Multi-Mission Tool Bench: Assessing the Robustness of LLM based Agents through Related and Dynamic Missions

Anonymous ACL submission

Abstract

Large language models (LLMs) demonstrate strong potential as agents for tool invocation due to their advanced comprehension and planning capabilities. Users increasingly rely on LLM-based agents to solve complex missions through iterative interactions. However, existing benchmarks predominantly access agents in single-mission scenarios, failing to capture real-world complexity. To bridge this gap, we propose the Multi-Mission Tool Bench. In the benchmark, each test case comprises multiple interrelated missions. This design requires agents to dynamically adapt to evolving demands. Moreover, the proposed benchmark explores all possible mission-switching patterns within a fixed mission number. Specifically, we propose a multi-agent data generation framework to construct the benchmark. We also propose a novel method to evaluate the accuracy and efficiency of agent decisions with dynamic decision trees. Experiments on diverse open-source and closed-source LLMs reveal critical factors influencing agent robustness and provide actionable insights to the tool invocation society. This benchmark is available in XXX.

1 Introduction

In recent years, large language models (LLMs) have achieved significant progress in natural language processing. These models demonstrate strong capabilities to understand contextual information and user instructions, making them effective agents for mission completion.

Real-world applications require agents to handle dynamic user demands. As users frequently adjust their requests during conversations (Figure 1), agents must complete sequential missions with evolving requirements. This situation challenges the robustness of an agent's decision-making. However, existing benchmarks focus primarily on single-mission scenarios.

This paper presents the Multi-Mission Tool Bench. This benchmark evaluates agent robustness in related and dynamic multi-mission scenarios. The benchmark addresses three core challenges: 1) it contains more mission-types than others, i.e. four major categories and six subcategories; 2) it includes all mission-type transition patterns in prefixed mission number; 3) all successive missions have strong relations with prior dialogues, agents are forced to extract information from previous missions. Therefore, it closely mirrors the complexity of real-world.

To simulate all mission-type switching patterns, we first define the mission-types by their corresponding agent action-types. Agent actions are divided into four main types: using a single tool, using multiple tools, chatting with users, and using tools after clarifying parameters. An agent accomplishes a single mission by performing one of these actions. Therefore, we define four types of missions. For sequential missions, agents combine multiple action-types to reach the objectives. Figure 2 a) displays that the agent employs the combination of four action-types to complete the four sequential missions in Figure 1. Thus, we introduce the mission switching space to describe the transformations of mission types. Figure 2 b) shows that our benchmark thoroughly explores the proposed space with a prefixed mission number. This indicates that our benchmark includes all mission-type transition patterns. In contrast, other benchmarks have a more limited range of action diversity.

To construct the multi-mission benchmark, we propose a controllable data generation framework with multiple characters. The framework simulates the mission execution process through dialogic interactions among five agents: user, planner, tool, AI, and checker. In each generation process, we assign the desirable mission type and mission relationship to guide the framework. Ultimately, our benchmark encompasses all potential combinations

