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 seamlessly transforming a conventional LLM into a synergistic multi-agent ensemble, Smurfs can enhance the model's ability to solve complex tasks at no additional cost. This is achieved through innovative prompting strategies that allocate distinct roles within the model, thereby facilitating collaboration among specialized agents and forming an intelligent multi-agent system. Our empirical investigation on both open-ended task of StableToolBench and closed-ended task on HotpotQA showcases Smurfs' superior capability in intricate tool utilization scenarios. Notably, Smurfs outmatches all the baseline methods in both experiments, setting new state-of-the-art performance. 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 encounter several challenges when using multiple tools to solve tasks. These challenges include effective solution planning and 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 plug-and-play 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.

	Pass Rate ↑ (%)	Win Rate ↑ (%)	# of Tokens per request ↓	# of Tokens per query ↓
ReACT	$44.4_{\pm 1.1}$	*base*	1,424	6,479
DFSDT	$55.4_{\pm 2.0}$	60.4	1,743	20,714
Smurfs (ours)	$57.4_{\pm 1.1}$	62.4	459	8,096

Table 1: Comparison of token cost and performance between tool planning methods over StableToolBench. Existing methods, *ReACT* and *DFSDT*, have limitations due to high token costs or poor performance. The results are averaged over the subtasks within StableToolBench.

To this end, we introduce '***Smurfs***' an innovative MAS framework inspired by the collaborative and versatile nature of its namesake cartoon characters. The proposed framework is based on the principle: *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. The effectiveness of Smurfs is validated through both open-ended

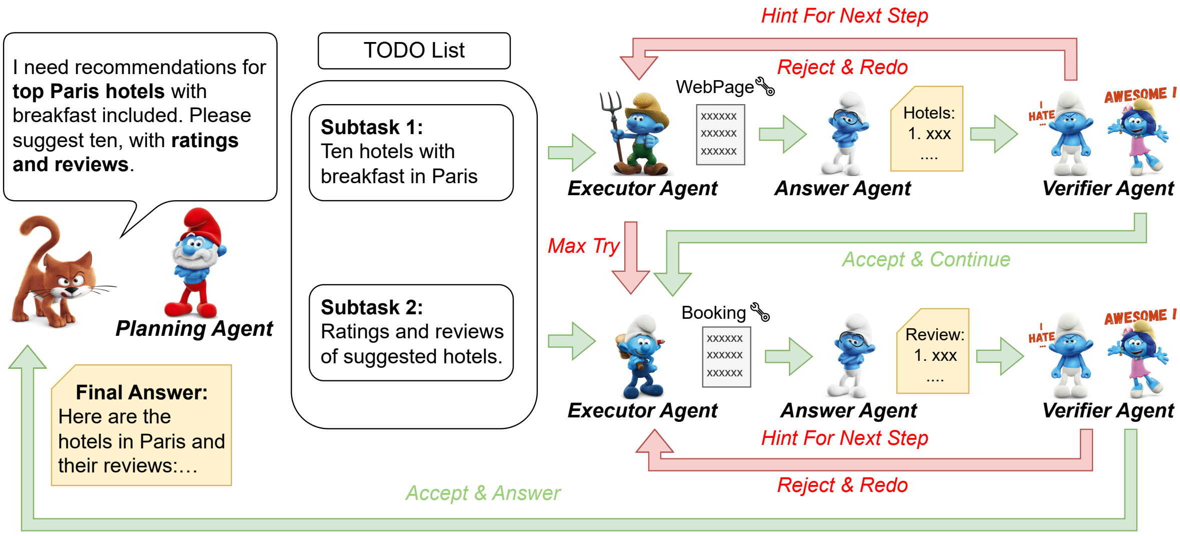


Figure 1: Demonstration of the whole process of the Smurfs framework.

and closed-ended tool planning benchmark experiments (Guo et al., 2024; Yang et al., 2018), where the proposed MAS system consistently outperform baseline methods on both benchmarks. An ablation study followed by a case study further investigates the underlying reasons for this effectiveness. These results not only establish a new state-of-the-art in the field but also offer concrete evidence of the multi-agent approach's efficacy in enhancing LLM capabilities.

The contributions of this paper can be summarized as follows:

1. We introduce a novel plug-and-play MAS framework to enhance the tool planning capabilities of LLMs. Experiments demonstrate the effectiveness of this approach, which is also more cost-efficient compared to existing tool planning methods.

2. Ablation studies further reveal the underlying reasons for the effectiveness of the MAS framework, providing valuable insights for future research.

2 Motivation

2.1 Multi-Tool Planning

To augment LLMs to do multi-tool planning for solving complex problems, previous work has seen numerous attempts. Chain-of-Thought (Wei et al., 2023) was the first to propose the method of thought and answer chain reasoning. ReACT (Yao et al., 2022) further introduced the thought-action-observation format for tool chain reasoning, leading to the development of various multi-tool planning methods (Chen et al., 2023a; Xu et al., 2023; Shinn et al., 2023). The latest work, DFSDT (Qin et al., 2024), was proposed to address the inherent limitations of CoT and ReACT: error propagation and limited exploration. Deep First Search Decision Tree, denoted as DFSDT, is powerful in addressing multi-tool planning problems. Its core concept involves employing a depth-first search (DFS) approach for multi-tool planning (for more details, see Appendix A). When a tool fails or is deemed inadequate for solving the current problem, DFSDT backtracks to the previous solution state and attempts to resolve the issue using a different tool. However, several limitations were identified with the mechanism of DFSDT: (1) **instability of the rollback mechanism**, (2) **redundant context**, and (3) **premature termination**. The following sections will introduce these limitations in detail.

2.1.1 Instability of the Rollback Mechanism

The rollback mechanism in DFSDT is determined by the model. The number of steps to roll back and the selection of new tools after rollback are guided using prompt containing the errors encountered in the previous failed trajectory. When the model is sufficiently robust, this rollback mechanism serves as a highly flexible and efficient planning strategy. However, when the model's capability is insufficient, it will fail to execute the correct rollback mechanism, i.e. retry the same error tools or roll back too far.

2.1.2 Redundant Context

In the process of planning with DFSDT, each tool plan is generated using the entire conversation history (including all the thoughts, actions, action inputs and tool responses) as context. However, in reality, each step of tool planning only requires a very small portion of the relevant history for effective planning.

