Selecting from Multiple Strategies Improves the Foreseeable Reasoning of Tool-Augmented Large Language Models

Yongchao Wu(✉) and Aron Henriksson

Stockholm University, NOD-huset, Borgarfjordsgatan 12, 16455 Stockholm, Sweden
{yongchao.wu,aronhen}@dsv.su.se

Abstract. Large language models (LLMs) can be augmented by interacting with external tools and knowledge bases, allowing them to overcome some of their known limitations, such as not having access to up-to-date information or struggling to solve math problems, thereby going beyond the knowledge and capabilities obtained during pre-training. Recent prompting techniques have enabled tool-augmented LLMs to combine reasoning and action to solve complex problems with the help of tools. This is essential for allowing LLMs to strategically determine the timing and nature of tool-calling actions in order to enhance their decision-making process and improve their outputs. However, the reliance of current prompting techniques on a single reasoning path or their limited ability to adjust plans within that path can adversely impact the performance of tool-augmented LLMs. In this paper, we introduce a novel prompting method, whereby an LLM agent selects and executes one among multiple candidate strategies. We assess the effectiveness of our method on three question answering datasets, on which it outperforms state-of-the-art methods like ReWOO, while also being a competitive and more cost-efficient alternative to ReAct. We also investigate the impact of selecting a reasoning trajectory from different strategy pool sizes, further highlighting the risks in only considering a single strategy.

Keywords: Large language models · Tool-augmented language models · Chain-of-thought prompting · Question answering

1 Introduction

The advanced capabilities of large language models (LLMs) [9] have extended their utility beyond mere language generation tasks, paving the way for their application as autonomous agents to make decisions across diverse environments [4,8]. Reasoning is crucial for autonomous agents in their the decision-making processes, particularly in scenarios involving tool usage to determine the appropriate timing and selection of tools for completing complex tasks. The Chain-of-Thought (CoT) prompting paradigm has become a prominent method for

A. Bifet et al. (Eds.): ECML PKDD 2024, LNAI 14943, pp. 197–212, 2024.
https://doi.org/10.1007/978-3-031-70352-2_12

enhancing the reasoning abilities of LLMs. By prompting LLMs to generate intermediate reasoning steps prior to delivering a final answer, CoT has significantly improved their performance in tasks requiring arithmetic, commonsense, and symbolic reasoning [13]. Recent prompting strategies like `ReAct` [17] and `ReWOO` [14] are two state-of-the-art (SOTA) methods that apply the CoT paradigm with tool-augmented LLMs (TA-LLMs). When tackling tasks with TA-LLMs, `ReAct` enhances the dynamism of CoT by generating reasoning thoughts based on environmental observations (observation-dependent reasoning) before deciding which tool to utilize in each step. Conversely, `ReWOO` transforms the intermediate reasoning steps of CoT into actionable plans (foreseeable reasoning), strategically determining the sequence of tool usage. The development of TA-LLMs has allowed some of the limitations of traditional LLMs to be addressed, such as restricted access to knowledge obtained during pre-training [6] and a tendency to hallucinate [5]. Unlike Retrieval-Augmented Generation (RAG) [7], which aims to reduce LLM hallucination by acquiring knowledge from a static external knowledge base that is indexed offline, TA-LLMs operate entirely online. They leverage external tools, such as software APIs, to access up-to-date information, offering greater flexibility by enabling the resolution of more complex problems through sophisticated API calls.

A limitation of current prompting techniques for TA-LLMs is that they either rely on a single reasoning path or can only adjust plans within the same reasoning trajectory. Since no individual reasoning path is infallible and can result in incorrect model output, not taking into consideration multiple reasoning trajectories may impede the performance of TA-LLMs. Hence, we introduce a novel prompting method named `ReMSV` (**Re**asoning with **M**ultiple **S**trategies and **V**oting) that generates and considers multiple reasoning trajectories before deciding on a course of action, including which tools to use. Specifically, building on the `ReWOO` framework, we design a process that incorporates the roles of *Director, Voter, Worker,* and *Solver* to generate multiple candidate reasoning trajectories, selects the trajectory with the most votes, and then executes the chosen reasoning trajectory to solve a given task. To assess the effectiveness of our method, we use several benchmark datasets, namely *HotpotQA* [15], *GSM4K* [2], and *PhysicsQA* [1]. `ReMSV` obtains a superior performance, surpassing `ReAct` in *HotpotQA* and *PhysicsQA*, while consistently outperforming `ReWOO` across all three benchmarks. Significantly, on the *HotpotQA* dataset, `ReMSV` achieves a relative improvement of 34.5% and 9.1% in comparison to `ReAct` and `ReWOO`, respectively. In addition, compared to `ReAct`, `ReMSV` provides a more cost-efficient alternative that consumes significantly fewer tokens, e.g., 200% fewer in the *HotpotQA* benchmark. We also carry out experiments to investigate the influence of considering multiple reasoning trajectories, underscoring the efficacy of our approach. To summarize, our key contributions are as follows:

– We present a prompting method called `ReMSV` that considers multiple reasoning trajectories. By selecting one among multiple sampled strategies through voting, our method not only consistently surpasses `ReWOO` but also generally outperforms ReAct, showcasing its effectiveness compared to current SOTA prompting strategies.

– We further analyze the proposed multi-strategy mechanism, both mathematically and empirically, showing that although ReMSV comes with a slightly higher cost compared to ReWOO, it is more cost-effective in terms of token consumption than ReAct, making it a viable alternative that improves performance at a slightly higher cost compared to ReWOO.