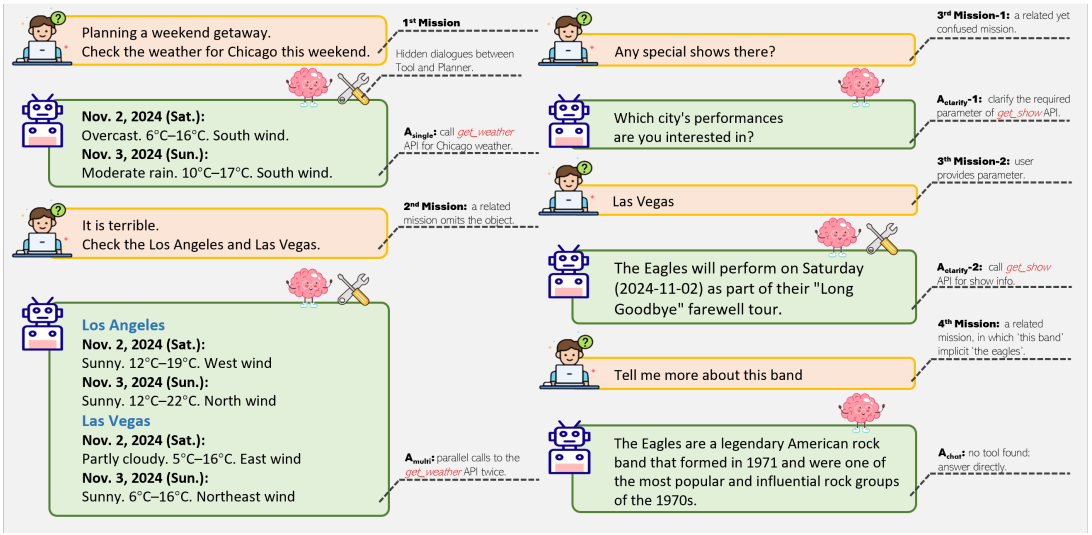

Figure 1: A multi-mission example. It contains four **related** missions, and the mission types are changing **dynamically**. This figure presents the conversation between a user and an AI. The inter-dialogues are hided.

in the mission switching space for a set number of missions. Notably, a complete mission involves multiple rounds of dialogues.

To evaluate the proposed benchmark, we introduce a novel evaluation method. It assesses the accuracy and efficiency of agents decisions, by employing dynamic decision trees.

Eventually, we evaluate a range of open-source and closed-source LLMs, encompassing both specific and general LLMs. Our comprehensive experiments reveal numerous factors influencing the robustness of agent decision-making. These findings offer valuable insights for guiding future research on the development of LLM-agents.

The main contributions of this paper are:

- To the best of our knowledge, this is the first benchmark that assesses agent robustness in related and dynamic multi-mission scenarios.
- We introduce a controllable multi-role data generation framework to explore the action-type space in multi-mission contexts.
- A novel testing method is proposed to evaluate the accuracy and efficiency of dynamic path planning.
- Comprehensive testing of open-source and closed-source LLMs is conducted, revealing various factors that affect the robustness of agent decision making.

Section 4 explains how we build the benchmark. It covers how to create related missions, predefine mission-types, and explore the mission switching space. Section 5 describes the evaluation methods we use for this benchmark. Section 6 shows the test results of LLMs and presents our analysis of these findings.

2 Related Work

2.1 Evaluation of LLMs

Recent benchmarks evaluate the capabilities of LLM-based agents from various point of views. Some research evaluates the generalizability of agents in various scenarios (Li et al., 2024; Trivedi et al., 2024; Liu et al., 2024c). Others(Du et al., 2024; Qin et al., 2024; Ye et al., 2024; Li et al., 2023) collected massive tools to investigate the impact of tool diversity on agent performance. Certain research (Zhuang et al., 2023; Guo et al., 2024; Xie et al., 2024) examines agents within specific domains. While some works (Shen et al., 2024b; Chen et al., 2024; Huang et al., 2024a) provide a comprehensive assessment of multiple agent abilities, others (Huang et al., 2024b; Tang et al., 2023; Qiao et al., 2024a) address specific issues like the illusion problem (Patil et al., 2023) and multistep execution capabilities (Shen et al., 2024a; Yao et al., 2024).

Our benchmark assesses agents' overall capabilities, emphasizing challenges of related and dynamic multi-missions. Importantly, the multistep tasks discussed in previous studies align with our approach of employing multiple tools to complete a single mission.

