Smurfs: Leveraging Multiple Proficiency Agents with Context-Efficiency for Tool Planning

Anonymous ACL submission

Abstract

The emergence of large language models (LLMs) has opened up unprecedented possibilities for automating complex tasks that are often comparable to human performance. Despite their capabilities, LLMs still encounter difficulties in completing tasks that require high levels of accuracy and complexity due to their inherent limitations in handling multifaceted problems single-handedly. This paper introduces "***Smurfs***," a cutting-edge multi-agent framework designed to revolutionize the application of LLMs. By transforming a conventional LLM into a synergistic multi-agent ensemble, Smurfs enhances task decomposition and execution without necessitating extra training. This is achieved through innovative prompting strategies that allocate distinct roles within the model, thereby facilitating collaboration among specialized agents. The framework gives access to external tools to efficiently solve complex tasks. Our empirical investigation, featuring the mistral-7b-instruct model as a case study, showcases Smurfs' superior capability in intricate tool utilization scenarios. Notably, Smurfs outmatches the ChatGPT-ReACT in the ToolBench I2 and I3 benchmark with a remarkable 84.4% win rate, surpassing the highest recorded performance of a GPT-4 model at 73.5%. Furthermore, through comprehensive ablation studies, we dissect the contribution of the core components of the multi-agent framework to its overall efficacy. This not only verifies the effectiveness of the framework, but also sets a route for future exploration of multi-agent LLM systems.

1 Introduction

Tool manipulation has traditionally been seen as a distinctive human characteristic, dating back approximately 2.5 million years (Oakley and Museum, 1972; Ambrose, 2001). For large language models (LLMs), access to external tools can equip them with broader capabilities beyond their fixed language modeling knowledge. For example, the search engine API empowers ChatGPT to access real-time information (Zhao et al., 2023). However, LLMs still face many challenges when attempting to use tools to solve tasks. These challenges include computational expense and a lack of adaptability to new tools (Hao et al., 2024; Guu et al., 2020; Qin et al., 2024).

This paper addresses the critical research problem of enhancing the problem-solving capabilities of LLMs through the adoption of a multi-agent system (MAS) framework (Dorri et al., 2018; Van der Hoek and Wooldridge, 2008). We posit that a MAS approach can significantly augment the efficacy of LLMs in handling tasks that require a high degree of precision, adaptability, and comprehensive knowledge integration.

	ToolBench-all	ToolBench-long	Increase
ChatGPT-DFSDT	71.5	66.0	−7.69%
GPT4-ReACT	72.0	65.1	−9.58%
GPT4-DFSDT	77.5	69.8	−9.94%
Mistral-Smurfs	**79.5**	**82.1**	+3.27%

Table 1: Pass rate of models on the ToolBench benchmark I2 category subset with long settings and all settings.

To this end, we introduce "***Smurfs***" an innovative MAS framework inspired by the collaborative and versatile nature of its namesake cartoon characters. The Smurfs framework is based on the principle that synergistic collaboration among specialized agents can overcome the limitations faced by individual LLMs. Each agent within the Smurfs framework is designed to perform specific sub-tasks, facilitating a more nuanced and effective approach to complex problem-solving. Our research delves into the architectural design, coordination mechanisms, and the operational dynamics of integrating specialized agents into a cohesive system. Through rigorous experimental evalua-

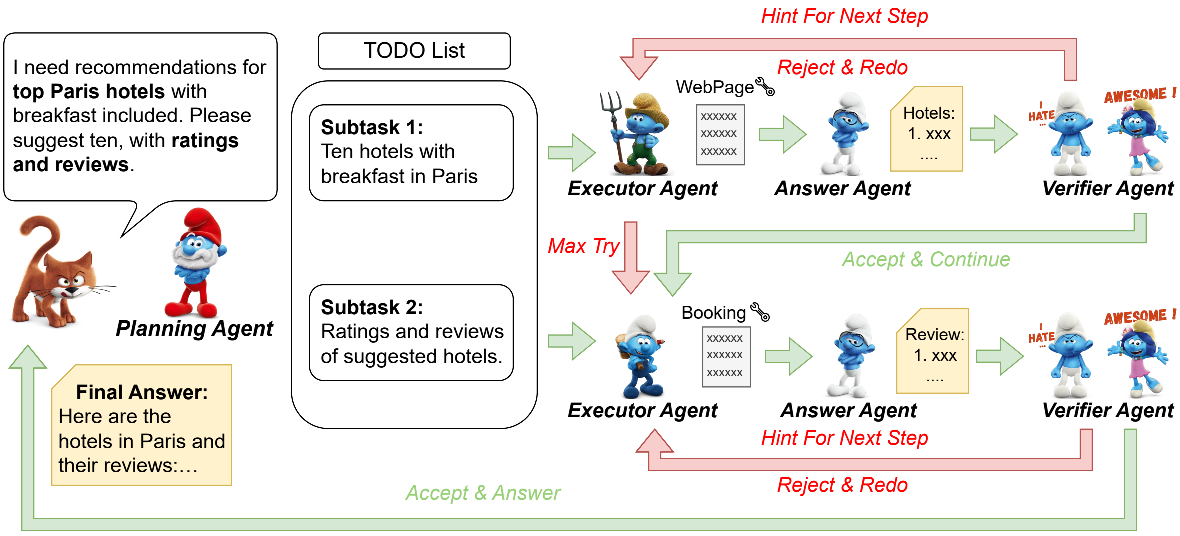

Figure 1: Demonstration of the whole process of the Smurfs framework.

tion, the Smurfs framework, utilizing the mistral-7b-instruct model (Jiang et al., 2023), achieved a remarkable 84.4% win rate against the benchmark set by ChatGPT-ReACT (Yao et al., 2022) on the ToolBench I2 and I3 benchmark (Qin et al., 2024). This outcome not only sets a new state-of-the-art in the field, but also provides concrete evidence of the effectiveness of the multi-agent approach in enhancing LLM capabilities.

