0.350	0.771
LATS	0.710	0.380	0.811
MetaGPT	-	-	0.877
AgentVerse	-	-	0.890
Beam Retrieval	0.733	-	-
AgentKit	-	0.702	-
AgentCoder	-	-	0.918
Ours	0.760	0.800	0.910

Table 1: Effectiveness comparison with accuracy.

5.2 Efficiency Analysis

As a tree-based method, token consumption is a concern due to the diverse reasoning trajectories. However, by removing the simulation step and introducing an early termination mechanism, our framework achieves higher efficiency compared to other tree-based methods using MCTS. We benchmark our efficiency against LATS because: 1). LATS is a typical tree-based approach with superior performance compared to similar frameworks; 2). LATS reports the lowest token consumption when compared with ToT (Yao et al., 2023a) and RAP (Hao et al., 2023) in their paper.

We measure token consumption for LATS ($n = 5$, $k = 50$) and MASTER (*number of branches = 2, maximum of expansion = 3*) on the same 100 HotpotQA questions, using the same hyperparameter settings as those in the effectiveness analysis experiments. The average cost per question was 185,392 tokens for LATS and 10,937 tokens for MASTER. Our approach uses only about 6% of the tokens compared to LATS while delivering better performance (Table 1).

LATS (Zhou et al., 2024) reports an average cost of 173,290 tokens per question, lower than our reproduced results. This discrepancy may be due to the fact that their tests were conducted on correctly answered questions, while our tests included some incorrectly answered questions, which tend to consume more tokens as the algorithm continues until reaching the maximum trial limit.

5.3 Ablation Study

5.3.1 UCT Modification

In MCTS, the UCT formula balances exploration and exploitation. One of our main contributions is adapting this formula to better accommodate LLM. We evaluate the impact of removing components

of our modified UCT formula, considering the following cases (Table 2):

1. Our full modified formula, incorporating the weighted sum of initial and updating rewards, and exploration weight influenced by the LLM’s confidence (Eqn 5).
2. A variant with the weighted sum of rewards and a fixed exploration weight (Eqn 6). This differs from the full formula by not incorporating the LLM’s confidence in the exploration term.

$$UCT = c_0 \cdot r_0 + (1 - c_0) \cdot \frac{\sum_{n=1}^{n_i} r_n}{n_i} + \sqrt{\frac{\ln(N_i)}{n_i}} \quad (6)$$

3. A variant using only the weighted sum of rewards (Eqn 7), removing the entire exploration term.

$$UCT = c_0 \cdot r_0 + (1 - c_0) \cdot \frac{\sum_{n=1}^{n_i} r_n}{n_i} \quad (7)$$

4. A variant using only the initial reward for exploitation (Eqn 8), excluding exploration term and updating reward from backpropagation.

$$UCT = r_0 \quad (8)$$

When using UCT with a fixed exploration weight (case 2), performance declines on two out of three datasets, even worse than the variant without the entire exploration term. As discussed in the Methodology section, the exploration term plays a dominant role and should be modulated by the exploration weight. This result supports that hypothesis, as the system sometimes over-explores without progressing toward task completion when using this formula. Omitting a dynamic exploration weight based on confidence is harmful to the system.

Using the weighted sum of initial and updating rewards (case 3) performs better than using the initial reward alone (case 4), likely because the updating reward incorporates additional information from deeper in the reasoning tree.

The MBPP results indicate that different UCT variants have no significant effect on this dataset. This outcome likely arises because the Observation, represented by the test case results, is objective. The Observation is appended to the prompt

	HotpotQA	WebShop	MBPP
Full modified UCT	0.760	0.800	0.910
Fixed Exploration Weight	0.700	0.677	0.910
w/o Exploration Term	0.737	0.750	0.910
Only Initial Reward	0.723	0.703	0.910

Table 2: Ablation study with removing different components of our modified UCT formula.

	HotpotQA	WebShop	MBPP
Full setting	0.760	0.800	0.910
w/o Validation	0.623	0.563	0.863
w/o Assessment	0.233	0.157	0.743

Table 3: Ablation study with removing validation or assessment of our agent.

of the Assessment, allowing the LLM to assign rewards with high confidence. When c_0 is 1, all UCT variants reduce to case 4 or very close to it. Therefore, all settings yield identical or nearly identical outcomes.

5.3.2 Agent Design

The validation step before assessment is another key feature of our framework. We conduct additional ablation studies, removing the validation and assessment steps individually. Since the initial reward is derived from the assessment and the algorithm cannot function without it, we assign random initial rewards when the assessment step is removed (Table 3).