The context redundancy not only increases computational overhead but also reduces the accuracy of model inference due to the inclusion of irrelevant historical data. As highlighted by (Liu et al., 2024), redundant context 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), which is known as the 'lost-in-the-middle' problem.

2.1.3 Premature Termination

The termination mechanism set by DFSDT involves adding a termination tool to the model's selectable toolkit. When the model selects this termination tool, DFSDT stops and provides an answer. However, in practical applications, this mechanism often prematurely terminates when dealing with complex problems requiring multi-step reasoning. We hypothesize that this issue arises due to the redundant interference of other tool information and history information, which disrupts the model's ability to judge whether the original problem should be terminated. Instead, the model focuses on whether the current sub-problem requires termination, leading the mechanism to terminate after resolving the sub-problem.

2.2 Multi Agent System

To address the limitations inherent in DFSDT and to further enhance LLM's multi-tool planning capabilities, multi-agent system (MAS) has emerged as a natural solution. Inspired by human social division of labor and cooperation, MAS aim to enable AI agents to accomplish more complex tasks more effectively and efficiently through the division of labor and collaboration. Previous works (Song et al., 2023; Liu et al., 2023; Chen et al., 2023b; Yin et al., 2024; Qiao et al., 2024) has leveraged MAS to achieve this goal. Table 2 shows the difference between them. Based on those works, we further design the MAS named Smurfs to address issues with DFSDT. By dividing tasks among different agents, each agent can focus on a specific part of the DFSDT task, accessing only the necessary history as context during task execution, which effectively addresses the issue of **redundant context**. The redesign of the rollback mechanism to incorporate memory and tool list rollback mechanisms addresses the **instability of the rollback mechanism**. Drawing on the concept of least-to-most prompting (Zhou et al., 2023), the original problem is first decomposed into sub-problems for macro-level planning. Subsequently, DFSDT is used to solve each sub-problem at the micro-level, with macro-level planning guiding the micro-level planning, thereby resolving the issue of **premature termination**.

Method	**Multi-Agent**	**Training**	**Generality**	**Reflection**	**Planning**
REACT (Yao et al., 2022)	✗	✗	✓	✗	Iterative
Reflexion (Shinn et al., 2023)	✗	✗	✓	✓	Iterative
Chameleon (Lu et al., 2023)	✗	✗	✓	✗	Global
HuggingGPT (Shen et al., 2023)	✗	✗	✓	✗	Global
BOLAA (Liu et al., 2023)	✓	✗	✓	✗	Iterative
AgentVerse (Chen et al., 2023b)	✓	✗	✓	✗	Iterative
FIREACT (Chen et al., 2023a)	✗	✓	✗	✓	Iterative
DFSDT (Qin et al., 2024)	✗	✓	✗	✗	Iterative
RESTGPT (Song et al., 2023)	✓	✗	✓	✗	Iterative
Lumos (Yin et al., 2024)	✓	✓	✗	✗	Iterative or Global
AutoAct (Qiao et al., 2024)	✓	✓	✗	✓	Iterative
Smurfs (Ours)	✓	✗	✓	✓	Iterative and Global

Table 2: Comparison of related works.

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 task 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. Each agent focus on its own task and only use the relevant part of the conversation history to reduce the **Redundant Context**. This process exempli-

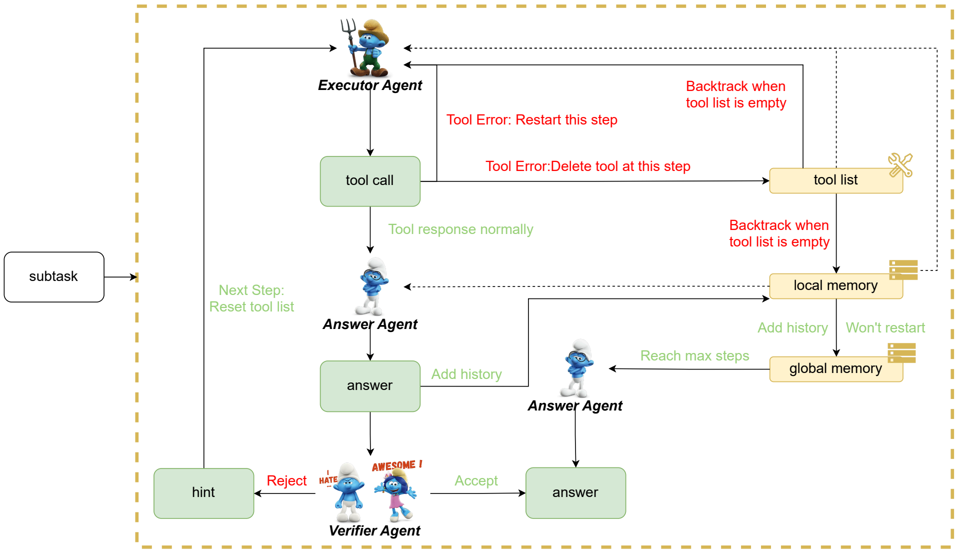


Figure 2: Details of the subtask-solving process of the Smurfs framework. The dotted line represents that the agent can see the memory and the full line stands for operation.

fies 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 system mechanism and functions of each agent will be detailed. More details of memory system can be seen at Appendix B.

3.2 Agent Components

Tools The tool documents about the tools that Smurfs can utilize in the completion of a complex task are denoted as $D = \{n_i, d_i, p_i\}_{i=1}^{|d|}$, where n represents the tool name, d represents tool usage description, p represents parameter description and $|d|$ represents the amount of the available tools. The available tool list is denoted as $\tau = \{n_i, d_i\}_{i=1}^{|\tau|}$. τ_t denotes the tool list Smurfs can utilize at time t.

Memory The memory of the agent system at time t is the history of the task-solving process before t, denoted as $M = (m_1, m_2, ..., m_{t-1})$ and $m_i = (\gamma_i, a_i)$, where m_i represents memory element at time i and γ_i, a_i represents thought and answer generated by the system at time i. There are two types of memory in Smurfs: local memory and global memory. the local memory is used to record the ongoing solution trajectory and to generate the next action in the current trajectory. The global memory, meanwhile, records all trajectories and is used to generate the sub-problem's answer by combining all trajectory records when the maximum number of retries is exceeded. This local-global combined memory system ensures that the planning of the current solution trajectory is not influenced by the context of erroneous trajectories. It also generates an answer that combines all trajectories when the verifier agent cannot determine task completion within the maximum number of planned steps. This memory system ensures context efficiency during the task-solving process.