The structure of this paper is as follows: Section 2 presents the motivation for utilizing a multi-agent system. The methodology employed within the framework is detailed in Section 3. Subsequently, Section 4 provides an in-depth evaluation of the experiments conducted on Smurfs. Section 5 then reviews the techniques currently related to our work. Lastly, we summarize and conclude our findings in Section 6.

2 Motivation

2.1 Limited Context Length in a Single Model

LLMs face considerable challenges when tasked with managing extensive contexts. As highlighted by (Liu et al., 2024), these limitations become particularly noticeable in tasks requiring assimilation and processing of large inputs, like verbose tool documents and API responses. The situation worsens when LLMs are supplemented with external information, such as document retrieval or online searching (Petroni et al., 2020; Ram et al., 2023; Mallen et al., 2022). Although numerous language models capable of handling larger contexts are emerging (Dai et al., 2019; Dao et al., 2022), they often face significant performance degradation when the important information is located at some positions (Liu et al., 2024; Shi et al., 2023). Moreover, within the MAS framework, the impact of extended contexts on performance remains unclear.

To measure the impact of extended contexts on the performance of LLMs in tool utilization tasks, we conducted a pilot study on the ToolBench benchmark (Qin et al., 2024). Additional details can be found in Appendix A.1. We selected the samples with more than 3 steps performed by the ChatGPT-DFSDT method as a subset, called "*ToolBench-long*", to compare the performance difference with the full set. As demonstrated in Table 1, there is a significant decrease in the pass rate of existing frameworks when faced with tasks involving lengthy questions. This result supports the hypothesis that not only do extended contexts strain the models' computational efficiency, but they also hinder their ability to accurately interpret and respond to the given instructions. The pilot study highlighted a major issue with current tool utilization frameworks: the excessive context length adversely impacts the models' planning and execution capabilities when using tools.

Advantages of MAS: The pilot study highlights the need for a new approach beyond the traditional single-agent LLM model. MAS offers a promising solution by distributing tasks among specialized agents. This approach enhances memory efficiency, minimizes distractions, and allows for modular design and optimization of agents. Key benefits include: **(1) Memory Efficiency:** MAS manages memory better by assigning distinct segments of tasks to different agents, avoiding overload from processing lengthy contexts. **(2) Reduced Distractions:** Specialized agents focus on specific tasks, reducing interference from irrelevant information and improving overall performance. **(3) Modular Design:** The modular framework enables individual agent optimization, scalability, and adaptability to diverse tasks.

2.2 Tools Using and Planning in LLMs

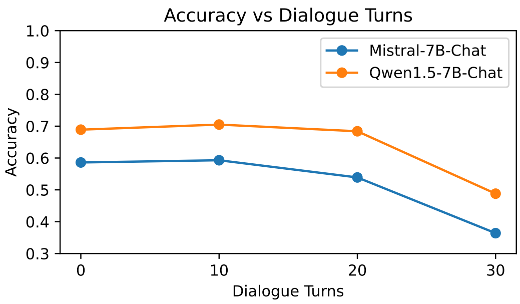

Figure 2: A pilot study explores the relationship between dialogue turns and the accuracy of tool selection for LLMs. It demonstrates that long-context instructions undermine the ability of LLMs to select the right tools.

Using external tools in LLMs is a well-established method to enhance model capabilities. However, as the number of available tools grows, managing multiple tools effectively becomes a complex planning challenge (Qin et al., 2024). This complexity often results in longer texts for tasks involving multiple tools. In the second pilot study, see details in Appendix A.2, we investigate the relationship between conversation turns and the model's tool selection accuracy. Figure 2 shows that as the number of dialogue rounds increases, the model's tool selection accuracy decreases linearly, underscoring the performance impact of longer texts in tool-involved tasks.

While numerous general MAS systems perform well, as shown in studies like (Du et al., 2023; Liang et al., 2023), there's a noticeable absence of multi-agent frameworks specifically designed for tool calling tasks. As highlighted in our two pilot studies, tool usage and planning tasks are particularly sensitive to context length and efficiency. Therefore, there's a pressing need for an optimized multi-agent framework tailored to tool calling scenarios.

3 Smurfs: A framework with multiple agents

***The Smurfs**, the beloved cartoon characters, symbolize unity and resourcefulness, and are good at using tools to overcome any challenge they encounter.*

3.1 Framework Overview

Figure 1 illustrates the entire workflow for the Smurfs framework. Initially, the **Planning Agent** identifies the user's complex request and breaks it down into manageable sub-tasks. **Executor Agents** are then tasked with collecting this specific information, utilizing access to external tools. **Answer Agent** compiles the findings into a cohesive response, which is subsequently verified by the **Verifier Agent** to ensure accuracy and relevance. This process exemplifies the framework's capability to efficiently handle complex queries by leveraging the specialized roles of multiple agents, thereby ensuring both the precision of task execution and the quality of the output. In the following sections, the functions of each agent will be detailed. More details about the memory of each agent can be seen at B

Planning Agent The primary responsibility of the Planning Agent is task decomposition. The strategy known as least-to-most prompting (Zhou et al., 2023) is highly effective in dissecting intricate problems into manageable sub-tasks and resolving them sequentially. This approach has demonstrated significant effectiveness and broad applicability. In scenarios involving complex reasoning tasks, we utilize the fundamental principles of the least-to-most strategy to break down intricate tasks into multiple sub-tasks, thereby improving management, efficiency and also the interpretability. An example illustrating how the Planning Agent employs this strategy to decompose a task is provided at Table 2, with the specific prompt available in Appendix C.