Performance drops significantly when the validation or assessment step is removed, even for the MBPP dataset, as validation greatly impacts reward allocation, the core component of the MCTS. Additionally, with random rewards assigned when the assessment step is removed, performance essentially relies on the LLM alone, or worse.

5.4 Parameter Study

We conduct parameter studies to determine optimal values for two key hyperparameters: *Number of Branches* (Table 4) and *Maximum of Expansion* (Table 5), across all three datasets.

The *Number of Branches* has little effect on performance in WebShop and MBPP. However, on HotpotQA, performance drops by nearly 3% when reducing it from 2 to 1. This reduction may hinder the system, as multiple reasoning trajectories help prevent getting stuck in incorrect states. Although WebShop faces similar challenges, the agent can use the dataset in-built action ‘prev’ to mitigate, though not completely avoid, this problem. To

Number of Branches	HotpotQA	WebShop	MBPP
1	0.733	0.797	0.903
2 (All datasets used)	0.760	0.800	0.910
3	0.763	0.797	0.910

Table 4: Parameter study on Number of Branches.

Maximum of Expansion	HotpotQA	WebShop	MBPP
1	0.000	0.000	0.747
3 (HotpotQA & MBPP)	0.760	0.013	0.910
8 (WebShop)	0.770	0.800	0.910

Table 5: Parameter study on Maximum of Expansion.

balance performance and cost, we use 2 for all datasets.

We select 1, 3, 8 for the *Maximum of Expansion* here because in our other experiments, 3 is used in HotpotQA and MBPP while 8 is used for WebShop. Normally, questions in HotpotQA and WebShop require multiple steps to solve so their performance drops dramatically when the *Maximum of Expansion* is lower than the steps needed to solve the problem, as they are forced to stop before getting an answer.

6 Conclusion

This paper introduces MASTER, a novel multi-agent system framework that leverages a specialized MCTS to enhance the planning capabilities of LLM. Our LLM-optimized MCTS broadens the applicability of MCTS to a wider range of tasks with reduced costs. Besides, we employ this algorithm to guide agent recruitment and communication protocols, thereby introducing an innovative form of multi-agent system. Extensive experiments across various datasets demonstrate MASTER’s effectiveness and efficiency over existing frameworks.

7 Limitations

There are some limitations in our framework. Firstly, it relies heavily on the LLM’s ability to provide accurate scores and confidence assessments of the current reasoning state. While GPT-4 performs this task effectively, smaller open-source models may encounter challenges at this step. Additionally, users must configure certain hyperparameters for the system, including the *Maximum of Expansion* and the *Number of Branches*. The optimal values for these parameters may vary depending on the specific task.

References

- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. [Program synthesis with large language models](#). *Preprint*, arXiv:2108.07732.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. [Language models are few-shot learners](#). *Preprint*, arXiv:2005.14165.
- Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2024a. [Autoagents: A framework for automatic agent generation](#). *Preprint*, arXiv:2309.17288.
- Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. [Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors](#). In *The Twelfth International Conference on Learning Representations*.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024c. [Teaching large language models to self-debug](#). In *The Twelfth International Conference on Learning Representations*.
- Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. 2023. [A survey of chain of thought reasoning: Advances, frontiers and future](#). *Preprint*, arXiv:2309.15402.
- Rémi Coulom. 2006. [Efficient selectivity and backup operators in monte-carlo tree search](#). In *5th International Conference on Computer and Games*.
- Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. [Reasoning with language model is planning with world model](#). In *The 2023 Conference on Empirical Methods in Natural Language Processing*.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiaowu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. [Metagpt: Meta programming for a multi-agent collaborative framework](#). *Preprint*, arXiv:2308.00352.
- Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. 2024. [Agentcoder: Multi-agent-based code generation with iterative testing and optimisation](#). *Preprint*, arXiv:2312.13010.
- Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. [A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions](#). *Preprint*, arXiv:2311.05232.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. [Large language models are zero-shot reasoners](#). *Preprint*, arXiv:2205.11916.
- Szepesvári C Lattimore T. 2020. *Bandit Algorithms*, page 80. Cambridge University Press.
- Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. 2023. [CAMEL: Communicative agents for "mind" exploration of large language model society](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Jianghao Lin, Rong Shan, Chenxu Zhu, Kounianhua Du, Bo Chen, Shigang Quan, Ruiming Tang, Yong Yu, and Weinan Zhang. 2024. [ReLLa: Retrieval-enhanced large language models for lifelong sequential behavior comprehension in recommendation](#). In *The Web Conference 2024*.
- Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2024. [Skeleton-of-thought: Prompting llms for efficient parallel generation](#). *Preprint*, arXiv:2307.15337.
- Shravan Pargaonkar. 2023. [A comprehensive research analysis of software development life cycle \(sdlc\) agile & waterfall model advantages, disadvantages, and application suitability in software quality engineering](#). *International Journal of Scientific and Research Publications*, 13:120–124.
- Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. [Communicative agents for software development](#). *Preprint*, arXiv:2307.07924.
- Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. 2024. [Phenomenal yet puzzling: Testing inductive reasoning capabilities of language models with hypothesis refinement](#). In *The Twelfth International Conference on Learning Representations*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. [Reflexion: language agents with verbal reinforcement learning](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.