3.3 Macro Planning

Planning Agent The primary responsibility of the Planning Agent is doing macro-level planning through task decomposition to prevent **premature termination**. The inference process of the Planning Agent is:

$$Plan\ P : (p1, p2, ...) = PA(q) \tag{1}$$

Where p_i represents sub-problem of the original query q, PA represents the Planning Agent. After the task decomposition, the agent system will use Executor Agent, Answer Agent an Verifier Agent to solve each sub-problem using DFSDT collaboratively in a sequential order. To utilize the answer of the previous sub-problem when solving subsequent sub-problem, the strategy known as least-to-most prompting (Zhou et al., 2023) is used.

3.4 Subtask Solving Process

After introducing the function of plan agent, this section outlines how the agents collaborate to solve sub-tasks, as shown in Figure 2.

Stable Rollback To address the **instability of the rollback mechanism** in DFSDT, we propose a rollback mechanism based on rules. Whenever an error occurs while using a tool $\tau_{t,i}$ at time t, the tool list at t τ_t will pop $\tau_{t,i}$ out and reperform tool selection and tool planning (ensuring that the faulty tool is not selected again). If, at time t, the tool list becomes empty, it signifies that after the system choosing tool $\tau_{t-1,j}$ at time t-1, no subsequent trajectory can solve the problem. In this case, the agent system will roll back to time t-1, meaning that the local memory M will pop out the memory element m_{t-1} at time t-1, and the tool list at time t-1 τ_{t-1} will pop out tool $\tau_{t-1,j}$. The agent system will then set the time t=t-1 and continue planning. This rule-based rollback mechanism, compared to the original model-based rollback mechanism of DFSDT, is less flexible and might reduce rollback efficiency. However, it is more stable, ensuring the correctness of deep first search and enabling models with weaker capabilities to utilize DFSDT on tool planning.

Executor Agent The Executor Agent is responsible for choosing and executing the tools to solve the sub-tasks. At each time t, the agent can invoke one tool to tackle the given task:

$$\gamma = EA.gen\_thought(p, M, \tau, h) \tag{2}$$

$$\alpha = EA.choose\_tool(p, \gamma, \tau) \tag{3}$$

$$\beta = EA.gen\_arguments(p, M, D[\alpha]) \tag{4}$$

$$r = EA.call\_tool(\alpha, \beta) \tag{5}$$

Where p is the sub-problem from Planning Agent, h is the hint from the Verifier Agent, τ is the tool list, M is local memory, $D[\alpha]$ means the tool document of tool α. The agent, using the ReACT format (Yao et al., 2022) to choose the tool and arguments, then execute the tool. Noticed that each inference process only uses the relevant part from the local memory and tool list to reduce the context redundancy. More detailed information of the Executor Agent can be found in Figure 6.

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 sub-problem:

$$Answer : a = AA(q, r, M) \tag{6}$$

Where q is sub-problem from the Planning Agent, r is response from the Executor Agent, M is the local memory (or global memory if max retry reaches). As the 'lost-in-the-middle' theory described 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 and tool responses to maintain the memory efficiency.

Verifier Agent The Verifier Agent serves as an early-stopping and reflection mechanism, allowing for a balance between effectiveness and efficiency

$$h, c = VA(q, a) \tag{7}$$

Where q denotes the sub-problems from the Planning Agent, a denotes the answer from the answer agent, h and c denotes hint and check status respectively. If check status generated is 0, that means the Verifier Agent thinks the sub-problem isn't completed, the system will add the thought and answer of this time to the local and global memory, set t=t+1 and continue the inference procedure.If check status is 1, the sub-problems is thought to be solved and the system will start to deal with the next sub-problem.

4 Experiments

To evaluate both the effectiveness and efficiency of the Smurfs framework, in thie section, we carried out two multi-tool planning tasks: (1) an open-ended task, *StableToolBench* (Guo et al., 2024), and (2) a closed-ended task, *HotpotQA* (Yang et al., 2018). In addition to these main experiments designed to assess the entire framework, we conducted an ablation studies followed by a case study to test the capabilities of each component within the multi-agent framework and investigate the underlying reasons for its effectiveness.

4.1 Open-ended Task: StableToolBench

StableToolBench is a tool learning benchmark derived from ToolBench (Qin et al., 2024), encompassing multi-step tool usage tasks across over 16,000 APIs. The benchmark employs two metrics for evaluation: **(1) Pass Rate** measures the percentage of instructions successfully executed within the allocated budget. **(2) Win Rate** represents the preference selection by a ChatGPT evaluator when presented with two solution paths.

Baselines Following the original paper that introduced the benchmark, we adopt *ReACT* (*CoT*) (Wei

Backbone	Method	StableToolBench													
		I1-Inst.		I1-Cat.		I1-Tool.		I2-Cat.		I2-Inst.		I3-Inst.		Average	
		Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win	Pass	Win
GPT-3.5 Turbo	ReACT	$41.6_{\pm 1.2}$	/	$48.4_{\pm 0.5}$	/	$52.5_{\pm 0.5}$	/	$52.2_{\pm 1.0}$	/	$31.6_{\pm 1.2}$	/	$39.9_{\pm 2.0}$	/	$44.4_{\pm 1.1}$	/
GPT-3.5 Turbo	DFSDT	$54.1_{\pm 1.0}$	64.4	$60.1_{\pm 0.0}$	61.4	$59.9_{\pm 1.7}$	53.8	$60.9_{\pm 0.9}$	62.9	$52.8_{\pm 3.7}$	66.0	$44.3_{\pm 4.8}$	54.1	$55.4_{\pm 2.0}$	60.4
GPT-3.5 Turbo	Smurfs	$\underline{60.3_{\pm 1.5}}$	65.0	$67.0_{\pm 1.0}$	$\underline{69.9}$	$60.3_{\pm 1.3}$	54.4	$54.3_{\pm 0.4}$	$\underline{63.7}$	$42.6_{\pm 1.6}$	64.2	$60.1_{\pm 1.0}$	57.4	$57.4_{\pm 1.1}$	62.4
Mistral-7B	ReACT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mistral-7B	DFSDT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mistral-7B	Smurfs	$\mathbf{76.3_{\pm 0.8}}$	63.8	$\mathbf{86.7_{\pm 1.2}}$	62.7	$\mathbf{81.0_{\pm 1.9}}$	58.2	$\mathbf{70.4_{\pm 2.7}}$	54.0	$\mathbf{63.8_{\pm 2.4}}$	$\underline{67.0}$	$\mathbf{85.2_{\pm 0.7}}$	57.4	$\mathbf{77.2_{\pm 1.6}}$	60.5
GPT-4 Turbo	ReACT	$41.1_{\pm 1.5}$	60.1	$53.2_{\pm 1.3}$	62.1	$42.2_{\pm 1.1}$	48.1	$50.0_{\pm 0.7}$	57.3	$38.7_{\pm 0.8}$	65.1	$37.7_{\pm 1.3}$	47.5	$43.8_{\pm 1.1}$	56.7
GPT-4 Turbo	DFSDT	$52.7_{\pm 1.4}$	$\underline{69.9}$	$58.2_{\pm 0.9}$	66.0	$59.7_{\pm 1.2}$	$\underline{58.2}$	$59.3_{\pm 0.7}$	62.1	$52.2_{\pm 2.3}$	**67.9**	$61.5_{\pm 1.8}$	$\underline{65.6}$	$57.3_{\pm 1.4}$	$\underline{65.0}$
GPT-4 Turbo	Smurfs	$59.3_{\pm 1.4}$	**71.2**	$\underline{73.3_{\pm 1.3}}$	**72.5**	$\underline{67.4_{\pm 0.7}}$	**69.6**	$\underline{66.7_{\pm 1.9}}$	**73.4**	$\underline{55.5_{\pm 1.4}}$	66.0	$\underline{70.5_{\pm 0.0}}$	**72.1**	$\underline{65.5_{\pm 1.1}}$	**70.8**