2 Related Work

Historically, the study of autonomous agents that can use external tools has primarily centered on reinforcement learning techniques. For example, WebGPT [8], which was designed to interact with web browsers to respond to complex questions, relies heavily on costly human feedback for its reinforcement learning process. Similarly, SimpleTOD [4], a task-focused dialogue system, requires extensive datasets derived from human feedback for its policy training. Prior to the introduction of CoT [13], reasoning capability was viewed as a primary constraint of LLMs that could not be addressed merely by scaling and enlarging the model size [10]. With few-shot in-context learning, CoT unlocks the reasoning capabilities of LLMs by incorporating sequential intermediary reasoning steps before producing the final result. Recently proposed prompting strategies [11,14,17] combine the reasoning and action capabilities of LLMs by converting the static intermediary reasoning phases into actionable plans that engage with the environment. Among these prompting strategies, ReAct [17] and ReWOO [14] are two approaches that achieve SOTA performance results in challenges that require several reasoning steps to conclude a final response. Specifically, ReAct [17] introduced an (*Obs, Thought, Action*) prompting technique, which consistently produces a *Thought* (verbal reasoning) based on environmental observations prior to executing task-specific actions. This method seamlessly connects the reasoning of an LLM with its actions, allowing it to interweave reasoning trajectories with tool-calling actions. Despite its remarkable performance, ReAct continuously generates observations and loops them back into the LLMs as context to generate the next (thought, action) pair. This leads to high token consumption, which can significantly increase cost and energy consumption. ReWOO utilizes the foreseeable reasoning capabilities of TA-LLMs to sketch a strategy for decomposing the problem into reasoning and actionable plans, without needing to resort to explicit observations. By compartmentalizing step-wise reasoning and tool-calling actions into distinct modules, it not only matches the performance of ReAct but also boasts a 5-fold increase in token efficiency.

A common limitation of both ReAct and ReWOO is their inability to account for multiple reasoning trajectories. Researchers have also explored the use of multi-reasoning trajectories with LLMs, such as self-consistency (SC) [12] and Tree-of-Thoughts (ToT) [16]. However, these methods were originally proposed to improve CoT in scenarios not involving tool calling. Moreover, these prompting strategies are computationally expensive and result in a substantial cost due to the necessity of executing each sampled reasoning trajectory. Our proposed method aims to combine the advantages of considering multiple reasoning trajectories with a cost-effective approach that avoids executing every trajectory. To achieve this, we are introducing a new method named ReMSV.

3 Methods

In this section, we formalize the problem and introduce the main components of the proposed method, ReMSV. Unlike ReAct and ReWOO, ReMSV considers multiple reasoning trajectories before deciding on a course of action involving tool-calling operations. Specifically, building on the ReWOO framework, we formulate a procedure that integrates the functions of *Director, Voter, Worker*, and *Solver*. Specifically, *Director* creates several potential reasoning trajectories/strategies[1], *Voter* reflects on which strategy is the best before deciding on which one to execute, and *Worker* subsequently executes the selected strategy. Compared to our approach, ReWOO only considers a single reasoning path, which could result in an erroneous strategy formulation.

Figure 1 illustrates the differences between ReAct, ReWOO, and ReMSV. Moreover, unlike SC that executes all (n) strategies and combines the outcomes through aggregation, leading to costs n times higher than ReWOO, our method solely executes the voted strategy after reflection, resulting in a more cost-efficient approach. Please note that SC has not been applied to tool-calling scenarios in any previous studies, primarily due to its high cost. In this study, we only estimate the cost and performance of SC in Sects. 6.1 and 6.2.

3.1 Problem Formulation

We explore the use of an LLM as an autonomous agent for handling tasks in text-based settings using external tools. Initially, the agent is equipped with the permissible actions $\mathcal{A}$ in the environment and a textual task directive $g \in \mathcal{G}$ from the task space $\mathcal{G}$. To accomplish the task g, the LLM navigates through a generated sequence of policies $[p_0, p_1, \cdots, p_n]$ to execute tool-calls. At time step t, the agent adheres to policy $p = \pi(a_t|c_t)$ to perform an action, where c_t is a trajectory context which contains observations $\mathcal{O}$ from the environment or evidence $\mathcal{E}$ from the tool-call executions in prior steps. Current SOTA methods, such as ReAct, first generate an initial policy based on the initial observation of the environment $p_0 = \pi(a_1|g, o_1)$, and formulate subsequent action policies by reasoning over the environmental observation at each step in an ad-hoc fashion, i.e. it lacks an overarching strategy for solving the problem. However, when scrutinizing how humans navigate and solve complex problems, we frequently choose a course of action from several viable alternative strategies. We design a similar framework which consists of the essential components *Director, Voter, Worker and Solver*, described below and illustrated in Fig. 2.

[1] In this paper, we use the terms reasoning trajectory and strategy interchangeably.