Executor Agent The Executor Agent is responsible for choosing and executing the tools to solve the sub-tasks. The agent has access to an external tool library (Qin et al., 2024). At each steps, the agent can invoke one tool to tackle the given task. As outlined in Algorithm 1, the agent, using the ReACT format (Yao et al., 2022) to choose the tool and arguments, then execute the tool. More detailed information of the Executor Agent can be

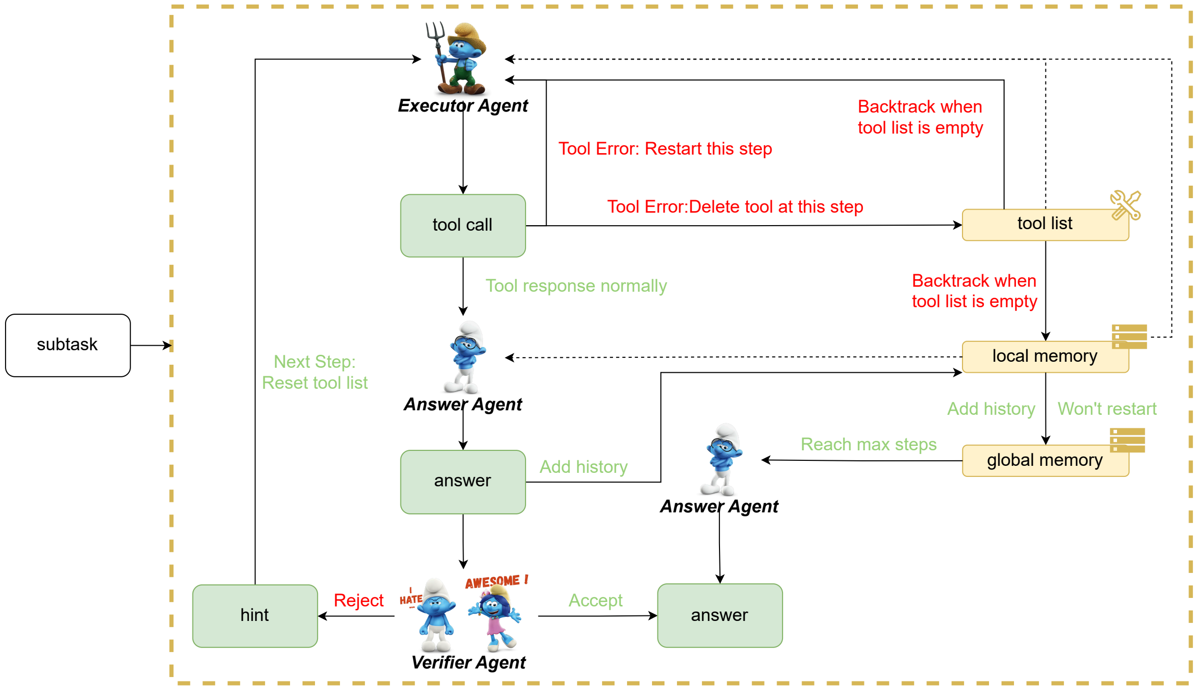

Figure 3: Details of the subtask-solving process of the Smurfs framework. The dotted line represents that the agent can see the memory and the full line stands for operation.

Task	I'm planning a trip to Turkey and need information about postal codes in Istanbul. Can you provide me with the postal code and district for Istanbul province with plate number 34? Additionally, I would like to know if there are any transit agencies available in Istanbul. Please fetch their names and contact numbers.
Decomposed Sub-tasks	1. Determine the postal code and district for Istanbul province with plate number 34. 2. Find out if there are any transit agencies in Istanbul. 3. Get the names of the transit agencies in Istanbul. 4. Obtain the contact numbers for the transit agencies in Istanbul.

Table 2: Example of task decomposition by Planning Agent.

found in B.2.

Algorithm 1: Tool Call

Input: A task q, a hint from the verifier agent h, problem solving history H and a set of available tools T.

Output: Tool response that is useful for solving the task.

thought $\leftarrow$ `gen_thought`(q, h, T);
tool $\leftarrow$ `choose_tool`$(q,$ thought$, T)$;
args $\leftarrow$ `gen_arguments`$(q,$ tool$, H,$ tool_doc$)$;
response $\leftarrow$ `call_tool`(tool, args);

Answer Agent To mitigate the performance degradation caused by lengthy contexts, we introduce the 'Answer Agent' role, designed to extract crucial content for each step and subtask. As demonstrated in the pilot study presented in Section 2.1, retaining all information may not always be beneficial, particularly in cases where the solution path is challenging to discern. Therefore, the primary role of the Answer Agent is to succinctly summarize the generated answers.

Verifier Agent Similar to current reasoning frameworks (Wei et al., 2022; Yao et al., 2022), sequential task reasoning may lead to high computational and tool resource waste within the framework (Qin et al., 2024). Therefore, inspired by (Qin et al., 2024), we employ a depth-first search with an early stopping strategy to find the solution path. The Verifier Agent serves as an early-stopping and reflection mechanism in the intelligent multi-agent ensemble, allowing for a balance between effectiveness and efficiency. This mechanism not only ensures the accuracy of the generated responses but also optimizes resource use by preventing unnecessary computations. Moreover, if the Verifier Agent thinks the answer at this step isn't accurate and reasonable enough, it will provide hints for the Executor Agent to get the missing information for the next steps. This dual role of the Verifier Agent enhances the overall performance of the framework, making it more robust and reliable for handling complex tasks.