Table 3: The open-end tool planning task evaluation on the StableToolBench benchmark (Guo et al., 2024). The most effective approach is highlighted in bold, while the second-best is underlined. Win rate is calculated by comparing each model with ChatGPT-ReACT. A win rate higher than 50% means the model performs better than ChatGPT-ReACT.

et al., 2023) and *DFSDT* (Touvron et al., 2023) as baseline methods for comparison. Additionally, we include the backbones used in the paper: gpt-3.5-turbo-0613 (GPT-3.5 Turbo) (OpenAI) and gpt-4-turbo-preview (GPT-4 Turbo). To explore the adaptability of the tool-planning methods, we also include Mistral-7B-Instruct-v0.2 (Mistral-7B) (Jiang et al., 2023) as one of the selected backbones in our experiments.

Settings To minimize the influence of varying tool APIs on experimental results, we conducted all experiments using the same API cache (Guo et al., 2024). For a fair comparison among the candidate methods and to reduce variability, each model was executed once and evaluated three times, with results averaged. Other settings follow those specified in the original benchmark paper.

Results Table 3 displays the results on StableToolBench. For the untrained LLM, Mistral-7B, existing agent frameworks did not improve its performance in tool planning tasks; Mistral-7B failed these tasks when integrated with the ReACT and DFSDT frameworks [1]. However, Smurfs exhibited exceptional performance: when combined with Mistral-7B, Smurfs achieved competitive scores among the baselines. Through its task decomposition mechanism, Smurfs transforms long-context tasks into simpler ones, enabling the untrained model to effectively utilize external tools for managing complex tasks. Regarding closed-source models, specifically GPT4 in these experiments, Smurfs also demonstrated outstanding performance on the benchmark compared to other agent frameworks and achieved state-of-the-art results on the benchmark. Its high success rate suggests that Smurfs is more effective at finding optimal solution paths compared to ChatGPT.

Further Analysis We conducted a detailed analysis of the token costs associated with each tool planning method for the tasks, a critical evaluation aspect for multihop reasoning tasks. As shown in Table 1 (detailed in Appendix E), the average token costs per question and API request are evaluated for ReACT, DFSDT, and Smurfs on StableToolBench. The analysis reveals that DFSDT generally requires about 20,000 tokens per question, encompassing both prompt and completion tokens. This is nearly three times the token cost compared to ReACT and twice as much as Smurfs. Despite this higher token cost, DFSDT does not demonstrate commensurate effectiveness improvements over other methods. These findings underscore the cost-efficiency of the proposed MAS framework, Smurfs, which not only reduces token expenditure in solving multihop planning tasks but also delivers outstanding performance in evaluations.

4.2 Closed-ended Task: HotpotQA

Compared to open-ended tasks, closed-ended tasks provide a more stable and robust evaluation. To this end, we evaluate the methods on HotpotQA (Yang et al., 2018) in addition to StableToolBench. HotpotQA is a multi-hop QA task that is challenging due to the requirement for rich background knowledge, with answers typically being short entities or yes/no responses.

Baselines The compared baselines include **CoT** (Wei et al., 2023), **REACT**(Yao et al., 2022), **Chameleon**(Lu et al., 2023), **Reflexion** (Shinn et al., 2023), **BOLAA** (Liu et al., 2023), **ReWOO** (Xu et al., 2023), **FIREACT** (Chen et al., 2023a), **AutoAct**(Qiao et al., 2024).

---

[1] Experiment results show that Mistral-7B failed to correctly execute the 'finish' action during inference, resulting in invalid responses.

Backbone	Method 👤 Single-Agent 👥 Multi-Agent	HotpotQA			
		Easy	Medium	Hard	All
GPT-3.5 Turbo	○ 👤 CoT	48.21	44.52	34.22	42.32
	○ 👤 Zero-Shot Plan	50.71	45.17	38.23	44.70
Mistral-7B Instruct-v0.2	○ 👤 CoT	33.70	22.38	22.14	26.07
	○ 👤 ReAct	38.09	27.57	22.05	29.24
	○ 👤 Chameleon	37.07	26.67	19.20	27.65
	○ 👤 Reflexion	40.78	35.02	28.36	34.72
	○ 👥 BOLAA	40.86	32.11	22.36	31.78
	○ 👥 ReWOO	38.42	31.89	25.98	32.10
	○ 👥 Smurfs (ours)	45.94	**40.74**	30.72	**39.13**
	● 👤 FireAct	45.52	32.02	30.17	35.90
	● 👥 AUTOACT	**48.69**	36.65	**31.37**	38.89
Llama-2 13B-chat	○ 👤 CoT	37.90	25.28	21.64	28.27
	○ 👤 ReAct	28.68	22.15	21.69	24.17
	○ 👤 Chameleon	40.01	25.39	22.82	29.41
	○ 👤 Reflexion	44.43	37.50	28.17	36.70
	○ 👥 BOLAA	33.23	25.46	25.23	27.97
	○ 👥 ReWOO	30.09	24.01	21.13	25.08
	○ 👥 Smurfs (ours)	42.62	27.21	22.92	30.92
	● 👤 FireAct	45.83	38.94	26.06	36.94
	● 👥 AUTOACT	**47.29**	**41.27**	**32.92**	**40.49**
Llama-2 70B-chat	○ 👤 CoT	45.37	36.33	32.27	37.99
	○ 👤 ReAct	39.70	37.19	33.62	36.83
	○ 👤 Chameleon	46.86	38.79	34.43	40.03
	○ 👤 Reflexion	48.01	46.35	35.64	43.33
	○ 👥 BOLAA	46.44	37.29	33.49	39.07
	○ 👥 ReWOO	42.00	39.58	35.32	38.96
	○ 👥 Smurfs (ours)	52.86	**50.77**	**44.87**	**49.50**
	● 👤 FireAct	50.82	41.43	35.86	42.70
	● 👥 AUTOACT	**56.94**	50.12	38.35	48.47