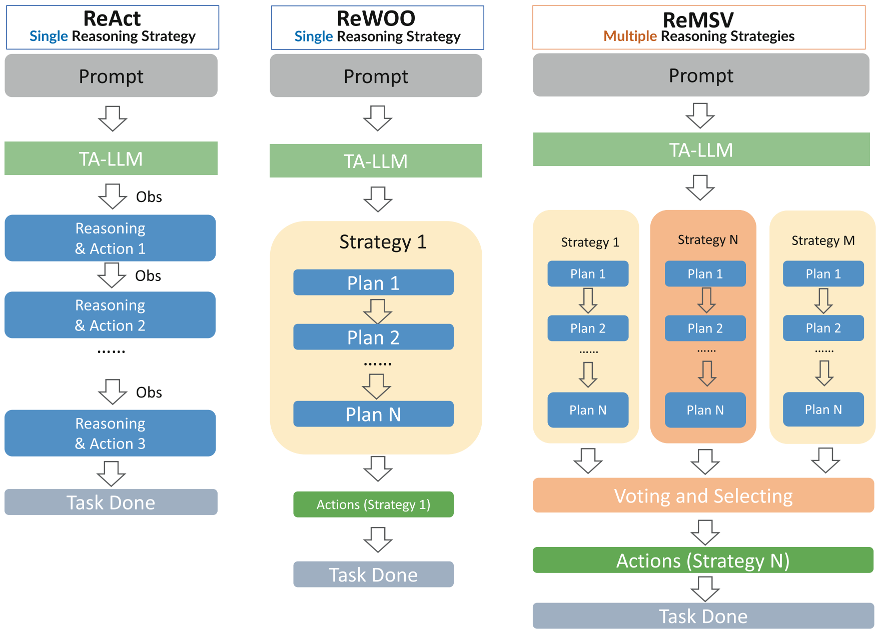


Fig. 1. A comparison of prompting techniques based on a single reasoning trajectory (`ReAct`, `ReWOO`) and multiple reasoning trajectories (`ReMSV`).

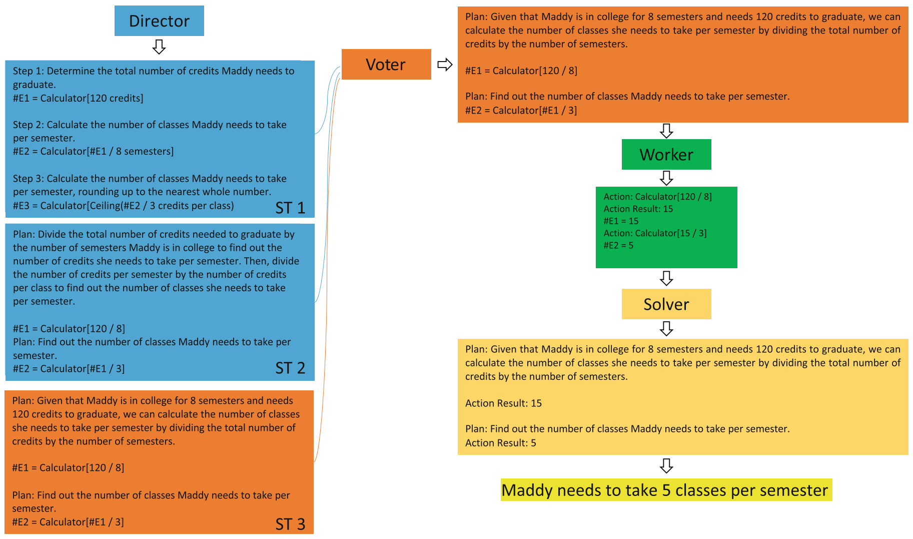
3.2 Method Components

Below, we describe the main components of the proposed method `ReMSV`, namely *Director*, *Voter*, *Worker*, and *Solver*.

Director: Strategy Making and Sampling. To produce n strategies, an LLM initially receives several pre-defined explicit `CoT` exemplars, illustrating the structure of a strategy $[(Plan_1, \mathcal{E}_1), (Plan_2, \mathcal{E}_2), \cdots, (Plan_n, \mathcal{E}_n)]$, which involves a series of step-by-step plans to tackle the problem. Specifically, for each step $(Plan_t, \mathcal{E}_t)$ of a strategy, it contains an instruction $Plan_t$ indicating the corresponding action a_t as well as a marking token $(\mathcal{E}_t)$ to store the execution result of action a_t, which, in turn, provides context for subsequent steps. Then, akin to the `SC` [12] method, we sample a variety of candidate outputs from the LLM, thereby creating a diverse collection of potential strategies.

Voter: Strategy Evaluation and Voting. Motivated by `ToT` [16], we incorporate a *Voter* component to assess various strategies created by the *Director*, utilizing a straightforward zero-shot voting prompt (*"Analyze the strategies and conclude the most promising one to solve the problem"*). We sample n candidates from the voting outputs and choose the strategy with the highest number of votes as the concluding strategy to be executed. We opt for multiple rounds (5) of voting instead of a single vote to enhance the robustness and generalizability of the voting procedure.

Worker and Solver: Strategy Execution. We use the `ReWOO` framework to implement the selected strategy following the voting process. Specifically, *Workers* adhere to the plans at each step $(Plan_t, \mathcal{E}_t)$ in the strategy to execute tool-calls from the action space $\mathcal{A}$, and the execution result will substitute the marking token $\mathcal{E}_t$, acting as observations or evidence. The *Solver* compiles all plans and observations to produce the final output.



Step 1: Determine the total number of credits Maddy needs to graduate.
#E1 = Calculator[120 credits]

Step 2: Calculate the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 8 semesters]

Step 3: Calculate the number of classes Maddy needs to take per semester, rounding up to the nearest whole number.
#E3 = Calculator[Ceiling(#E2 / 3 credits per class)]

Plan: Divide the total number of credits needed to graduate by the number of semesters Maddy is in college to find out the number of credits she needs to take per semester. Then, divide the number of credits per semester by the number of credits per class to find out the number of classes she needs to take per semester.

#E1 = Calculator[120 / 8]
Plan: Find out the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 3]

Plan: Given that Maddy is in college for 8 semesters and needs 120 credits to graduate, we can calculate the number of classes she needs to take per semester by dividing the total number of credits by the number of semesters.

#E1 = Calculator[120 / 8]

Plan: Find out the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 3]

Plan: Given that Maddy is in college for 8 semesters and needs 120 credits to graduate, we can calculate the number of classes she needs to take per semester by dividing the total number of credits by the number of semesters.

#E1 = Calculator[120 / 8]

Plan: Find out the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 3]

Action: Calculator[120 / 8]
Action Result: 15
#E1 = 15
Action: Calculator[15 / 3]
#E2 = 5

Plan: Given that Maddy is in college for 8 semesters and needs 120 credits to graduate, we can calculate the number of classes she needs to take per semester by dividing the total number of credits by the number of semesters.