3.2 Subtask Solving Process

After introducing the function of each agent, this section outlines how the agents collaborate to solve sub-tasks, as shown in Figure 3. Upon receiving a subtask or entering a new step, the tool list is refreshed, allowing the use of all available tools. The Executor Agent then calls the tools according to the task instruction. If a tool call fails, that tool is marked as unusable for this task at this step and the Executor Agent will try using other tools to solve the subtask at this step. This system also introduces a backtracking mechanism similar to DFSDT (Qin et al., 2024) to handle the situation where errors frequently occur. Once the executor agent gets a correct response from a tool, the Answer Agent will refine the information, filtering out irrelevant details such as lengthy web pages and generate the answer for the subtask at this step using the local memory. After this, the Verifier Agent checks the answer's accuracy. If it's incorrect, the process returns to the Executor Agent with the hint from the Verifier Agent for the next step; Otherwise, the final answer is provided. If the Executor Agent reaches its retry limit without success, Answer Agent will review the entire process from the global memory to produce an answer. More details of the subtask solving process can be seen at B

4 Experiments

To evaluate the effectiveness and efficiency of the Smurfs framework, we carried out a series of thorough experiments. In addition to the main experiment designed to assess the entire framework, we conducted an ablation study to test the capabilities of each component within the multi-agent framework. This section offers a detailed description of the experimental setup, methodologies employed, and key findings. Our goal is to showcase how the Smurfs framework, through the cooperative work of its multi-agent ensemble, effectively manages complex tasks while optimizing resource use. The experiments were designed with the following research objectives:

- To validate the capability of the entire framework in managing tool planning tasks;
- To independently assess the impact of each component on the overall performance, and identify the key factors influencing the multi-agent framework;

4.1 Evaluation

Our experiments are conducted on ToolBench (Qin et al., 2024), which encompasses multi-step tool usage tasks across over 16,000 APIs. To evaluate the planning and reasoning capabilities of the LLMs, we focused our experiments on **intra-category multi-tool instructions (I2)** and **intra-collection multi-tool instructions (I3)**. These instructions involve selecting 2-5 tools from the same category or collection and sampling up to 3 APIs from each tool to formulate the instructions. We employed two metrics for evaluation: **(1) Pass Rate** measures the percentage of instructions successfully executed within the allocated budget, evaluated by ChatGPT. **(2) Win Rate** represents the preference selection by a ChatGPT evaluator when presented with two solution paths. All other settings are kept consistent with those of the ToolBench benchmark.

4.2 Baselines

To investigate the varying impacts of the agent framework on models with different capabilities, we categorize our baseline into three groups. The first group consists of models that are fine-tuned based on the tool dataset, represented by ToolLLaMA (Qin et al., 2024). The second group encompasses untrained general language models such as Vicuna (Chiang et al., 2023), and Mistral-Instruct-7B (Jiang et al., 2023). The third category represents the closed-source model, embodied by GPT4. We subsequently contrast our approach with two agent frameworks, ReACT (Yao et al., 2022) and DFSDT (Qin et al., 2024), both of which are utilized for multi-step reasoning and model invocation. Notably, all methods employ the ground truth toolset for tool selection, thereby eliminating the influence of the tool retriever.

4.3 Main Experiments

Table 3 displays the results of the comprehensive evaluation of our proposed framework on ToolBench. For the untrained LLMs, it is clear that existing agent frameworks do not improve their performance in tool planning tasks; Vicuna, and Mistral-Instruct-7B all failed at the given tasks with the ReACT and DFSDT frameworks. However, Smurfs exhibits exceptional performance: Mistral combined with Smurfs achieves the highest score

Models	Method	I2-Inst. Pass	I2-Inst. Win	I2-Cat. Pass	I2-Cat. Win	I3-Inst. Pass	I3-Inst. Win	Average Pass	Average Win
ToolLLaMA-7B	ReACT	30.5	50.8	31.5	41.8	25.0	55.0	29.0	49.2
	DFSDT	77.0	68.5	77.0	58.0	66.0	69.0	73.3	65.2
Vicuna-7B	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Smurfs (ours)	77	73	70.5	64.25	78.0	87.0	75.2	74.8
Mistral-Instruct-7B	ReACT & DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	Smurfs (ours)	77.5	**80.0**	**79.5**	**79.2**	**79.0**	**94.0**	**78.7**	**84.4**
GPT4	ReACT	67.0	65.8	72.0	60.3	47.0	78.0	62.0	68.0
	DFSDT	**79.5**	73.3	77.5	63.3	71.0	84.0	76.0	73.5
	Smurfs (ours)	71.0	77.5	72.0	77.0	64.0	89.5	69.0	81.3

Table 3: ToolBench evaluation, with some results derived from (Qin et al., 2024; Yuan et al., 2024). The most effective approach is highlighted in bold, while the second is underlined.

among the baselines. Through its task decomposition mechanism, Smurfs transforms a long-context task into several simpler tasks, enabling the untrained model to effectively utilize external tools to manage complex tasks. Regarding the closed-source models, specifically GPT4 in these experiments, Smurfs also demonstrates competitive performance on the benchmark compared to other agent frameworks. A high win rate suggests that Smurfs is more adept at finding better solution paths than ChatGPT, a success that can likely be credited to the verifier agent.