Table 4: The closed-end tool planning evaluation on HotpotQA (Yang et al., 2018), with some results derived from (Qiao et al., 2024). The most effective approach for each group is highlighted in bold, while the second-best is underlined. Methods marked with ● require model training.

Settings and Metrics Following the settings in (Qiao et al., 2024), we use open-source Llama-2 models (Touvron et al., 2023) and Mistral-7B (Jiang et al., 2023) as the backbones of each agent to evaluate the performance of Smurfs. The evaluation metrics is `reward` $\in [0, 1]$, defined as the F1 score grading between the prediction and ground-truth answer. For more details about the experiment, see Appendix C.

Results Smurfs, as an untrained MAS system, not only comprehensively outperforms untrained agents but also achieves and even surpasses the accuracy of trained agents across most backbone models. This sufficiently demonstrates that the mechanism of smurfs ensures strong generalization capabilities while maintaining high effectiveness.

Observations indicate that the performance of LLama-2-13b-chat on smurfs-related tasks is suboptimal, likely due to its limited capabilities in tool arguments generation. Specifically, the primary issue identified is that, when the Executor agent successfully selects relevant tool, it tends to produce hallucination arguments that can't be used by the tools. This indicates that LLama-2-13b-chat may need further training for usage of tools. The experimental results may substantiate this viewpoint, demonstrating that the untrained methods of llama-2-13b-chat generally exhibit significantly lower accuracy compared to the trained methods. Nevertheless, Smurfs achieves the second highest accuracy among the untrained methods, only slightly behind reflexion, which still attests to Smurfs' capability.

	I3-Inst.	
	Pass (%)	Win (%)
GPT-3.5 Turbo *with Smurfs*	$60.1_{\pm1.0}$	57.4
w/o Answer Agent	$57.4_{\pm2.9}$	49.2
w/o Verifier Agent	$54.1_{\pm2.7}$	42.6
w/o Planning Agent	$35.5_{\pm3.3}$	42.6
GPT-4 Turbo *with Smurfs*	$70.5_{\pm1.0}$	72.1
w/o Answer Agent	$82.2_{\pm2.5}$	72.1
w/o Verifier Agent	$79.2_{\pm0.8}$	63.9
w/o Planning Agent	$71.9_{\pm2.8}$	63.9

Table 5: Ablation study on StableToolBench I3-Inst subset to investigate the importance of each component within the framework.

5 Ablation Study

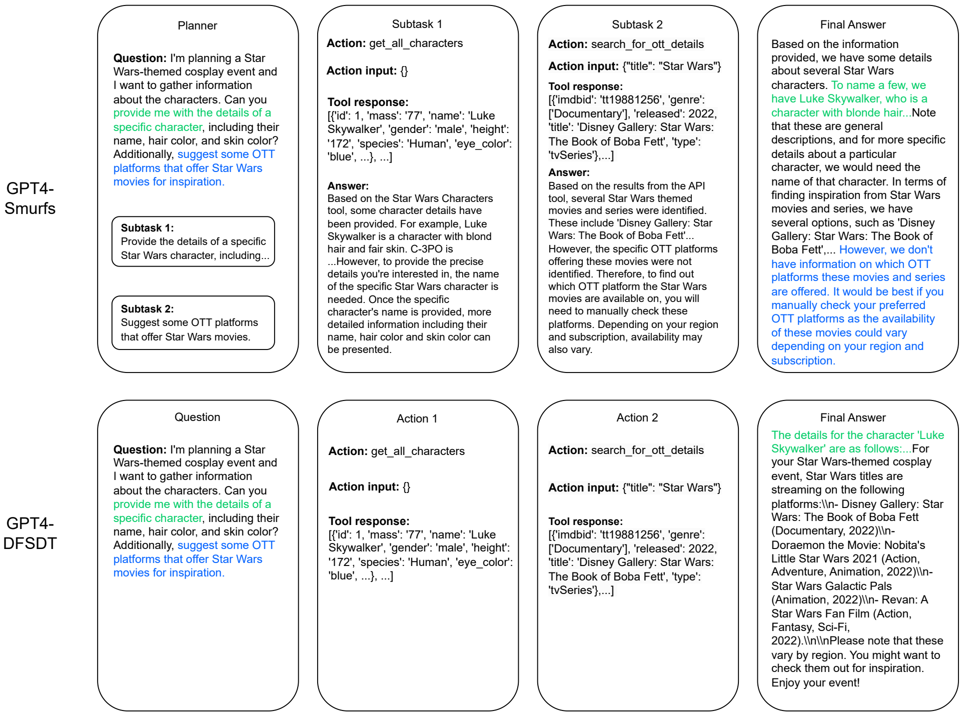
5.1 Importance of each component in MAS

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 5 shows that the Planning Agent is the most crucial component, followed by the Verifier Agent, with the Answer Agent being the least important.

(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 answers won't be summarized, risking the 'lost-in-the-middle' problem due to lengthy tool responses. As shown in the results, a more intelligent model, GPT-4 Turbo, can mitigate the negative impact of the Answer Agent's removal. We believe this is because the more powerful model can leverage more information effectively.

(3) Planning Agent Removal: Removing the Planning Agent affects the global path-searching strategy. Models with Smurfs may show reduced performance without preliminary planning, as seen in current frameworks like ReACT and DFSDT.