- David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.
- Mirac Suzgun and Adam Tauman Kalai. 2024. **Meta-prompting: Enhancing language models with task-agnostic scaffolding**. *Preprint*, arXiv:2401.12954.
- Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. **Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change**. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. **Attention is all you need**. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Arun Verma and Manjesh Kumar Hanawal. 2021. **Stochastic multi-armed bandits with control variates**. In *Advances in Neural Information Processing Systems*.
- Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024. **Promptagent: Strategic planning with language models enables expert-level prompt optimization**. In *The Twelfth International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. **Chain of thought prompting elicits reasoning in large language models**. In *Advances in Neural Information Processing Systems*.
- Jimmy Wei, Kurt Shuster, Arthur Szlam, Jason Weston, Jack Urbanek, and Mojtaba Komeili. 2023. **Multi-party chat: Conversational agents in group settings with humans and models**. *Preprint*, arXiv:2304.13835.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. **Autogen: Enabling next-gen llm applications via multi-agent conversation**. *Preprint*, arXiv:2308.08155.
- Yue Wu, Yewen Fan, So Yeon Min, Shrimai Prabhumoye, Stephen McAleer, Yonatan Bisk, Ruslan Salakhutdinov, Yuanzhi Li, and Tom Mitchell. 2024. **Agentkit: Flow engineering with graphs, not coding**. *Preprint*, arXiv:2404.11483.
- Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. **The rise and potential of large language model based agents: A survey**. *Preprint*, arXiv:2309.07864.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. **HotpotQA: A dataset for diverse, explainable multi-hop question answering**. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. **Webshop: Towards scalable real-world web interaction with grounded language agents**. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023a. **Tree of thoughts: Deliberate problem solving with large language models**. In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. **React: Synergizing reasoning and acting in language models**. In *The Eleventh International Conference on Learning Representations*.
- Bin Zhang, Hangyu Mao, Jingqing Ruan, Ying Wen, Yang Li, Shao Zhang, Zhiwei Xu, Dapeng Li, Ziyue Li, Rui Zhao, Lijuan Li, and Guoliang Fan. 2024a. **Controlling large language model-based agents for large-scale decision-making: An actor-critic approach**. *Preprint*, arXiv:2311.13884.
- Jiahao Zhang, Haiyang Zhang, Dongmei Zhang, Yong Liu, and Shen Huang. 2024b. **End-to-end beam retrieval for multi-hop question answering**. *Preprint*, arXiv:2308.08973.
- Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. 2024c. **Meta prompting for ai systems**. *Preprint*, arXiv:2311.11482.
- 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**.

A Pseudo Code

The pseudo code of our framework is demonstrated in this section.

After initializing a root agent with the question, multiple LLM calls are made in 'root.action()' to get the solution, validation and assessment of this root agent. The score and confidence are extracted from assessment. If this is a terminal agent, LLM is called again to decide whether its solution can pass the evaluation. All the information above including solution, validation, assessment, score, confidence and pass (bool) is saved in the memory of this agent and is passed to its child agent for information. In this pseudo code, only pass, solution and score are explicitly expressed for simplicity but all of them are obtained in our full code.

Then the node with the highest UCT is selected by the function 'select_with_uct(root)'. This function will go through all the agents under root agent, which is the entire reasoning tree to select the one with the highest UCT for expansion. This procedure is called Selection which will be done i (Maximum of Expansion) times.

When an agent is selected, the algorithm will generate a few child agent. Every generation is done by 'agent.expand()' which will initialize a new agent and generate its own solution, validation, assessment, etc. like above. The only difference is that the root agent only has the question in its prompt but these child agent has all the context of the agents on the path from the root to itself in its prompt (its parent, parent of parent, etc.). This creation of child agent will be repeated j (Number of Branches) times which is called Expansion.

Whenever a terminal agent is generated, the evaluation as mentioned above is conducted. If it fails the evaluation (the variable pass is False), a procedure of Backpropagation is triggered. The reward of the agents on the path to this terminal agent will be updated with the reward of it.

B Deduction of UCT Formula

In the methodology section, we proposed a modification to the UCT formula. Here, we provide the derivation of the original UCT formula.