4.4 Ablation Study

4.4.1 Importance of each component in MAS

	Average Pass	Average Win
Mistral	0.0	0.0
Mistral *with Smurfs*	79.3	86.6
w/o Verifier Agent	74.5$_{\downarrow 6.0\%}$	83.8$_{\downarrow 3.2\%}$
w/o Answer Agent	73.3$_{\downarrow 7.6\%}$	81.7$_{\downarrow 5.7\%}$
w/o Planning Agent	64.0$_{\downarrow 19.3\%}$	82.9$_{\downarrow 4.3\%}$

Table 4: Ablation study on ToolBench I2-Cat., I3-Inst. with Mistral-Instruct-7B.

We performed an ablation study to investigate the impact of each agent in our framework. We removed each agent individually, except for the indispensable Executor Agent, and compared the results to the complete framework. Table 4 shows that the Planning Agent is the most crucial component, followed by the Answer Agent and the Verifier Agent. **(1) Verifier Agent Removal:** Without verification, the framework uses a general depth-first search, leading to increased computational demand and more tool invocations. **(2) Answer Agent Removal:** Removing this agent means the Executor Agent's answer won't be summarized, risking the 'lost-in-the-middle' problem due to lengthy tool responses. **(3) Planning Agent Removal:** Without this agent, the global path searching strategy is affected. Models with Smurfs may show reduced performance without preliminary planning, as seen in current frameworks like ReACT and DFSDT.

4.4.2 Effect of number of agents

Models	# of Agents	I2-Cat. Pass	I2-Cat. Win	I3-Inst. Pass	I3-Inst. Win	Average Pass	Average Win
Vicuna	1	49.5	53.0	60.0	86.0	54.8	69.5
	2	**71.0**	66.0	75.0	83.0	73.0	74.5
	3	70.5	**64.3**	**78.0**	**87.0**	**74.3**	**75.6**
Mistral	1	65.5	70.5	56.0	94.0	60.8	82.3
	2	76.0	75.5	73.0	92.0	74.5	83.8
	3	**79.5**	**79.2**	**79.0**	**94.0**	**79.3**	**86.6**
GPT4	1	60.5	71.8	57.0	89.0	58.8	80.4
	2	69.5	**77.8**	59.0	**94.0**	64.3	**85.9**
	3	**72.0**	77.0	**64.0**	89.5	**68.0**	83.3

Table 5: The impact of MAS complexity on the performance of different models. More agents don't always bring more benefits.

In addition to evaluating the importance of each component, another relevant question arises: *Can a fixed agent system effectively accommodate every model?* Our hypothesis suggests that larger language models possess comprehensive intelligence, and employing complex agent systems may potentially hinder their performance compared to simpler ones. As shown in Table 5, we conducted an ablation study on Vicuna, Mistral-Instruct-7B,

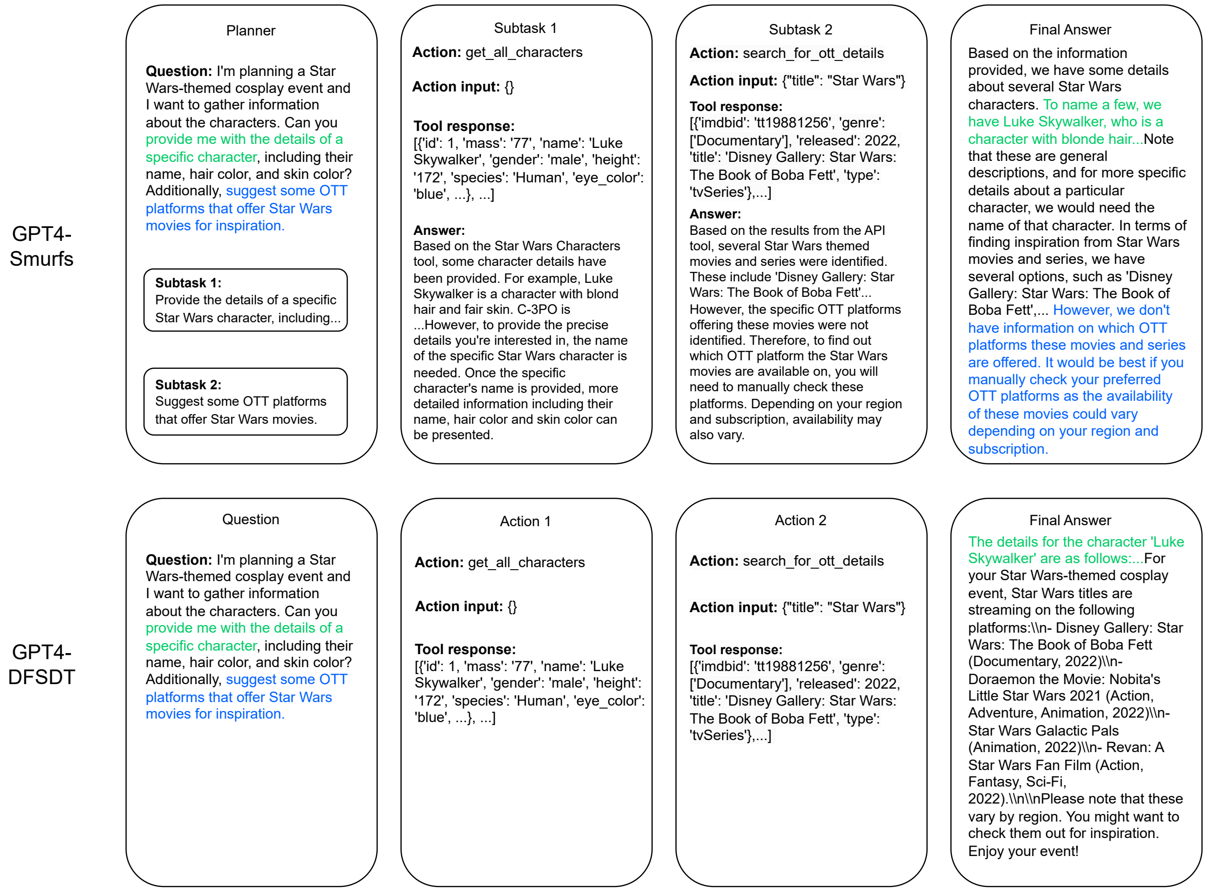

Figure 4: The illustration of how GPT4-Smurfs and GPT4-DFSDT solve long context problem. The two sub-questions and their corresponding answers are marked in two colors.