Question: I'm planning a Star Wars-themed cosplay event and I want to gather information about the characters. Can you provide me with the details of a specific character, including their name, hair color, and skin color? Additionally, suggest some OTT platforms that offer Star Wars movies for inspiration.

Subtask 1: Provide the details of a specific Star Wars character, including...

Subtask 2: Suggest some OTT platforms that offer Star Wars movies.

Action: get_all_characters

Action input: {}

Tool response: [{'id': 1, 'mass': '77', 'name': 'Luke Skywalker', 'gender': 'male', 'height': '172', 'species': 'Human', 'eye_color': 'blue', ...}, ...]

Answer: Based on the Star Wars Characters tool, some character details have been provided. For example, Luke Skywalker is a character with blond hair and fair skin. C-3PO is ...However, to provide the precise details you're interested in, the name of the specific Star Wars character is needed. Once the specific character's name is provided, more detailed information including their name, hair color and skin color can be presented.

Action: search_for_ott_details

Action input: {"title": "Star Wars"}

Tool response: [{'imdbid': 'tt19881256', 'genre': ['Documentary'], 'released': 2022, 'title': 'Disney Gallery: Star Wars: The Book of Boba Fett', 'type': 'tvSeries'},...]

Answer: Based on the results from the API tool, several Star Wars themed movies and series were identified. These include 'Disney Gallery: Star Wars: The Book of Boba Fett'... However, the specific OTT platforms offering these movies were not identified. Therefore, to find out which OTT platform the Star Wars movies are available on, you will need to manually check these platforms. Depending on your region and subscription, availability may also vary.

Based on the information provided, we have some details about several Star Wars characters. To name a few, we have Luke Skywalker, who is a character with blonde hair...Note that these are general descriptions, and for more specific details about a particular character, we would need the name of that character. In terms of finding inspiration from Star Wars movies and series, we have several options, such as 'Disney Gallery: Star Wars: The Book of Boba Fett',... However, we don't have information on which OTT platforms these movies and series are offered. It would be best if you manually check your preferred OTT platforms as the availability of these movies could vary depending on your region and subscription.

Question: I'm planning a Star Wars-themed cosplay event and I want to gather information about the characters. Can you provide me with the details of a specific character, including their name, hair color, and skin color? Additionally, suggest some OTT platforms that offer Star Wars movies for inspiration.

Action: get_all_characters

Action input: {}

Tool response: [{'id': 1, 'mass': '77', 'name': 'Luke Skywalker', 'gender': 'male', 'height': '172', 'species': 'Human', 'eye_color': 'blue', ...}, ...]

Action: search_for_ott_details

Action input: {"title": "Star Wars"}

Tool response: [{'imdbid': 'tt19881256', 'genre': ['Documentary'], 'released': 2022, 'title': 'Disney Gallery: Star Wars: The Book of Boba Fett', 'type': 'tvSeries'},...]

The details for the character 'Luke Skywalker' are as follows:...For your Star Wars-themed cosplay event, Star Wars titles are streaming on the following platforms:\\n- Disney Gallery: Star Wars: The Book of Boba Fett (Documentary, 2022)\\n- Doraemon the Movie: Nobita's Little Star Wars 2021 (Action, Adventure, Animation, 2022)\\n- Star Wars Galactic Pals (Animation, 2022)\\n- Revan: A Star Wars Fan Film (Action, Fantasy, Sci-Fi, 2022).\\n\\nPlease note that these vary by region. You might want to check them out for inspiration. Enjoy your event!

Figure 3: 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.

The results demonstrate that the impact of removing the Planning Agent is significant, as it directly influences the multihop reasoning ability of the MAS.

5.2 Case Study

As shown in Figure 3, although 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.

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 conduct experiments on the multi-step tool usage benchmark, *StableToolBench* and *HotpotQA*, and the results demonstrate the overall effectiveness and efficiency of the Smurfs framework compared to baseline methods.

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. For future work, we believe incorporating more dedicated and specific roles within the system may further enhance effectiveness and efficiency, based on the 'Smurfs principle': *synergistic collaboration among specialized agents can overcome the limitations faced by individual LLMs.*

7 Limitations

Model Size Constraints: Due to computational constraints, our experiments did not include larger and more diverse types of LLMs.

Agent Component Scale-Up: Although we selected the most common and intuitive agent roles for the proposed MAS, there are many possibilities for researchers to explore. Investigating more well-designed agent roles may help improve the effectiveness of the agent system, and developing automated methods to identify these roles could facilitate effective scaling.

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.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023a. Fireact: Toward language agent fine-tuning. *Preprint*, arXiv:2310.05915.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. *Preprint*, arXiv:2308.10848.

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.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *Preprint*, arXiv:2403.07714.

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*.

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.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2023. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *Preprint*, arXiv:2308.05960.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *Preprint*, arXiv:2304.09842.

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.

OpenAI. ChatGPT. `https://openai.com/blog/chatgpt`.

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*.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. Autoact: Automatic agent learning from scratch for qa via self-planning. *Preprint*, arXiv:2401.05268.

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.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Preprint*, arXiv:2303.17580.

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.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world restful apis. *Preprint*, arXiv:2306.06624.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *Preprint*, arXiv:2305.18323.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

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*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent lumos: Unified and modular training for open-source language agents. *Preprint*, arXiv:2311.05657.

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 Details of DFSDT

See Figure 5.

B Details of the Smurfs

See Figure 6 for executor working process and Figure 4 for memory and tool library of Smurfs.

C Experiment Settings for Hotpot QA

Following settings in (Qiao et al., 2024), which is randomly select 300 dev questions divided into three levels for evaluation, with 100 questions in each level. For tool library that can be used in HotpotQA see Table 6

D Prompts for multi-agent implementation

Prompts used by each agent and their example outputs are shown in Figure 7 to 13.

E Token Cost on StableToolBench Evaluation

We analyzed the token cost for the StableToolBench experiments. As shown in Table 7, the total token cost for each subtask within the StableToolBench is compared across three candidate tool-planning methods. The data demonstrates that, across all tasks from easy to hard, DFSDT consistently incurs high token costs, while the other two methods maintain relatively low token costs. This verifies the context-efficiency of the proposed method.