According to Hoeffding's Inequality, for a random variable r such that $r \in [a, b]$, if r_1, r_2, \dots, r_n are independent and $r_i \in [a, b]$ almost surely with $a_i < b_i$ for all i , then:

$$\begin{aligned} P\left(\frac{1}{n_i} \sum_{n=1}^{n_i} (r_n - E[r_n]) \geq \varepsilon\right) \\ \leq \exp\left(\frac{-2n_i^2 \varepsilon^2}{\sum_{n=1}^{n_i} (b_n - a_n)^2}\right) \end{aligned} \quad (9)$$

Algorithm 1 MASTER algorithm

Input: Question

Parameter: Expansion Maximum, Branches Number

Output: Solution

```

1: Initialize root with Question and Generate its
   Context
2:  $root \leftarrow \text{Agent}(question)$ 
3:  $pass, solution, sc \leftarrow root.\text{action}()$ 
4: for  $i \leftarrow 1, \dots$ , Expansion Maximum do
5:   SELECTION
6:    $agent \leftarrow \text{select\_with\_uct}(root)$ 
7:   for  $j \leftarrow 1, \dots$ , Branches Number do
8:     EXPANSION
9:      $pass, child\_solution, child\_score \leftarrow agent.\text{expand}()$ 
10:    if  $pass$  is True then
11:      return solution
12:    else if  $pass$  is False and  $child$  is Terminal
        then
13:      BACKPROPAGATION
14:      while  $agent$  is not None do
15:         $agent.value \leftarrow update(child\_score)$ 
16:         $agent.visits \leftarrow agent.visits + 1$ 
17:         $agent \leftarrow agent.parent$ 
18:      end while
19:    end if
20:  end for
21: end for

```

The following term could be rewritten as:

$$\begin{aligned} \frac{1}{n_i} \sum_{n=1}^{n_i} (r_n - E[r_n]) &= \frac{1}{n_i} \sum_{n=1}^{n_i} r_n - \frac{1}{n_i} \sum_{n=1}^{n_i} E[r_n] \\ &= \frac{1}{n_i} \sum_{n=1}^{n_i} r_n - E[r_n] \end{aligned} \quad (10)$$

In Eqn 10, the second term is the mean of the random variable r_n which is noted as q_i . The first term is the sample mean or the estimation of q_i which is noted as Q_i . If r_n as reward is always between 0 and 1, $\sum_{i=1}^{n_i} (b_n - a_n)^2 = \sum_{n=1}^{n_i} 1 = n_i$. Then Eqn 9 is:

$$P(Q_i - q_i \geq \varepsilon) \leq \exp(-2n_i^2 \varepsilon^2) \quad (11)$$

Consider the symmetry:

$$P(Q_i - q_i \leq -\varepsilon) \leq \exp(-2n_i^2 \varepsilon^2) \quad (12)$$

Combine Eqn 11 and Eqn 12:

$$P(-\varepsilon \leq Q_i - q_i \leq \varepsilon) \geq 1 - 2 \cdot \exp(-2n_i^2\varepsilon^2) \quad (13)$$

Let $N_i = \exp(2n_i\varepsilon^2)$, then

$$\varepsilon = \sqrt{\frac{\ln(N_i)}{2n_i}} \quad (14)$$

When we consider the upper bound of the reward:

$$Q_i - q_i \geq -\varepsilon \quad (15)$$

$$q_i \leq Q_i + \sqrt{\frac{\ln(N_i)}{2n_i}} \quad (16)$$

Eqn 16 is the normal UCT formula mentioned in our paper.

C Case Study

C.1 HotpotQA

Question: Dawn French's Girls Who Do Comedy included which American comedienne and television host?

Agent₀

Thought: The question asks for the name of an American comedienne and television host who was included in Dawn French's Girls Who Do Comedy. To answer this question, I need to find out which American female comedians were featured on that series.

Action: Search[Dawn French's Girls Who Do Comedy]

Observation: Dawn French's Girls Who Do Comedy is an interview series shown on BBC Four... Whoopi Goldberg, Catherine Tate, Kathy Burke, Julie Walters, Victoria Wood and Joan Rivers...

Validation: The latest 'Thought' is looking for information about Dawn French's Girls Who Do Comedy which aligns with the 'Question' that mentioned the interview series. The latest 'Action' searched for Dawn French's Girls Who Do Comedy as instructed by the latest 'Thought', so they are aligned.

Assessment: The trajectory has successfully found out comediennes included in Dawn French's Girls Who Do Comedy. However, the trajectory has not identify the American comedienne and television host. Thus the correctness score is 3. The confidence in this score is 9.