and GPT4, varying the number of agents [1]. The findings indicate that as the number of agents in a MAS increases, leading to increased complexity of the MAS, the system's performance enhancement does not scale linearly. In certain instances, GPT4 with a 2-agent MAS outperforms the one with a 3-agent MAS, suggesting that larger language models do not consistently benefit from more complex Multi-Agent Systems. Smaller language models can benefit more from the agent system, as seen with Vicuna increasing from 69.5 to 75.6.

4.5 Case Study

As shown in Figure 4, even though GPT4-DFSDT and GPT4-Smurfs use the same tool calls to solve the problem, GPT4-DFSDT only answers the first sub-question correctly while GPT4-Smurfs answers both sub-questions accurately. In the process of addressing the second sub-question, it is notable that the tool response only mentions titles of film and television products related to "Star Wars", without addressing OTT platforms. GPT-4-DFSDT erroneously interprets these titles as responses to the question, while GPT-4-Smurfs adeptly identifies this discrepancy and provides a more appropriate response. This case highlights that in situations where tool responses are lengthy and questions are complex, the single agent framework like DFSDT may be susceptible to distractions from irrelevant information, leading to erroneous answers. Conversely, the context-efficient Smurfs framework demonstrates a reduced susceptibility to irrelevant information, thereby generating more accurate answers.

5 Related Work

In this section, we review the literature related to multi-agent collaboration, tool-augmented language models, and complex task planning and reasoning. Each subsection provides an overview of

[1]Agents are incrementally added to the system based on their importance.

the topic, discusses recent advancements, identifies challenges, and suggests avenues for future research.

5.1 Multi-Agents Collaboration

Multi-agent systems have garnered significant attention due to their applicability in various domains such as robotics, economics, and computer networking. One of the key challenges in multi-agent systems is achieving effective collaboration among autonomous agents. Recent studies have focused on understanding collaborative behaviors and developing coordination strategies to enable agents to work together towards common goals. For example, research in reinforcement learning has explored techniques for emergent coordination, where agents learn to collaborate through interaction with the environment and other agents. (Du et al., 2023; Liang et al., 2023) carry out multi-agent interaction in the form of debate, which improves mathematical and strategic reasoning tasks. (Li et al., 2023; Wang et al., 2023) uses role-playing to conduct the interaction between multi-agents.

5.2 Tool-Augmented Language Models

Language models augmented with external knowledge sources, often termed as tool-augmented language models, have demonstrated promising outcomes in a variety of natural language processing tasks. These models utilize external knowledge bases, ontologies, or pre-trained models to enhance their contextual understanding and boost performance in tasks such as text generation, summarization, and question answering. Recent advancements in this field include techniques for integrating structured knowledge into language models, such as graph-based representations or semantic parsing. For instance, (Qin et al., 2024) introduces ToolBench, an instruction-tuning dataset for tool usage, along with a fine-tuned tool-oriented model, ToolLLaMA. Another significant contribution is Gorilla (Patil et al., 2023), which excels at writing API calls and also introduces a benchmark for evaluating LLMs with tools.

5.3 Complex task planning and reasoning

Problem decomposition ((Zhou et al., 2023; Drozdov et al., 2022; Khot et al., 2022; Press et al., 2022)) is a popular paradigm used in the LLM and leads to good performance in challenging reasoning tasks. This divide-and-conquer strategy provides sufficient explanation of how the model works. The current trend is that when model size reaches some limit, people start looking for a way to make the most of the limited model. Chain-of-thought prompting (Wei et al., 2022) is one of the strategy that lead models to divide the tasks into a chain, and solve the task step by steps. (Chen et al., 2022) propose a program of thoughts structure to enhance the reasoning ability. (Yao et al., 2024) simulate thought as tree structure to solve task. (Besta et al., 2024) using graph to simulation.

6 Conclusion

In this study, we present a novel MAS framework, ***Smurfs***, tailored to enhance the planning and reasoning capabilities of LLMs in handling complex tasks that involve lengthy contexts and tools. We conducted experiments on the multi-step tool usage benchmark, *ToolBench*, and the results demonstrated the overall effectiveness of the Smurfs framework compared to the baseline models.

Ablation studies are carried out to investigate and compare the significance of different components within the MAS framework. The findings revealed that preliminary planning was the most crucial element. Content summarizing also played a key role in mitigating the 'lost-in-the-middle' issue often encountered in long-context multi-step reasoning scenarios. While verification was not as influential on effectiveness, it proved valuable in enhancing computational efficiency by identifying the optimal solution path for complex task resolution. By dissecting and comparing the different aspects of the MAS framework, we aim to offer insights that could inspire advancements in the applicability and accuracy of LLMs.

In conclusion, this research contributes to the expanding field of study focused on enhancing LLM capabilities, particularly for multi-step tool usage tasks. It emphasizes the importance of task decomposition, preliminary planning, and efficient verification for improving task execution performance. We are confident that the knowledge gained from this study will lay the groundwork for the development of more sophisticated and efficient LLM frameworks in the future.