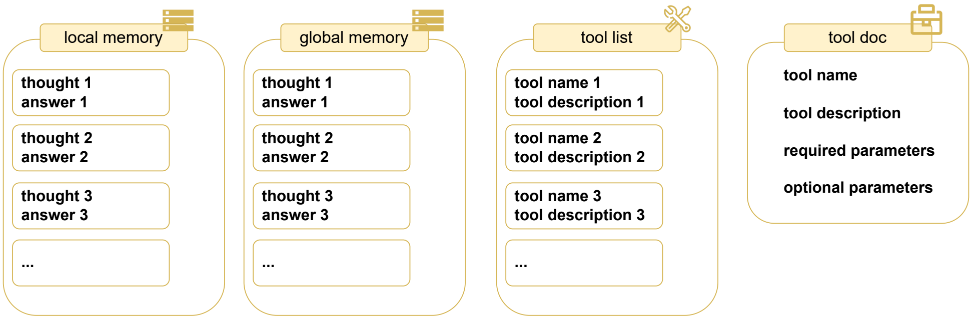
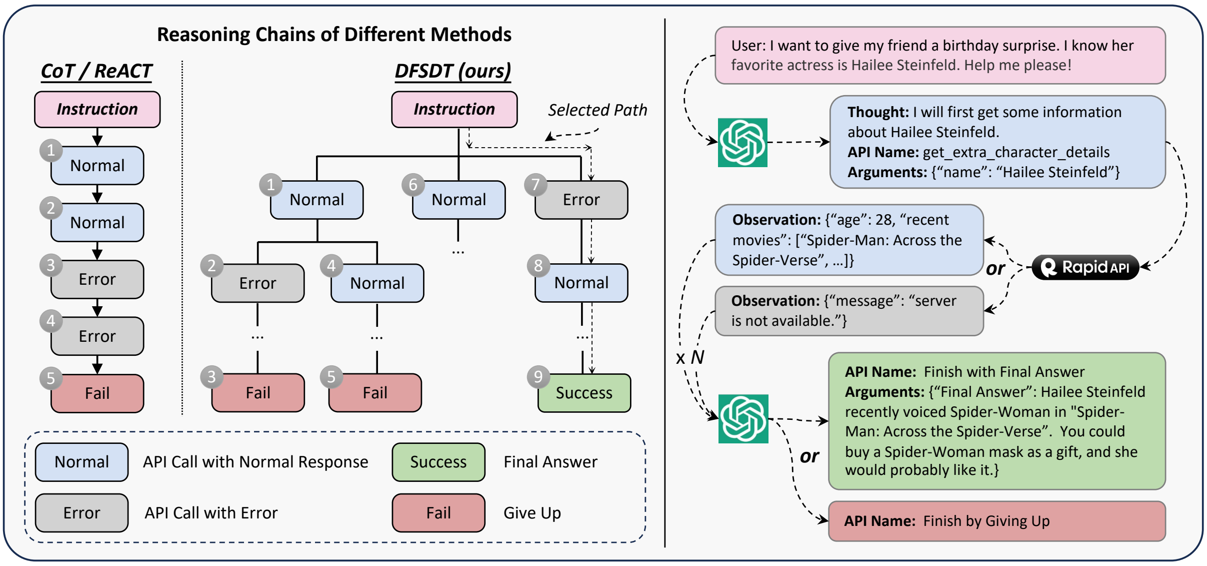


Figure 4: Demonstration of the memory of the Smurfs framework.



User: I want to give my friend a birthday surprise. I know her favorite actress is Hailee Steinfeld. Help me please!

Thought: I will first get some information about Hailee Steinfeld.
API Name: get_extra_character_details
Arguments: {"name": "Hailee Steinfeld"}

Observation: {"age": 28, "recent movies": ["Spider-Man: Across the Spider-Verse", ...]}

Observation: {"message": "server is not available."}

API Name: Finish with Final Answer
Arguments: {"Final Answer": Hailee Steinfeld recently voiced Spider-Woman in "Spider-Man: Across the Spider-Verse". You could buy a Spider-Woman mask as a gift, and she would probably like it.}

API Name: Finish by Giving Up

Figure 5: A comparison of our and conventional CoT or ReACT during model reasoning (left) (Qin et al., 2024). We show part of the solution path annotation process using (right).

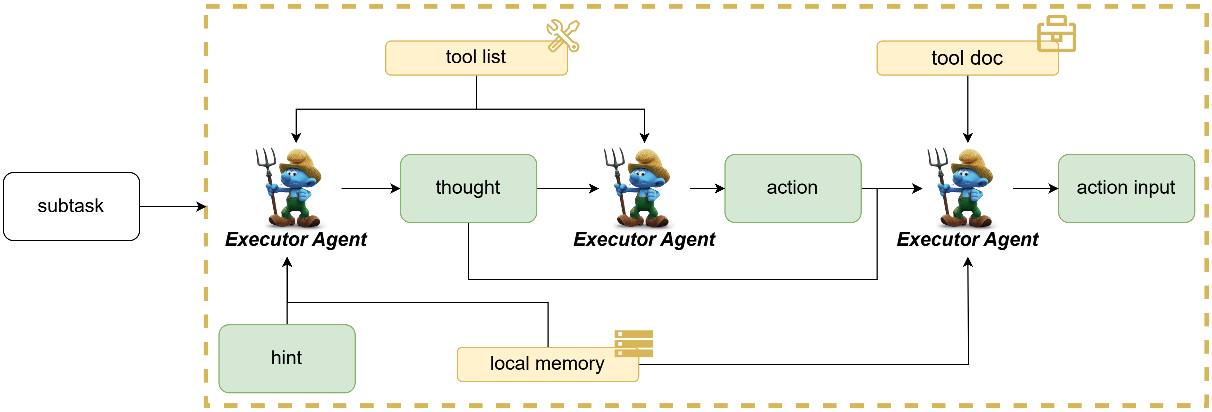


Figure 6: Details of the executor agent working process

Name	**Definition**	**Usage**
BingSearch	BingSearch engine can search for rich knowledge on the internet based on keywords, which can compensate for knowledge fallacy and knowledge outdated.	BingSearch[query], which searches the exact detailed query on the Internet and returns the relevant information to the query. Be specific and precise with your query to increase the chances of getting relevant results. For example, Bingsearch[popular dog breeds in the United States]
Retrieve	Retrieve additional background knowledge crucial for tackling complex problems. It is especially beneficial for specialized domains like science and mathematics, providing context for the task	Retrieve[entity], which retrieves the exact entity on Wikipedia and returns the first paragraph if it exists. If not, it will return some similar entities to retrieve. For example, Retrieve[Milhouse]
Lookup	A Lookup Tool returns the next sentence containing the target string in the page from the search tool, simulating Ctrl+F functionality on the browser.	Lookup[keyword], which returns the next sentence containing the keyword in the last passage successfully found by Retrieve or BingSearch. For example, Lookup[river].