Agent₁

Thought: Whoopi Goldberg, Catherine Tate, Kathy Burke, Julie Walters, Victoria Wood and Joan Rivers were included in Dawn French's Girls Who Do Comedy. To answer the question, I need to search everyone of them for more information.

Action: Search[Whoopi Goldberg]

Observation: Caryn Elaine Johnson (born November 13, 1955), known professionally as Whoopi Goldberg, is an American actor, comedian, author, and television personality...

Validation: The latest 'Thought' is looking for information about Whoopi Goldberg, Catherine Tate, Kathy Burke, Julie Walters, Victoria Wood and Joan Rivers which aligns with the 'Observation' that they are included in Dawn French's Girls Who Do Comedy. The latest 'Action' searched for Whoopi Goldberg as instructed by the latest 'Thought', so they are aligned.

Assessment: This trajectory has enough information to answer the question since Whoopi Goldberg is an American comedienne who is included in Dawn French's Girls Who Do Comedy. Thus the correctness score is 8. The confidence in this score is 7.

Agent₂

Thought: The question asks for the name of an American comedienne in Dawn French's Girls Who Do Comedy. To answer the question, I need to search American comedienne.

Action: Search[American comedienne]

Observation: Katherine Beth Nolan (born January 28, 1987) is an American sports personality, comedienne, and television host on Fox Sports 1... Joan Alexandra Molinsky (June 8, 1933 – September 4, 2014), widely known as Joan Rivers, was an American comedian, actress, writer, producer, and television host...

Validation: The latest 'Thought' is looking for American comedienne which aligns with the 'Observation' that comediennes included in Dawn French's Girls Who Do Comedy are listed. The latest 'Action' searched for American comedienne as instructed by the latest 'Thought', so they are aligned.

Assessment: This trajectory searched American comedienne but did identify the one included in Dawn French's Girls Who Do Comedy. Thus the correctness score is 6. The confidence in this score is 9.

Agent₃

Thought: Whoopi Goldberg is an American comedienne who is included in Dawn French's

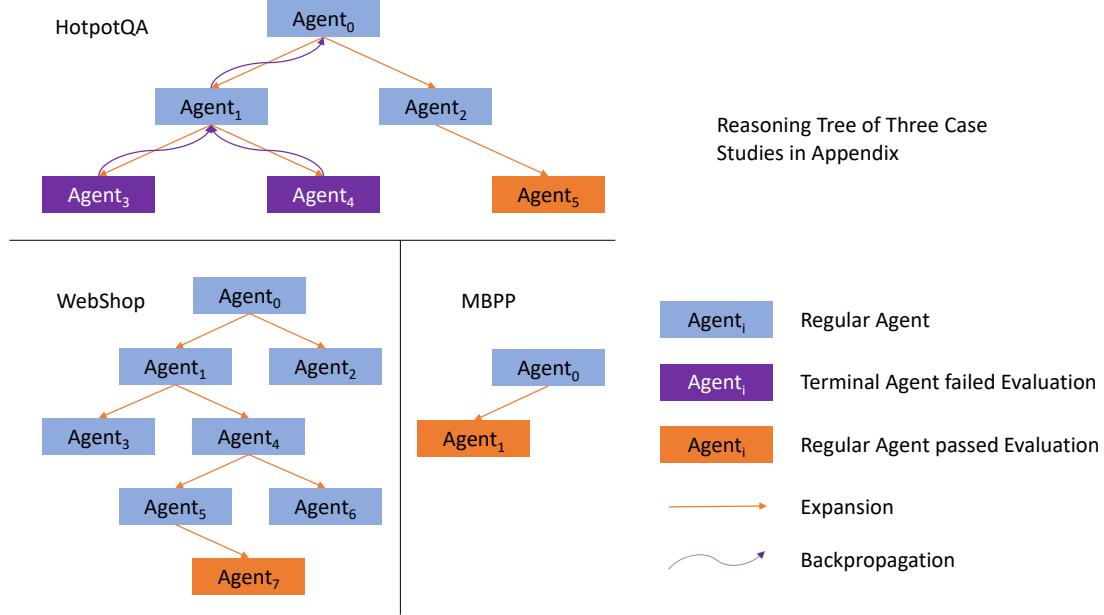


Figure 2: Reasoning Tree of case studies on three datasets.

Girls Who Do Comedy and all the information needed is found. I should answer the question now.

Action: Finish[Whoopi Goldberg]

Validation: The latest 'Thought' indicated to answer the question with Whoopi Goldberg which does not align with the 'Observation' because it is not mentioned that Whoopi Goldberg is a television host. The latest 'Action' finished the question with answer 'Whoopi Goldberg' as instructed by the latest 'Thought', so they are aligned.