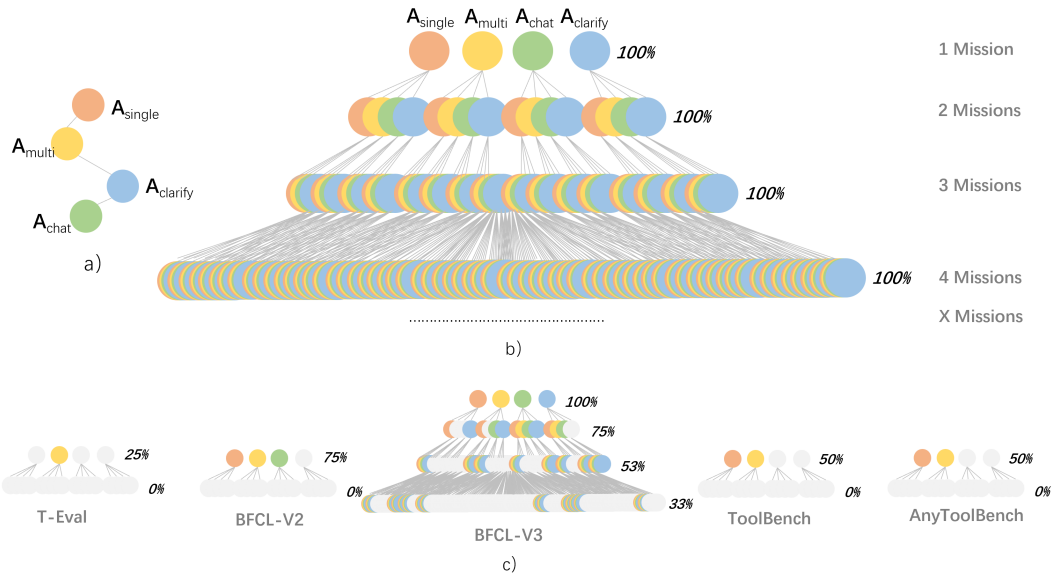

Figure 2: Visualization of mission switching space. a) Four distinct colors represents four different action-types. The green dot indicates the agent sequentially selects four type of actions to execute four missions. b) The distribution of the proposed benchmark within the mission switching space. Each row corresponds to a different number of missions. Each dot indicates a specific combination of the current and preceding action-types. Colored dots indicate combinations included in the benchmark, while gray dots indicate their absence. c) Distribution of four other agent benchmarks in the space.

The work most similar to ours is BFCL V3 (Charlie Cheng-Jie Ji, a). It also involves four types of agent actions and various user missions in one test case. However, BFCL V3 only covers a small part of the mission switching space. In contrast, our work simulates all possible mission transitions within a predefined set of missions. In most test data of BFCL V3, missions have no information dependencies. Agents can complete any given mission autonomously without relying on information from previous dialogues. In our case, all data contain related missions.

Other studies, WorfBench and TaskBench (Qiao et al., 2024a; Shen et al., 2024b), also introduce a graph-based evaluation method for multi-tool invocation. However, they only compute the similarity between the agent's planned path and the annotation through graph matching, unable to explicitly determine its correctness or calculate the optimal probability of the agent's plan, as our work does.

Table 1 compares the mentioned benchmarks with our proposed one in various aspects.

2.2 LLM-as-Agent

User mission automation is a significant research area for large LLMs. General (Achiam et al., 2023; Sun et al., 2024; Yang et al., 2024; Team et al., 2024; GLM et al., 2024; Srivastava and Dames, 2024) LLMs with larger scale primarily integrate it within multi-task learning process. While there are also many smaller specialized LLMs based agents.

We categorize agent research into various approaches. Some studies (Xu et al., 2024; Qiao et al., 2024b; Zhang et al., 2024b) equip agents with self-reflection and self-correction capabilities to improve their understanding of environmental feedback. Others (Zhang et al., 2024a; Han et al., 2024; Islam et al., 2024) introduce heuristic decision frameworks to solve complex problems. Further research (Shi et al., 2024; Schick et al., 2023; Liu et al., 2024b) focuses on strengthening agents' core skills. Concurrently, some work (meetkai; Lin et al., 2024; Liu et al., 2024b) generate more diverse training data with proposed frameworks. Our study also introduces a novel data generation framework. Unlike previous works, our framework uniquely specifies desired agent action-types.