7 Limitations

Generalization Ability: While our empirical investigation has demonstrated promising results for the Mistral-7b-instruct model, additional benchmark evaluations may be needed to validate the generalization ability of the proposed ***Smurfs*** framework across various LLM architectures and tasks.

Model Size Constraints: Due to device limitations and computational constraints, our experiments primarily focused on the 7B models. Further evaluations with larger and smaller LLMs are required to assess the impact of the ***Smurfs*** framework on models of different sizes.

Computational Efficiency: Although preliminary findings suggest that the ***Smurfs*** framework can enhance computational efficiency through efficient verification, a more detailed analysis is needed to quantify the computational overhead introduced by the multi-agent architecture.

Acknowledging these limitations, future research should aim to address these gaps to provide a more comprehensive understanding of the ***Smurfs*** framework's capabilities and potential areas for improvement.

References

Stanley H Ambrose. 2001. Paleolithic technology and human evolution. *Science*, 291(5509):1748–1753.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Ali Dorri, Salil S Kanhere, and Raja Jurdak. 2018. Multi-agent systems: A survey. *Ieee Access*, 6:28573–28593.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. 2022. Compositional semantic parsing with large language models. In *The Eleventh International Conference on Learning Representations*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *Preprint*, arXiv:2305.14325.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for "mind" exploration of large language model society. *Preprint*, arXiv:2303.17760.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *Preprint*, arXiv:2305.19118.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*.

Kenneth Page Oakley and London British Museum. 1972. *Man the tool-maker*. 538. British Museum (Natural History) London.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. 2020. How context affects language models' factual predictions. *arXiv preprint arXiv:2005.04611*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. 2023. Tool learning with foundation models. *Preprint*, arXiv:2304.08354.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. 2023. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. *Preprint*, arXiv:2302.00093.

Wiebe Van der Hoek and Michael Wooldridge. 2008. Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 1(2):3.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. *Preprint*, arXiv:2205.10625.

A Pilot Study Settings

A.1 Pilot Study 1

We conducted the first pilot study using the ToolBench benchmark (Qin et al., 2024), which comprises over 16,000 APIs focused on tool usage and planning. We divided the benchmark into two subsets: "ToolBench-long," containing samples requiring more than three steps for ChatGPT-DFSDT, and "ToolBench-all," which includes all samples from ToolBench. On average, we consider the "ToolBench-long" subset to be more complex than the "ToolBench-all" set. We maintained all other settings consistent with the original source settings.

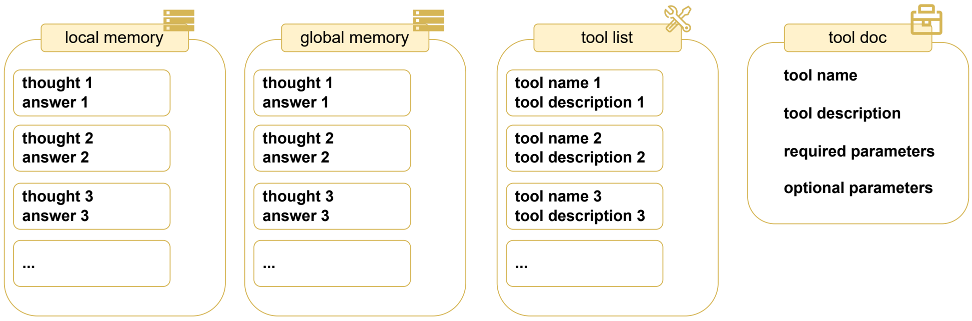

Figure 5: Demonstration of the memory of the Smurfs framework.

A.2 Pilot Study 2

The second pilot study is conducted using the BM-Tools (Qin et al., 2023) test datasets. We augmented each sample by adding random, irrelevant test samples as prefix prompts to increase their input length. We used tool selection accuracy as the metric to evaluate the LLMs' model selection capabilities. The experiment uses Mistral-7B-Instruct-v0.1 (Jiang et al., 2023) and Qwen1.5-7B-Chat (Bai et al., 2023) for evaluation.

B Details of the Smurfs

B.1 Memory management of Smurfs

When each agent performs a task, the prompt information they receive varies. We refer to this as the memory management system of Smurfs. As illustrated in Figure 5, information is primarily divided into four categories. The first category is local memory, consisting of the thought and answer from the previous steps. When backtracking happens, the thought and answer of the backtracking step will be popped out from the local memory. The second category is global memory, which stores all the previous steps history including those that have been backtracked. The third category is the tool list, similar to other tool-using frameworks, storing brief information about all accessible tools. Lastly, the tool document provides detailed usage information like the parameters infomation for each tool listed. As illustrated in 3, each agent has its own memory context. The Executor Agent's memory will be discussed in the next section. The Answer Agent takes in the subtask, the tool response and the local memory to generate the answers to the subtask using all the local memory so far. Then the thought and answer of this step will be added to the local memory and the global memory. The Verifier Agent takes in the subtask, the answer from the Answer Agent and return the status of the subtask. If the status is solved, then the system will return the subtask's answer and process to the next subtask; If the status is pending, the Verifier Agent will give hint to the Executor Agent to process to the next step. When the max steps have been reached for a subtask, the answer agent will generate the answer for the subtask using the global memory and process to the next subtask.

B.2 Executor Agent Workflow

Next, let's illustrate how memory is accessed during the Executor Agent's working process, as shown in Figure 6. When a subtask arises, the Executor Agent first receives the hint (task instruction) from the Verifier Agent, the subtask, the available tool list to generate the thought(a strategy for task execution) at this step. It then takes in the subtask, the thought and the tool list to generate an action with a specific action name. Subsequently, it takes in the subtask, the thought, the tool document of the chosen action and the local memory to generate the input of the chosen action.