Action Result: 15

Plan: Find out the number of classes Maddy needs to take per semester.
Action Result: 5

Maddy needs to take 5 classes per semester

Fig. 2. The workflow of `ReMSV`. The question is: *Maddy is in college for 8 semesters. She needs 120 credits to graduate. If each class is 3 credits, how many classes does she need to take per semester?* The *Director* generates multiple candidate strategies, from which the *Voter* selects one, the *Worker* executes the tool-calls, and the *Solver* compiles all plans and observations to produce the final output.

4 Token Consumption Estimation

In order to compare our proposed method, `ReMSV`, to `ReAct` and `ReWOO` in terms of cost efficiency, we conduct a mathematical analysis to estimate their token consumption. Let $N(p)$ denote the tokens for a text p. Suppose a TA-LLM addressing a question $\mathcal{Q}$ requires k steps of reasoning to arrive at the correct answer. Initially, the TA-LLM is provided with a context prompt $\mathcal{C}$ and several in-context learning examples $\mathcal{I}$. In the process of using the `ReAct` prompting approach, the TA-LLM persistently produces observations $\mathcal{O}$, which are then fed back into the TA-LLM as context to create subsequent pairs of thoughts ($\mathcal{T}$) and actions ($\mathcal{A}$). The token consumption of `ReAct` can be calculated as:

$$\text{Token}^{\texttt{ReAct}} = N(\mathcal{C}+\mathcal{I}+\mathcal{Q}) + \sum_{i=1}^{k-1} N(\mathcal{C}+\mathcal{I}+\mathcal{Q}+\sum_{j=1}^{i}(\mathcal{T}_j+\mathcal{A}_j+\mathcal{O}_j)) \quad (1)$$

$$= kN(\mathcal{C}+\mathcal{I}+\mathcal{Q}) + \sum_{i=1}^{k-1}(k-i)N(\mathcal{T}_i+\mathcal{A}_i+\mathcal{O}_i) \quad (2)$$

`ReWOO` leverages the foreseeable reasoning abilities of TA-LLMs to outline a strategy for breaking down the problem into actionable plans and evidences $(Plan_n, \mathcal{E}_n)$, eliminating the need for explicit observations. The token consumption of `ReWOO` is:

$$\text{Token}^{\texttt{ReWOO}} = N(\mathcal{C}_{\text{planner}}+\mathcal{I}+\mathcal{Q}) + N(\mathcal{C}_{\text{solver}}+\mathcal{Q}+\sum_{i=1}^{k}(P_i+E_i)) \quad (3)$$

$$\approx 2\,N(\mathcal{C}+\mathcal{Q}) + N\mathcal{I} + \sum_{i=1}^{k} N(P_i+E_i) \quad (4)$$

`ReMSV` adds two additional components, *Director* and *Voter*, to sample and vote for m strategies. The token consumption of `ReMSV` can be calculated as:

$$\text{Token}^{\texttt{ReMSV}} = N(\mathcal{C}_{\text{Director}}+\mathcal{I}+\mathcal{Q}) + m\sum_{i=1}^{k} N(P_i+E_i) + N(\mathcal{C}_{\text{Voter}}+\mathcal{Q}) \quad (5)$$

$$+ N(\mathcal{C}_{\text{solver}}+\mathcal{Q}+\sum_{i=1}^{k}(P_i+E_i)) \quad (6)$$

$$\approx 3\,N(\mathcal{C}+\mathcal{Q}) + N(\mathcal{I}) + (m+1)\sum_{i=1}^{k} N(P_i+E_i) \quad (7)$$

As indicated by the formulas above, $\text{Token}^{\texttt{ReAct}}$ scales linearly with the size of $(\mathcal{C}, \mathcal{I}, \mathcal{Q})$ by a factor of k and quadratically with the size of $(\mathcal{T}_i, \mathcal{A}_i, \mathcal{O}_i)$ in terms of k. On the other hand, $\text{Token}^{\texttt{ReWOO}}$ increases linearly with the size of $(\mathcal{C}, \mathcal{I}, \mathcal{Q})$ due to constant factors and linearly with the size of $(P_i + E_i)$. Similarly, $\text{Token}^{\texttt{ReMSV}}$ also grows linearly with the size of $(\mathcal{C}, \mathcal{I}, \mathcal{Q})$, albeit with a slightly larger constant factor, and linearly with the size of $(P_i + E_i)$, multiplied by a factor of m. The analysis indicates that with `ReAct`, token consumption escalates substantially as the number of reasoning steps grows. In contrast, `ReWOO` and `ReMSV` do not suffer from this issue, maintaining efficiency in token usage regardless of the increase in reasoning steps. Nevertheless, `ReMSV` will consume more tokens than `ReWOO` due to sampling m multiple strategies. We also empirically evaluate actual token consumption in the results Sect. 6.

5 Experiments

We evaluate our approach against SOTA methods `ReAct` and `ReWOO` using question answering (QA) datasets related to general knowledge and arithmetic reasoning that require multiple reasoning steps.

5.1 Benchmarks

The following three datasets are used to evaluate our approach.

- *HotpotQA* is a dataset that comprises multi-hop reasoning questions, which, based on Wikipedia, require the identification and analysis across various supporting documents to generate an answer [15].
- *GSM8K* is a dataset featuring linguistically varied school math word problems, crafted by human problem composers. Solving these problems typically involves a series of elementary calculations employing basic arithmetic operations $(+ - \times \div)$ to obtain the final answer [2].
- *PhysicsQA* focuses on high school physics questions that test knowledge in areas such as Newton's second law, force identification, kinematics, and so forth [1].