The proposed benchmark simulates real-world application scenarios, and evaluates the core abilities of agents and tests various general LLMs and specialized agent LLMs.

Benchmark	MutMiss*	Rate of RelMiss†	$MSSS_4^‡$	Mission-Types					
				A_{single}	A_{chat}	$A_{clarity}$	A^S_{multi}	A^P_{multi}	A^{S+P}_{multi}
Ours	✓	**100**	**100**	✓	✓	✓	✓	✓	✓
BFCL v3(Charlie Cheng-Jie Ji, a)	✓	**15.7**	**39.7**	✓	✓	✓	✗	✓	✗
BFCL v1(Patil et al., 2023)	✗	0.0	0.9	✓	✓	✗	✗	✓	✗
BFCL v2(Charlie Cheng-Jie Ji, b)	✗	0.0	0.9	✓	✓	✗	✗	✓	✗
ToolBench(Qin et al., 2024)	✗	0.0	0.0	✓	✗	✗	✓	✗	✗
AnyToolBench(Du et al., 2024)	✗	0.0	0.0	✓	✗	✗	✓	✗	✗
τ-bench(Yao et al., 2024)	✗	0.0	0.0	✓	✗	✗	✓	✗	✗
T-EVAL(Chen et al., 2024)	✗	0.0	0.0	✓	✗	✗	✓	✗	✗
UltraTool(Huang et al., 2024a)	✗	0.0	0.0	✓	✗	✗	✓	✗	✗

Table 1: Comparative Analysis of the Multi-Mission Tool Bench against other benchmarks in the field. The symbol '*' indicates Multi-Mission, while '†' denotes Related Missions. Moreover, in the four-mission action-type space, the Mission Switching Space Scale ($MSSS_4$) represents the proportion of combination coverage for each dataset relative to all possible combinations.

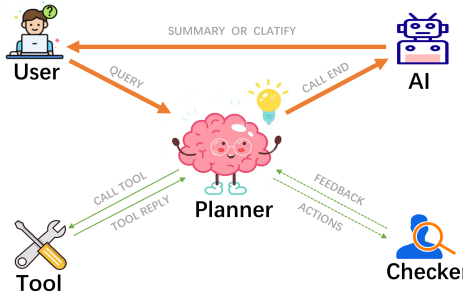

Figure 3: The multi-agent framework.

3 Terminologies

We use agent action-type to describe the mission-type switching patterns. In this section, we introduce the concepts of agent action-type and mission switching space.

As stated above, agents use four types of action to accomplish user missions: invoking a single tool, invoking multiple tools, chatting with the user, and invoking tools after clarifying their parameters. We denote these action-types as A_{single}, A_{multi}, A_{chat}, and $A_{clarify}$ respectively. As inter-tool dependencies cause diverse execution sequences, we further divide A_{multi} into the following categories: serial execution, parallel execution, and a combination of both, represented as A^S_{multi}, A^P_{multi}, and A^{S+P}_{multi}.

Furthermore, we define the concept of mission switching space to describe the combination of action-types corresponding to serially related missions, labeled $\mathbf{A}_N = \{A_0, A_1, \ldots, A_N\}$. Here, N is the total number of missions and A_i is the action-type corresponding to the i-th mission.

4 Benchmark Construction

To construct multi-mission test data, and thoroughly explore the mission switching space, we proposed a novel data generation framework. In this section, we explain the proposed framework and how to construct the benchmark. Subsection 4.1 presents the five roles in the framework and their interaction mechanism. Subsection 4.2 describes how these roles complete a mission. It includes specifying mission-types and setting up dependencies with earlier missions for later ones. Subsection 4.3 we expand the scope from generating a single mission to creating a test data with multiple related missions. Subsequently, we thoroughly explore the mission switching space to construct the entire benchmark. Furthermore, Appendixes A and B present the method for collecting tools and the distribution of the test set.