Assessment: The answer is incorrect according to the validation because it is not mentioned that Whoopi Goldberg is a television host. Thus the correctness score is 2. The confidence in this score is 7.

Evaluation: False

Agent₄

Thought: Whoopi Goldberg is an American comedienne included in Dawn French's Girls Who Do Comedy and all the information needed is found. I should answer the question now.

Action: Finish[Whoopi Goldberg]

Validation: The latest 'Thought' answers the question with Whoopi Goldberg which does not align with the 'Observation' because it is not mentioned that Whoopi Goldberg is a television host. The latest 'Action' finished the question with answer 'Whoopi Goldberg' as instructed by the latest 'Thought', so they are aligned.

Assessment: The answer is incorrect according

to the validation because Whoopi Goldberg is not a television host. Thus the correctness score is 2. The confidence in this score is 7.

Evaluation: False

Agent₅

Thought: Joan Rivers was an American comedian and television host who is included in Dawn French's Girls Who Do Comedy and all the information needed is found. I should answer the question now.

Action: Finish[Joan Rivers]

Validation: The latest 'Thought' indicated to answer the question with Joan Rivers which aligns with the 'Observation' because Joan Rivers is an American comedian and a television host who is included in Dawn French's Girls Who Do Comedy. The latest 'Action' Finished the question with answer 'Joan Rivers' as instructed by the latest 'Thought', so they are aligned.

Assessment: The answer is correct according to the validation because all the information is aligned. Thus the correctness score is 10. The confidence in this score is 9.

Evaluation: True

C.2 WebShop

Question: Instruction: Find me dual band streaming media players with quad core, and price lower than 350.00 dollars

Agent₀

Thought: The instruction requires finding a dual-band streaming media player with a quad-core processor and a price lower than \$350.00. Since there is a search bar available, the best course of action is to use it to search for products that match the criteria.

Action: search[dual band streaming media player with quad core]

Observation: Product index is B09LSKQF8C. Product description is Superbox S3 Pro Dual Band Wi-Fi 2.4Ghz 5Ghz Supports 6K Video. Product price is \$329.0... Available actions: has search bar: False, clickables: [b09lskqf8c, b09nmzs7hk, b00r249c5g, b07srs3mxp, b00ofqr8d2, b0871vqt3q, b085vfmxmz, b01lzwdnx6, b08qsnm69h, b08m5byqn9]

Validation: The common attributes include dual-band, streaming media player, quad-core, and price lower than \$350.00, while there are no optional attributes specified in the instruction. The group of products to be discarded (Group 1) includes B09NMZS7HK, B00R249C5G, B07SRS3MXP, B00OFQR8D2, B0871VQT3Q, B085VFMXMZ, B01LZWDXN6, B08QSNM69H, and B08M5BYQN9 because they either do not match the common attributes of being a dual-band streaming media player with a quad-core processor or exceed the specified price limit. The group of products to be considered (Group 2) includes B09LSKQF8C because it matches the common attributes of being a dual-band streaming media player with a quad-core processor and is priced below \$350.00.

Assessment: The product B09LSKQF8C meets all the specified requirements: it is a dual-band streaming media player with a quad-core processor and is priced below \$350.00. Thus, this product is a suitable choice and the task is close to completion, pending a final review of the product details and the purchase action. Thus the correctness score is 6. The confidence in this score is 9.

Agent₁

Thought: The product B09LSKQF8C seems to fit the requirements as it is a dual-band streaming media player and is priced below \$350.00. However, the product description does not mention if it has a quad-core processor. To ensure that it meets all the criteria, I need to click on the product index B09LSKQF8C to view more detailed information before making a purchase decision.

Action: click[b09lskqf8c]

Observation: Product description is Superbox

S3 Pro Dual Band Wi-Fi 2.4Ghz 5Ghz Supports 6K Video. Price: \$329.0. Click corresponding options if needed. Click 'features' if you need more information. Click 'buy now' if you decide to purchase this product. Available actions: has search bar: False, clickables: [back to search, prev, description, features, reviews, buy now]

Validation: The common attributes include dual-band streaming media players, quad-core, and price lower than 350.00 dollars, while there are no optional attributes specified in the instruction. The product Superbox S3 Pro fits the common attributes as it is a dual-band streaming media player and is priced below \$350.00. However, the product description does not mention if it has a quad-core processor, which is a required common attribute.

Assessment: The latest 'Thought' indicates that the quad-core processor attribute is crucial and has not yet been confirmed for the Superbox S3 Pro. The 'Action' taken is to click on 'features' to verify this attribute, and the 'Observation' is pending, awaiting confirmation on whether the quad-core processor is present. If the feature is confirmed, the task is close to completion; otherwise, further search is required. Thus the correctness score is 7. The confidence in this score is 8.