5.2 Baselines

The following SOTA methods, which allow LLMs to engage with external tools, serve as baselines.

- `ReAct` [17] represents a prompting strategy that combines reasoning and actions to resolve language reasoning and decision-making tasks by interacting with the external environment. Specifically, it uses a *(Thought, Act, Obs)* prompting approach to adjust action plans according to the external environment.
- `ReWOO` [14] is designed with a *Plan - Work - Solve* strategy to leverage the foreseeable reasoning capabilities of LLMs without relying on explicit observations.

5.3 Action Space

We have designed the task-specific action space for TA-LLMs to interact with the following APIs. Considering the various characteristics of the benchmark datasets, action spaces are configured differently as shown in Table 1.

- `LLM-Tool`. Considering that the term "tools" in this context specifically refers to external tools, we employ LLM-tool, a separate LLM[2] that can process language reasoning tasks, which is consistent with the experiment setting of the `ReWOO` paper [14].

[2] GPT-3.5-Turbo is used in our experiment.

- `Wikipedia Search`. The search API for Wikipedia[3] pages to query knowledge from Wikipedia's database.
- `Google Search`. The search API from Google SERP[4] service to query knowledge from the Internet.
- `WolframAlpha`. The complex mathematical computation result from WolframAlpha[5].
- `Calculator`. A simple calculator[6] to perform basic arithmetic operations.

Table 1. Action space configurations for different benchmark datasets

Dataset	No. Tools	Action Space
HotpotQA	2	LLM-tool, Wiki
GSM8K	3	LLM-tool, WolframAlpha, Calculator
PhysicsQA	5	LLM-tool, WolframAlpha, Calculator, Wiki, Google

5.4 Evaluation Metrics

Evaluation metrics commonly used to assess QA performance of LLMs [3,13,14,17], including exact match (EM), character-level F_1 scores, and accuracy, are used in this study.

- `EM`. The model's prediction is checked whether it precisely matches the correct answer in the reference data. If it does, the score is 1 (100%); if not, it is 0 (0%). EM scores are not presented for *GSM8K* and *PhysicsQA* because the inclusion of strings alongside numbers interferes with accurate matching, struggling to accurately represent true performance.
- `F`$_1$. The character-level F_1-score evaluates how accurately the model predicts individual characters compared to the ground truth, with a score of 1 indicating perfect accuracy.
- `Accuracy`. Since the output of LLMs vary syntactically, accuracy is determined by using GPT-4 to evaluate the predictions against the ground truth, giving a binary score of 1 or 0.
- `Token Consumption`. We calculate the average token consumption per question, incorporating both prompting tokens and text completion tokens, to assess the cost-effectiveness of each prompting strategy.

[3] https://www.mediawiki.org/wiki/API.
[4] https://serpapi.com/search-api.
[5] https://products.wolframalpha.com/api.
[6] https://js.langchain.com/docs/api/tools_calculator/.

5.5 Experimental Setup

For both the baselines and our method, `CoT` exemplars are employed to assist the LLM in generating strategies for executing tool-calls. The same task-dependent exemplar questions (released by `ReAct`, `ReWOO`) were utilized across the three datasets for both the baseline methods and our approach to ensure fair comparisons. Specifically, for *HotpotQA*, six exemplars (six-shot) are used. While for *GSM8K* and *PhysicsQA*, one exemplar (one-shot) is used, which is consistent with the experiment settings in the `ReWOO` paper [14]. For the LLM-Agent, we employ *GPT-3.5-Turbo*[7] as the base model. Considering the cost, we assess the benchmark performance of our method and the baselines by utilizing 500 randomly chosen samples from *HotpotQA*, *GSM8K*, and all examples (57) from *PhysicsQA*. We sample 3 strategies when conducting the general benchmarking experiment. Subsequently, we conduct additional experiments to investigate the effects of varying strategy counts (from 2 to 5) and to evaluate the efficiency of individual strategies using 50 randomly sampled questions from *HotpotQA*, *GSM8K* and *PhysicsQA*.

6 Results

We present the results of our experiments in three parts. First, we benchmark our proposed method on three QA datasets vs. the baseline methods. We then investigate the impact of the multi-strategy mechanism in an attempt to explain why the proposed method outperforms the baselines. Finally, we conduct an error analysis to identify situations in which the proposed method fails and how.

6.1 Benchmarking Prompting Methods

Table 2 presents the experimental results of the baselines and our method on *HotpotQA*, *GSM8K*, and *PhysicsQA*. Our approach outperforms `ReAct` on *HotpotQA* and *PhysicsQA*, while consistently outperforming `ReWOO` on all three datasets.

Notably, our approach delivers the best performance on *HotpotQA* with an accuracy of 54.6, achieving a relative improvement of 44.4% and 4.5% in accuracy, 38.6% and 9.6% in F_1, and 34.5% and 9.1% in EM when compared to `ReAct` and `ReWOO`, respectively. `ReAct` yields the best results in *GSM8K*, with our method securing the second-highest performance. Our approach consistently outperforms `ReWOO` across all three benchmark datasets, demonstrating that incorporating the multi-strategy mechanism enhances overall performance. Regarding token costs, `ReMSV` is shown to be more cost-effective in terms of token consumption than `ReAct`, although it does use more tokens than `ReWOO`. More precisely, `ReMSV` uses roughly 50% more tokens than `ReWOO` across the benchmarks. However, it uses significantly fewer tokens compared to `ReAct`. In particular, `ReMSV` requires over 200% fewer tokens than `ReAct` in benchmarks such as *HotpotQA*. This result is in line with the cost estimations detailed in Sect. 4 that showed

[7] https://platform.openai.com/docs/models/gpt-3-5. access on 1st, Sep, 2023.

that ReMSV and ReWOO scale well for complex problems requiring many reasoning steps. Moreover, employing SC, which executes all strategies, would lead to an estimated cost that is threefold higher than ReWOO and almost double that of ReMSV.