B.3 Backtracking Mechanism

Smurfs uses a backtracking mechanism similar to DFSDT (Qin et al., 2024). As illustrated in 3, when tool responses error, the system will delete the tool from the tool list and retry this step. However, when all tools have reported error or the model thinks the available tools can't solve the subtask at this step, the system will pop out the latest local memory and the latest step, delete the tool used by the popped out step from the tool list of the current step and retry the the other solution path.

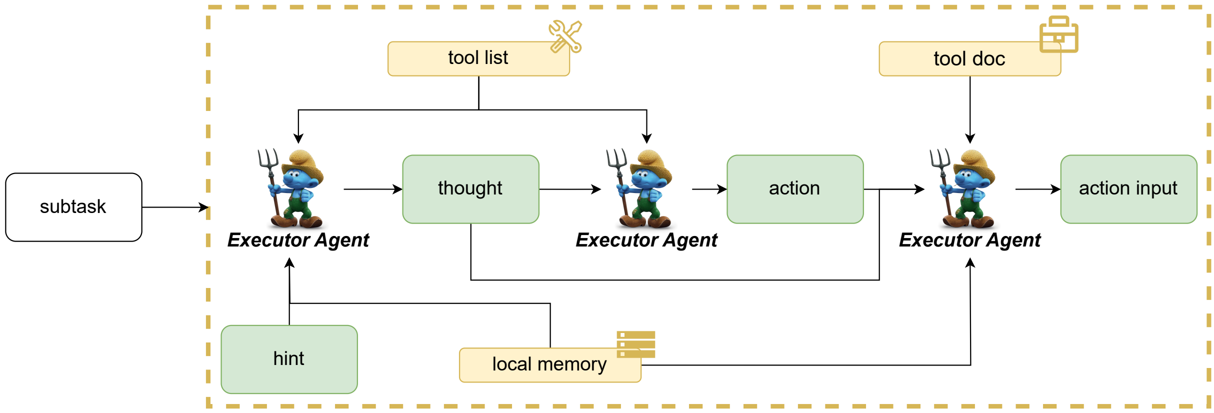

Figure 6: Details of the executor agent working process

C Prompts for multi-agent implementation

C.1 Task Decomposition

Figure 7 is an example of a task decomposition prompt.

Task Decomposition

Prompt:

You need to decompose a complex user's question into some simple sub-tasks and let the model execute it step by step. Please note that:

1. You should only decompose this complex user's question into some simple sub-tasks which can be executed easily by using a single tool.
2. Each simple subtask should be expressed into natural language.
3. Each subtask should contain the necessary information from the original question and should be complete, explicit and self-consistent.
4. You must ONLY output in a parsible JSON format. An example output looks like:

'''
{"Tasks": ["Task 1", "Task 2", ...]}
'''

This is the user's question: I'm planning a trip to Turkey and need information about postal codes in Istanbul. Can you provide me with the postal code and district for Istanbul province with plate number 34? Additionally, I would like to know if there are any transit agencies available in Istanbul. Please fetch their names and contact numbers.
Output: "Tasks": ["Find the postal codes and districts for plate number 34 in Istanbul.", "Search for transit agencies and their contact numbers in Istanbul."]

This is the user's question: I recently moved to a new address and I need to update my information. Can you retrieve my address details using the postal code 75094080? Additionally, I would like to know the companies that offer shipping services.
Output: {"Tasks": ["retrieve the address details using the postal code 75094080", "search for companies that offer shipping services to my address"]}

This is the user's question: I'm planning a trip to Turkey and need information about postal codes in Istanbul. Can you provide me with the postal code and district for Istanbul province with plate number 34? Additionally, I would like to know if there are any transit agencies available in Istanbul. Please fetch their names and contact numbers.
Output:

Expected Output:

{"Tasks": ["Determine the postal code and district for Istanbul province with plate number 34.", "Find out if there are any transit agencies in Istanbul.", "Get the names of the transit agencies in Istanbul.", "Obtain the contact numbers for the transit agencies in Istanbul."] }

Figure 7: An example prompt for task decomposition in the framework.

Tool Check

Prompt:

As a powerful language model, you're equipped to answer user's question with accumulated knowledge.

However, in some cases, you need to use external APIs to answer accurately.

Thus, you need to check whether the user's question requires you to call an external API to solve it.

Here are some tips to help you check:

1. If the user's question requires real-time information, since your knowledge base isn't updated in real-time, any such question will demand an API call.
2. If you need to obtain information (e.g., ID, name, phone number, geographical location, rank, etc.), you need to call the database APIs if you are not sure.
3. If the question demand a database search or internet research to generate an answer, this is another situation where an API call is necessary.

If need, please output 'YES'; If not, please output 'NO'

You need to give reasons first and then decide whether to keep it or not. You must only output in a parsible JSON format. Two example outputs look like:

Example 1: "Reason": "The reason why you think you do not need to call an external API to solve the user's question", "Choice": "No"

Example 2: "Reason": "The reason why you think you need to call an external API to solve the user's question", "Choice": "Yes"

This is the user's question: question: Determine the postal code and district for Istanbul province with plate number 34.

Output:

Expected Output:

{"Reason": "To determine the postal code and district for a specific location based on a plate number, we would typically need to access a combination of databases, including vehicle registration databases and postal code databases. Since we do not have direct access to these databases, we will need to call external APIs to retrieve this information. Therefore, the user's question requires an API call.", "Choice": "Yes"}

Figure 8: An example prompt for tool check in the framework.