Agent₂

Agent₂ is similar to **Agent₁**, omitted due to space limitation.

Agent₃

Thought: The Superbox S3 Pro mentions that it supports dual-band Wi-Fi and 6K video, and the price is listed as \$329.00, which meets the price requirement. However, it is not confirmed whether the device has a quad-core processor. The details could be found in 'description' section. It is necessary to check the 'description' to find out the specified requirements before proceeding with the purchase.

Action: click[description]

Observation: The newest SuperBox S3 Pro is an advanced voice control IPTV box, with a brand new Bluetooth remote and built-in artificial intelligence system, you can easily speak to control your TV box. The box has 2GB of RAM and 32GB of storage, utilizes a quad-core ARM Cortex-A53 processor, supports Android 9.0 OS. The new model has a new flat rectangular shape of design with an additional led display and new interface, the SuperBox S3 Pro has also added more features. It adopts 2T2R WiFi technology makes our streaming device 60% faster and more stable. In conclusion,

the SuperBox S3 Pro is really the game-changer in streamer, it is by far the most powerful and most user-friendly TV Box on the market. Available actions: has search bar: False, clickables: [back to search, prev]

Validation: The common attributes include dual-band streaming media players, quad-core, and price lower than 350.00 dollars, while there are no optional attributes specified in the instruction. It also has a quad-core processor, which is a required common attribute.

Assessment: The latest 'Thought' indicates that the quad-core processor attribute is crucial and is not confirmed for the Superbox S3 Pro. The 'Action' taken is to click on 'description' to verify this attribute, as quad-core processor is present, the task is close to completion. Thus the correctness score is 7. The confidence in this score is 7.

Agent₄

Thought: The product description provided for the Superbox S3 Pro mentions that it supports dual-band Wi-Fi and 6K video, and the price is listed as \$329.00, which meets the price requirement. However, the description does not confirm whether the device has a quad-core processor. The 'features' section is likely to contain more detailed specifications about the product, including the processor details. It is necessary to check the 'features' to ensure that the product meets all the specified requirements before proceeding with the purchase.

Action: click[features]

Observation: Prev Built-in Voice Control System Android 9.0 OS Quad-core ARM Cortex-A53 Processor 2T2R(2 Transmitter, 2 receivers) antenna and 2.4G/5G Dual Band WiFi 2GB DDR3 Memory 32GB eMMC Internal Storage Support 4K@60fps. Available actions: has search bar: False, clickables: [back to search, prev]

Validation: The common attributes include dual-band streaming media players, quad-core processor, and price lower than \$350.00 dollars. There are no optional attributes specified in the instruction. The current state indicates that the product B09LSKQF8C, identified as the Superbox S3 Pro, has been clicked on and its features have been examined. The features confirm that the product has a quad-core ARM Cortex-A53 processor and supports 2.4G/5G dual-band Wi-Fi. The price of \$329.00 is also within the specified budget. Since the product meets all the common attributes specified in the instruction, it is the correct product to select. The trajectory does not require reselection

of any attributes, as there are no optional attributes to consider, and the product matches all the common attributes. Therefore, the next step should be to proceed with the purchase.

Assessment: The product B09LSKQF8C, Superbox S3 Pro, has been confirmed to have a quad-core ARM Cortex-A53 processor, supports dual-band Wi-Fi, and is priced at \$329.00, which meets all the specified requirements. The task is close to being solved as the only remaining step is to initiate the purchase by clicking 'buy now'. Thus the correctness score is 9. The confidence in this score is 8.

Agent₅

Thought: The features of the Superbox S3 Pro confirm that it has a quad-core ARM Cortex-A53 processor, supports 2.4G/5G dual-band WiFi, and is within the specified price range. Since it meets all the requirements outlined in the instruction, the next step is to go back to the previous page where the 'buy now' button is available and complete the purchase.

Action: click[prev]

Observation: Product description is Superbox S3 Pro Dual Band Wi-Fi 2.4Ghz 5Ghz Supports 6K Video. Price: \$329.0. Click corresponding options if needed. Click 'features' if you need more information. Click 'buy now' if you decide to purchase this product. Available actions: has search bar: False, clickables: [back to search, prev, description, features, reviews, buy now]

Validation: The common attributes include dual-band streaming media players with a quad-core processor and a price lower than \$350.00. There are no optional attributes specified in the instruction. The current state shows that the Superbox S3 Pro meets all the common attribute requirements: it supports dual-band Wi-Fi, has a quad-core ARM Cortex-A53 processor, and is priced at \$329.00, which is below the \$350.00 threshold. Since there are no optional attributes to consider and the product matches all the common attributes, the next logical step is to proceed with the purchase. The topic of the response is to remind the agent to complete the purchase. Since the product meets all the necessary criteria and there are no optional attributes to adjust, the agent should click 'buy now' to finalize the purchase.