4.1 Data Generation Framework

We employ five agents to generate multi-mission test data. We simulate this process with a single LLM. For each dialogue, we assign different roles and specific tasks to the LLM, denoted $\mathbf{R}$. We define five roles: User, Planner, AI, Checker, and Tool, represented as R_{user}, $R_{planner}$, R_{AI}, $R_{checker}$, and R_{tool} respectively. The Planner is the key to analyzing the mission, planning tool invocation paths, and deciding action-types. Figure 3 shows the interaction among these five roles.

In this framework, only R_{AI} communicates with R_{user}, and $R_{planner}$ gets instructions from R_{user}. When $R_{planner}$ starts A_{single} or A_{multi}, R_{tool} simulates tool feedback. For $A_{clarify}$ or A_{chat}, R_{AI} asks about tool parameters or summarizes responses. $R_{checker}$ checks the format and sequence of $R_{planner}$'s plans, ensuring accurate planning. Note that $R_{checker}$ is only involved in data generation. Moreover, R_{user} has different tasks at different stages. R^Q_{user} responses to generate a new mission, while R^A_{user} responses to answer

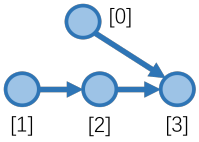

Figure 4: The dependencies among tools.

the questions of R_{AI}.

We provide the prompts for the roles mentioned above in Appendix E.

4.2 Generate Single Mission

We first introduce how to construct a single mission using the proposed multi-agent framework.

In the generation process, we first generate user missions. When generating user missions, we first sample a tool list for the missions.

To achieve a desirable mission type, we insert the predefine action-type A_i into the role prompt R^Q_{user}.

To generate related missions, we generate several candidate missions, then employ expert refinement to get the final successive mission. We categorize mission relationships into three types: implicit understanding, ellipsis, and long-term memory, and insert the relationship types into R^Q_{user} to generate three candidate missions. The R^Q_{user} also contains the previous user-AI dialogues. Finally, we manually select and refine the candidate missions to achieve the final one.

With the user missions, we use the five roles mentioned above to complete the entire execution.

4.3 Construct the Whole Benchmark

In Subsection 4.2, we obtain the ability to generate a specific type of mission and create related missions. Subsequently, we apply this ability to construct the benchmark. This benchmark aims to fully demonstrate the diversity of mission switching in the test data. To achieve this goal, we employ the proposed method to explore the entire mission switching space in prefixed mission number.

First, we identify all combinations of action-types for the given number of missions, represented as $\mathbb{A} = \mathbf{A}_1^1, \mathbf{A}_1^2, ..., \mathbf{A}_N^{4^N}$. Here, $\mathbf{A}_i^j$ indicates the j-th combination for i missions. For i missions, there exist 4^i combinations.

Subsequently, we generate test data independently for each action-type combination. If the action-type combination contains N elements, we use the aforementioned generation framework N times to construct the test data. It is important to note that the generation results from both R_{tool} and R_{AI} are crucial information provided to the agents during our testing process.

5 Dynamic Evaluation Method

The dependencies among tools lead to multiple possible execution sequences. This challenge becomes more pronounced in multi-mission scenarios. To address this, we propose a novel evaluation framework. This framework accurately verifies the correctness and optimality of agent actions. The method follows three steps: dependency analysis, decision tree construction, and path validation.

First, we manually identify tool dependencies. We then implement a topological sorting algorithm with depth-first search to generate all possible execution paths. Unlike previous methods (Qiao et al., 2024a; Shen et al., 2024b) that produce limited suboptimal paths, our algorithm constructs complete optimal and suboptimal sequences.

During agent testing, we perform incremental path matching against the decision tree. Each agent action triggers either: 1) Path termination for mismatched actions. 2) Subtree pruning for valid actions, narrowing subsequent options.

To illustrate the process clearly, take a simplified toy example. Consider a user aiming to create a PowerPoint presentation about the year's most popular movie. This task requires four tools: Tool 0 for creating the presentation, Tool 1 for retrieving the popular movie ranking, Tool 2 for gathering detailed movie information, and Tool 3 for transforming this information into slides, labeled as [0], [1], [2], and [3] respectively.