Table 6: Tool library for HotpotQA.

Backbone	**Method**	**I1-Inst.**		**I1-Cat.**		**I1-Tool.**		**StableToolBench I2-Cat.**		**I2-Inst.**		**I3-Inst.**		**Average**	
		Total	Avg.	Total	Avg.	Total	Avg.	Total	Avg.	Total	Avg.	Total	Avg.	Total	Avg.
GPT-3.5 Turbo	ReACT	1,010,304	6,198	824,676	5,390	1,010,514	6,396	900,855	7,265	824,510	7,778	461,121	7,559	838,663	6,764
GPT-3.5 Turbo	DFSDT	3,303,062	20,264	2,745,667	17,945	3,152,532	19,953	2,560,297	20,648	3,098,365	29,230	1,390,787	22,800	2,708,452	21,807
GPT-3.5 Turbo	Smurfs	1,090,404	7,127	1,917,348	11,763	1,464,535	9,269	957,088	7,638	1,096,162	10,341	632,084	10,362	1,191,270	9,417

Table 7: Token costs for various candidate tool-planning methods on the StableToolBench benchmark (Guo et al., 2024). ‘Total’ indicates the total number of tokens used to complete each subtask, including both prompt and completion tokens. ‘Avg.’ represents the average number of tokens used per question within the subtasks. Higher token counts imply greater costs for solving the same task.

Planning Agent

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: {question}

Output:

Example 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}

Output:

Example 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.

Executor Agent: Thought

Prompt:

You need to analyse the previous execution history and generate your internal reasoning and thoughts on the task, and how you plan to solve it based on the current attempts.

Do not output thought that is too long. Output in 2-3 sentences is OK.

This is the user's task:
{task}

This is the Tool List:
{functions}

This is the previous execution history:
{messages}

This is the hint comes from the evaluator:
{hint}

Output:

Example Output:

Looking at the task, the user wants to find the postal codes and districts for the number plate 34 in Istanbul. I have a tool that provides Turkish plates, and since Istanbul is in Turkey, this tool might provide the needed information. I will proceed to use the 'Logistics:Turkey Postal Codes:il' tool to accomplish the task.

Figure 9: An example prompt for tool check in the framework.

Executor Agent: Action

Prompt:
This is the user’s question:
{question}
These are the tools you can select to solve the question:
Tool List:
{Tool_list}

Please note that:
1. You should only chooce one tool from the Tool List to solve this question.
2. You must ONLY output the ID of the tool and your reason for choosing it in a parsible JSON format. An example output looks like:
’’’
Example: {"ID": ID of the tool, "Reason": The reason for choosing the tool}
’’’

Output:

Example Output:
{’ID’: ’1’, ’Reason’: ’This tool provides information about Turkish plates and postal codes, which can be applied to the given task of finding the postal codes and districts for the number plate 34 in Istanbul.’}

Figure 10: An example prompt for action generation in the framework.

Executor Agent: Action Input

Prompt:
Given a user’s question and a API tool documentation, you need to output parameters according to the API tool documentation to successfully call the API to solve the user’s question.
Please note that:
1. The Example in the API tool documentation can help you better understand the use of the API.
2. Ensure the parameters you output are correct. The output must contain the required parameters, and can contain the optional parameters based on the question. If no paremters in the required parameters and optional parameters, just leave it as {}
3. If the user’s question mentions other APIs, you should ONLY consider the API tool documentation I give and do not consider other APIs.
4. The question may have dependencies on answers of other questions, so we will provide logs of previous questions and answers for your reference.
5. You must ONLY output in a parsible JSON Format. The example output looks like:
’’’
Example: {"keyword": "Artificial Intelligence", "language": "English"}
’’’

There are logs of previous questions and answers:
previous_log

This is the current user’s question: question

This is API tool documentation: api_dic

Output:

Example Output:
{"il": 34}

Figure 11: An example prompt for action input generation in the framework.

Answer Agent

Prompt:

You should answer the question based on the response output by the API tool.
Please note that:
1. Answer the question in natural language based on the API response reasonably and effectively.
2. The user cannot directly get API response, so you need to make full use of the response and give the information in the response that can satisfy the user's question in as much detail as possible.
3. Do not output answer that is too long. Output in 3-6 sentences is OK.

This is the user's question:
{question}
This is the API response:
{call_result}

Output:

Example Output:

The postal codes and districts for the plate number 34 in Istanbul include: Adalar district with postal code 34975 for Burgazada area, 34970 for Büyükada area, 34973 for Heybeliada area and 34977 for Kınalıada area. There is also the Arnavutköy district with postal code 34275 for areas like Anadolu, Arnavutköy Merkez, İmrahor, İslambey, Mustafa Kemal Paşa, Nenehatun, and Yavuz Selim. Another area in Arnavutköy district is Baklalı with postal code 34277.

Figure 12: An example prompt for Answer Agent in the framework.

Verifier Agent

Prompt:

An agent is trying to solve the query proposed by the user.
You need to evaluate whether the given query has been completed reasonably and accurately. If so, summarize the solution to the user. If not, summarize the current progress, and propose what is missing.

You response contains following elements:
Speak: (your words to the agent if the task is pending, or a complete answer based on the full execution log to the user if the task is finished)
Status: (0 or 1. 0 for pending and 1 for finished)

You must only output in a parsible JSON format. Two example outputs look like:
Example 1: {"Speak": "answer based on the full execution log to the user", "Status": "1"}
Example 2: {"Speak": "your words to the group if the task is pending", "Status": "0"}

This is the answer from the previous execution result:
{answer}

This is the original question: {question}

Output:

Example Output:

{"Speak": "The postal codes and districts for plate number 34 in Istanbul include: Adalar district with postal code 34975 for Burgazada area, 34970 for Büyükada area, 34973 for Heybeliada area and 34977 for Kınalıada area. In addition, Arnavutköy district has postal code 34275 for areas such as Anadolu, Arnavutköy Merkez, İmrahor, İslambey, Mustafa Kemal Paşa, Nenehatun, and Yavuz Selim. Another part of Arnavutköy district, Baklalı, has the postal code 34277.", "Status": "1"}

Figure 13: An example prompt for Verifier Agent in the framework.