Assessment: The product Superbox S3 Pro has been confirmed to meet all the specified requirements, including the dual-band capability, quad-core ARM Cortex-A53 processor, and the price

point of \$329.00. The next step is to click 'buy now' to proceed with the purchase, as there are no optional attributes to adjust and all common attributes are satisfied. Thus the correctness score is 9. The confidence in this score is 9.

Agent₆ is similar to **Agent₅**, omitted due to space limitation.

Agent₇

Thought: The Superbox S3 Pro has been confirmed to meet all the requirements of the instruction, which include being a dual-band streaming media player with a quad-core processor and having a price lower than \$350.00. The 'buy now' button is available to complete the purchase.

Action: click[buy now]

Observation: Product description is Superbox S3 Pro Dual Band Wi-Fi 2.4Ghz 5Ghz Supports 6K Video. Price: \$329.0. Click corresponding options if needed. Click 'features' if you need more information. Click 'buy now' if you decide to purchase this product. Available actions: has search bar: False, clickables: [back to search, prev, description, features, reviews, buy now]

Validation: The common attributes include dual-band streaming media players with a quad-core processor and a price lower than \$350.00. There are no optional attributes specified in the instruction. The current state shows that the Superbox S3 Pro meets all the common attribute requirements and there are no optional attributes to consider.

Assessment: The product Superbox S3 Pro has been confirmed to meet all the specified requirements, including the dual-band capability, quad-core ARM Cortex-A53 processor, and the price point of \$329.00. Thus the correctness score is 10, The confidence in this score is 10.

Evaluation: True

C.3 MBPP

Question: Write a function to count all the distinct pairs having a difference of k in any array.

Test cases:

```
assert countpairs([1, 5, 3, 4, 2], 5, 3) == 2
assert countpairs([8, 12, 16, 4, 0, 20], 6, 4) == 5
assert countpairs([2, 4, 1, 3, 4], 5, 2) == 3
```

Agent₀

Thought: To solve this problem, I need to find all the unique pairs (a, b) in the array such that the absolute difference between a and b is k. To do this efficiently, I can use a set to keep track of the elements in the array for constant-time lookups. Then, I can iterate through the array and for each element,

I can check if the element plus k or the element minus k exists in the set. If it does, it constitutes a distinct pair with the required difference. I should also ensure that I count each pair only once.

Action:

```
def countpairs(arr, n, k):
    unielements = set(arr)
    count = 0
    for number in unielements:
        if (number + k) in unielements:
            count += 1
        if (number - k) in unielements:
            count += 1
    return count // 2
```

Observation:

Tests passed:

```
assert countpairs([1, 5, 3, 4, 2], 5, 3) == 2
assert countpairs([8, 12, 16, 4, 0, 20], 6, 4) == 5
```

Tests failed:

```
assert countpairs([2, 4, 1, 3, 4], 5, 2) == 3 while
output: 2
```

Passed 2 out of 3 tests. Failed 1 out of 3 tests.

Validation: The code did not pass the last test case. The issue lies in the logic used to count the pairs. The current implementation incorrectly counts a pair twice if both 'number + k' and 'number - k' exist in the set of unique elements. However, this approach does not account for duplicate elements in the array, which can lead to incorrect pair counts.

Assessment: The code is almost correct but fails one test case due to a logical error in counting pairs. Thus the correctness score is 7. The confidence in this score is 10.

Agent₁

Thought: The previous code does not correctly handle duplicates where the difference between numbers should be counted multiple times if there are multiple occurrences of a number. I should use a dictionary to keep track of the occurrences of each number as we iterate through the array.

Action:

```
def countpairs(arr, n, k):
    count = 0
    seen = {}
    for num in arr:
        if num + k in seen:
            count += seen[num + k]
        if num - k in seen:
            count += seen[num - k]
        if num in seen:
            seen[num] += 1
```

```
else:  
seen[num] = 1  
return count
```

Observation:

Tests passed:

```
assert countpairs([1, 5, 3, 4, 2], 5, 3) == 2  
assert countpairs([8, 12, 16, 4, 0, 20], 6, 4) == 5  
assert countpairs([2, 4, 1, 3, 4], 5, 2) == 3
```

Tests failed:

Passed 3 out of 3 tests. Failed 0 out of 3 tests.

Validation: Since the code has passed all the tests and there are no observed errors or failures, I suggest the evaluator to pass the code.

Assessment: The code is correct and passes all the test cases. Thus the correctness score is 10. The confidence in this score is 10.

Evaluation: True