Table 2. Predictive performance of ReAct, ReWOO, and ReMSV across three QA datasets. The best performance is marked in bold, while the second best results are underlined. We also report the average number of reasoning steps for each benchmark.

Dataset	Method	Acc	F_1	EM	# Tokens	Steps
HotpotQA	ReAct	37.8	33.7	28.4	6,771	5.4
	ReWOO	<u>52.2</u>	<u>42.6</u>	<u>35.0</u>	**1,447**	4.1
	ReMSV	**54.6**	**46.7**	**38.2**	<u>2,194</u>	4.3
GSM8K	ReAct	**65.2**	**35.8**	N/A	1,662	3.2
	ReWOO	59.4	28.9	N/A	**759**	3.5
	ReMSV	<u>62.6</u>	<u>29.6</u>	N/A	<u>1,359</u>	3.2
PhysicsQA	ReAct	32.0	11.0	N/A	2,252	2.8
	ReWOO	<u>35.0</u>	<u>12.0</u>	N/A	**908**	3.0
	ReMSV	**36.0**	**13.0**	N/A	<u>1,548</u>	2.9

6.2 Impact of the Multi-strategy Mechanism

We conduct further experiments to explore the impact of the multi-strategy mechanism, particularly in terms of the size of the strategy pool and the effectiveness of individual strategies. These experiments are described below.

Effect of Strategy Pool Size. To explore the impact of varying the number of strategies created by the *Director*, we conducted an experiment on *HotpotQA* using the same 500 samples, varying the strategy pool size from two to five. Table 3 demonstrates that, in general, our method, employing between two to five strategies, surpasses ReWOO, which relies on executing a single strategy. We also observe that our method yields improved results when choosing either 3 or 4 strategies. This suggests that with too few strategies (i.e. one or two), there is a risk that a more optimal strategy might be overlooked. Conversely, with an excessive number of strategies (like five), the likelihood increases that a sub-optimal strategy might be chosen given the quality of the *Voter*. We intend to further validate this in future work.

Efficacy of Individual Strategies. We conduct an experiment using a subset of 50 examples from each of *HotpotQA*, *GSM8K* and *PhysicsQA* to explore the performance of each individual strategy in the strategy pool, i.e. if they were

Table 3. Results when considering different numbers of strategies (STs). The best performance is highlighted in bold while the second best performance is underlined. Note that `ReWOO` corresponds to considering only a single strategy.

Method	No. STs	Acc	F_1	EM
ReWOO	1	52.2	42.6	35.0
ReMSV	2	52.2	43.1	34.6
	3	<u>54.6</u>	**46.7**	**38.2**
	4	**54.8**	<u>46.4</u>	<u>38.0</u>
	5	53.4	44.7	36.0

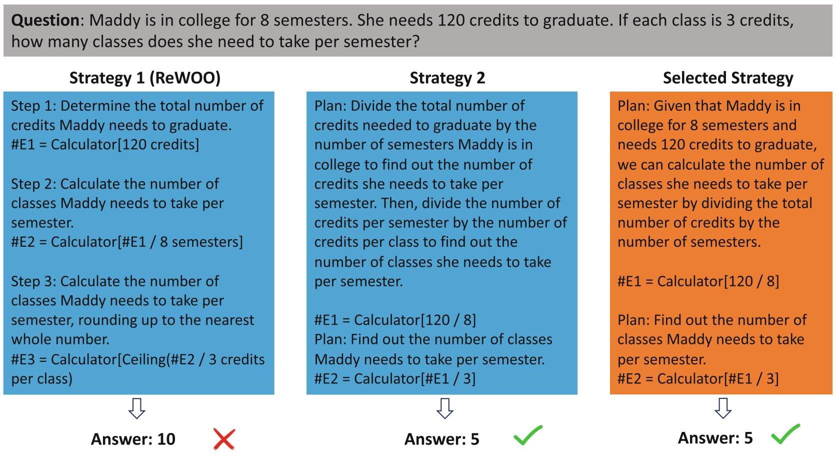
Table 4. Performance for selected and not selected strategies. The best performance is marked in bold, while runner-up performance is underlined. The average outcome for not selected and all strategies is given for comparison.

Method	Acc.	F_1
1st ST (ReWOO)	46.0	27.2
Selected ST (ReMSV)	**50.0**	**30.5**
Not selected STs	47.6	29.9
Avg. all STs	<u>48.9</u>	<u>30.2</u>

to be executed, as well as the effectiveness of the voting mechanism. To provide a comprehensive comparison between `ReMSV` and `ReWOO`, we present the average performance metrics across three datasets, focusing on the shared metrics `Accuracy` and $\texttt{F}_1$. In line with the results in Table 2, Table 4 shows that our approach using the selected strategy outperforms `ReWOO`, which essentially corresponds to selecting the first generated strategy. Additionally, the performance of the selected strategy also generally achieves superior performance to the average of the strategies that were not selected. From the data presented in Table 2 and Table 4, it is evident that `ReMSV` consistently surpasses `ReWOO`. Notably, the average results from the strategies that were not selected are also on par with or exceed the performance of `ReWOO`, which suggests that `ReMSV` effectively reduces the risk of executing sub-optimal strategies due to greedy sampling. Furthermore, the average results of both selected and non-selected strategies fall short of `ReMSV`, indicating that `ReMSV` would also outperform `SC`, which aggregates all executed strategies. Together with the cost estimation for `SC` in Sect. 6.1, without resorting to executing all strategies, `ReMSV` not only is able to surpass the performance of methods based on single trajectories, but also proves to be more cost-effective in terms of token consumption compared to `SC`. Figure 3 illustrates an instance where the single reasoning trajectory selection of `ReWOO` leads to an erroneous strategy implementation, yielding an incorrect prediction. Conversely, in this scenario, `ReMSV` casts votes across the sampled strategies and chooses one that ultimately produces the correct prediction.

6.3 Error Analysis

To gain insights into when and why our proposed method fails to produce the correct prediction, we conduct an error analysis on the incorrect predictions generated by `ReMSV`. We summarize the statistics of different sources of failure for these incorrect predictions in Table 5. Out of the 50 experimental examples in each dataset, `ReMSV` produces 21 incorrect predictions for *HotpotQA* and *GSM8K*,


Question: Maddy is in college for 8 semesters. She needs 120 credits to graduate. If each class is 3 credits, how many classes does she need to take per semester?

Strategy 1 (ReWOO)

Step 1: Determine the total number of credits Maddy needs to graduate.
#E1 = Calculator[120 credits]

Step 2: Calculate the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 8 semesters]

Step 3: Calculate the number of classes Maddy needs to take per semester, rounding up to the nearest whole number.
#E3 = Calculator[Ceiling(#E2 / 3 credits per class)

Answer: 10 ✗

Strategy 2

Plan: Divide the total number of credits needed to graduate by the number of semesters Maddy is in college to find out the number of credits she needs to take per semester. Then, divide the number of credits per semester by the number of credits per class to find out the number of classes she needs to take per semester.

#E1 = Calculator[120 / 8]
Plan: Find out the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 3]

Answer: 5 ✓

Selected Strategy

Plan: Given that Maddy is in college for 8 semesters and needs 120 credits to graduate, we can calculate the number of classes she needs to take per semester by dividing the total number of credits by the number of semesters.

#E1 = Calculator[120 / 8]

Plan: Find out the number of classes Maddy needs to take per semester.
#E2 = Calculator[#E1 / 3]

Answer: 5 ✓

Fig. 3. An example from GSM8K illustrating different sampled strategies: the not selected strategies are colored in blue, while the selected strategy is colored in orange. (Color figure online)

Table 5. *Director* failure indicates that the strategy pools contains no successful strategy. *Worker* failure indicates that the wrong prediction is due to a failed strategy execution, resulting in an inability to produce a valid answer. *Voter* failure indicates that an unsuccessful strategy is selected when there is at least one successful strategy in the strategy pool. Note that there can be more than one source of failure.

	HotpotQA	GSM8K	PhysicsQA
Incorrect predictions	21	21	33
- *Director* failure	17	12	29
- *Voter* failure	2	2	1
- *Worker* failure	4	9	13

and 33 for *PhysicsQA*. Most incorrect predictions stem from *Director* failures, indicating that none of the generated strategies in the strategy pool leads to a correct prediction. This suggests that there is potential for improvement in the generation of strategies, although it could also mean that these particular questions are challenging to decompose based on the foreseeable reasoning capabilities of current LLMs. Specifically, `ReMSV` tends to incur more *Director* failures on *HotpotQA* and *PhysicsQA* than on *GSM8K*. Conversely, *Worker* failures occur more frequently in *GSM8K* and *PhysicsQA* than in *HotpotQA*. These findings suggest that for *HotpotQA* and *PhysicsQA*, ReMSV more often generates unsuccessful strategies, whereas in *GSM8K* and *PhysicsQA* – where more tools are utilized – ensuring the proper functioning of tool calls is critical. We also report *Voter* failures, where, across all benchmarks, only 1 or 2 failures are due to not selecting a successful strategy, indicating that, compared to other components in ReMSV, the *Voter* performs more robustly. The overall error analysis implies that there is room for task-specific enhancements in future work. For instance, enhancing the sampling quality for tasks such as *HotpotQA* and *PhysicsQA*, or improving the stability of tool calling for tasks like *GSM8K* and *PhysicsQA*, could lead to significant improvements.

7 Discussion

In this section, we discuss the results of our experiments, focusing especially on the pros and cons of prompting techniques based on observation-dependent reasoning vs. foreseeable reasoning and single vs. multiple reasoning trajectories. We also discuss possible directions for future work.

7.1 Observation-Dependent Reasoning Vs. Foreseeable Reasoning

Prompting strategies based on observation-dependent reasoning, such as ReAct, continuously observe the environment – here in the form of tool-calling output – and relay these observations back to the LLM to formulate relevant reasoning and action plans. Such prompting strategies are highly responsive to environmental changes and devise action plans in an ad-hoc, dynamic manner. Consequently, while such prompting strategies perform well and allow an LLM to interact with external tools to solve complex reasoning problems, they tend to incur a relatively high cost by requiring more interactions with the LLM-Agent and hence a higher token consumption since the observation in each iteration is fed to the LLM to generate the next thought and action. Prompting methods based on foreseeable reasoning, like ReWOO and ReMSV, instead devise a strategy up-front that includes both reasoning and action plans, negating the necessity to continuously provide observations and interacting with the LLM-Agent. Therefore, prompting strategies based on foreseeable reasoning are more cost-effective and can play a crucial role in the development of efficient TA-LLMs. ReWOO and ReMSV, which both rely on the foreseeable reasoning capabilities of LLMs, perform on par with or outperform ReAct, which is based on observation-dependent reasoning. The results on *GSM8K* show, however, that in some situations, ReAct leads to better performance. One possible explanation could be that certain problems are not readily decomposable up-front without first obtaining more information through tool-calls. Such problems may require dynamic reasoning and are perhaps beyond the foreseeable reasoning capabilities of current LLMs.

7.2 Single Vs. Multiple Reasoning Trajectories

Both ReAct and ReWOO rely on a single reasoning trajectory, which might misguide the TA-LLMs into producing incorrect predictions. Prompting methods based on multiple reasoning trajectories, like ToT, on the other hand, are computationally expensive and incur a relatively high cost by needing to execute each sampled reasoning trajectory. ReMSV is a prompting method that aims to combine the advantages of prompting methods based on single and multiple reasoning trajectories, respectively. It does so by generating and taking into consideration multiple reasoning trajectories, while operating in a cost-efficient manner by only selecting – based on LLM voting – and executing a single reasoning trajectory. Its cost-effectiveness stems also from leveraging the economical attributes of the foreseeable reasoning capabilities of LLMs, i.e. by generating strategies up-front rather than in an ad-hoc and dynamic fashion. The experiments demonstrate that `ReMSV` surpasses `ReAct` on *HotpotQA* and *PhysicsQA*, and consistently outperforms `ReWOO` across all three datasets. This clearly validates the efficacy of incorporating multiple reasoning trajectories, albeit in a cost-efficient manner. While `ReMSV` exploits the cost-efficient properties of `ReWOO`, there is a slightly higher cost of `ReMSV` that stems from generating a pool of strategies. However, as the experimental results show, it leads to improved predictive performance. Generally speaking, there is a likely trade-off to be made between the cost-efficiency of prompting based on single reasoning trajectories and the predictive performance of prompting based on multiple reasoning trajectories. While `ReMSV` attempts to strike a good balance in this regard, it may be possible to obtain higher predictive performance at a higher cost. One alternative in this direction is to execute all generated strategies and aggregate the results in a late fusion fashion. Although we did not fully evaluate this approach, the results in Table 4 indicate that this method would, in fact, not lead to higher predictive performance since the predictive performance of the selected strategy is higher than the average of the strategies that were not selected, as well as the average of all generated strategies. That said, there is certainly room for developing more sophisticated ensemble methods in this context.

8 Conclusions, Future Work, and Ethical Statement

In this paper, we propose a new prompting method, `ReMSV`, that leverages the foreseeable reasoning capabilities of LLMs to generate multiple strategies up-front, one of which is selected through LLM voting and strategically executed. `ReMSV` is benchmarked on three question-answering datasets and, overall, surpasses SOTA methods such as `ReAct` and `ReWOO`, and offers a more cost-effective solution compared to `ReAct`.

The present multi-reasoning trajectories of `ReMSV` operate solely on the strategy level. A compelling direction for future research could be to implement plan sampling at each reasoning step within the strategy to generate more diverse and better strategies. By generating more accurate and diverse strategies, ensemble methods are also more likely to be successful. Another avenue for future research might be to enhance the voting process. One possibility for improving the voting could be to incorporate few-shot learning into the voting mechanism to render it more task-aware.

Allowing LLMs to communicate with external environments can introduce certain dangers, such as potential data breaches or the generation of harmful actions. To mitigate these concerns in our experiments, we have imposed boundaries on interactions, restricting them to specific action spaces like Wikipedia and

WolframAlpha and confining the task to question answering on public benchmarks that does not access private or confidential information.

Acknowledgement. The computations and data handling were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725. Acknowledgment is also extended to the `ReWOO` project[8] for providing the code base used to conduct the experiments.([8]https://github.com/billxbf/ReWOO)

References

1. Beatty, I.D., Gerace, W.J., Leonard, W.J., Dufresne, R.J.: Designing effective questions for classroom response system teaching. Am. J. Phys. **74**(1), 31–39 (2006)
2. Cobbe, K., et al.: Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168 (2021)
3. Hao, S., Gu, Y., Ma, H., Hong, J.J., Wang, Z., Wang, D.Z., Hu, Z.: Reasoning with language model is planning with world model. arXiv preprint arXiv:2305.14992 (2023)
4. Hosseini-Asl, E., McCann, B., Wu, C.S., Yavuz, S., Socher, R.: A simple language model for task-oriented dialogue. Adv. Neural. Inf. Process. Syst. **33**, 20179–20191 (2020)
5. Ji, Z., et al.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12), 1–38 (2023)
6. Komeili, M., Shuster, K., Weston, J.: Internet-augmented dialogue generation. arXiv preprint arXiv:2107.07566 (2021)
7. Lewis, P., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Adv. Neural. Inf. Process. Syst. **33**, 9459–9474 (2020)
8. Nakano, R., et al.: Webgpt: Browser-assisted question-answering with human feedback. arXiv preprint arXiv:2112.09332 (2021)
9. OpenAI, R.: Gpt-4 technical report (arxiv: 2303.08774). View in Article (2023)
10. Rae, J.W., et al.: Scaling language models: Methods, analysis & insights from training gopher. arXiv preprint arXiv:2112.11446 (2021)
11. Shinn, N., Cassano, F., Labash, B., Gopinath, A., Narasimhan, K., Yao, S.: Reflexion: language agents with verbal reinforcement learning **4**. arXiv preprint arXiv:2303.11366 (2023)
12. Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., Zhou, D.: Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171 (2022)
13. Wei, J., et al.: Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural. Inf. Process. Syst. **35**, 24824–24837 (2022)
14. Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., Xu, D.: Rewoo: Decoupling reasoning from observations for efficient augmented language models. arXiv preprint arXiv:2305.18323 (2023)
15. Yang, Z., et al.: Hotpotqa: a dataset for diverse, explainable multi-hop question answering. arXiv preprint arXiv:1809.09600 (2018)
16. Yao, S., et al.: Tree of thoughts: Deliberate problem solving with large language models. arXiv preprint arXiv:2305.10601 (2023)
17. Yao, S., et al.: React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629 (2022)