Analysis shows [2] needs parameters from [1], and [3] depends on parameters from [0] and [2]. Figure 4 shows this dependency.Figure 5 a) shows the initial decision tree based on tool dependencies. Here, [0, 1] means tools [0] and [1] are called in parallel. This tree reveals five candidate paths to complete the task with three to four tool calls.

When the agent calls Tool [1] in the first step, check if this action is among the first-step candidate actions. Then, prune the sub-decision trees related to operations [0] and [0,1], getting an updated decision tree as in Figure 5 b). In the second step, when the agent calls Tool [0], confirm the action's correctness and prune the sub-decision trees for candidate actions [0] and [0,2] in the second layer, as in Figure 5 c). At this point, only one

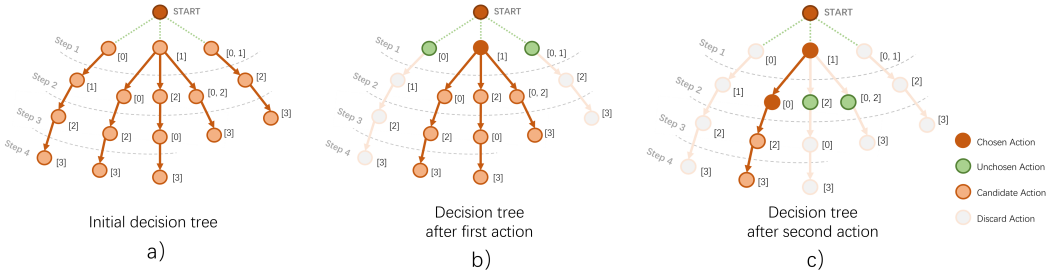

Figure 5: Visualization of the dynamic decision tree during evaluation.

candidate path remains, and verify its correctness by sequential path matching.

Additionally, we calculate two metrics. Success rate: percentage of valid paths completed. Optimality rate: percentage of paths that match minimal tool invocations. Appendix C provides formal algorithm specifications.

6 Experiments

The Multi-Mission Tool Bench consists of 1,024 test entries, each containing one to four missions. We divide the test set into four subsets based on the number of missions, with each subset containing 256 entries.

We evaluated 24 state-of-the-art models on the test set, including closed-source general models, open-source general models, and specialized models. Specifically, the closed-source general models are: o1-2024-12-17(OpenAI), GPT-4o-2024-11-20(Achiam et al., 2023), Gemini-1.5-Pro-002(Team et al., 2024), Mistral-Large-2411(Mistral), and doubao-1.5-pro-32k(Doubao). The open-source general models include: Qwen2.5-Instruction-Series(Yang et al., 2024), GLM-4-9B-Chat(GLM et al., 2024), DeepSeek-R1(Guo et al., 2025), DeepSeek-V3(Liu et al., 2024a), and Llama-3.3-70B-Instruct(Dubey et al., 2024). The specialized models are: Toolace (Liu et al., 2024b), Hammer2.1-Series(Lin et al., 2024), watt-tool-8b(Shi et al., 2024), xLAM-7b-fc-r(Zhang et al., 2024a), and gorilla-openfunctions-v2(Charlie Cheng-Jie Ji, a). Model sizes range from several hundred billions to 70b, 30b, and the smallest at 0.5b.

This section details the test results and analysis. Subsection 6.1 shows the overall performances. Subsection 6.2 analyzes effects of the number of missions, mission action-types, and mission switching. Subsection 6.3 explores the impact of inter-mission relationship types. Further error analysis is detailed in Appendix D.

6.1 Overview

This subsection analyzes the accuracy of models on the whole dataset, with Figure 6 showing the accuracy of 15 models. The models are arranged from low to high accuracy, with different colored dots indicating model types and varying dot sizes representing model sizes.

From the analysis of Figure 6, we draw the following conclusions. The o1 model, with strong reasoning capabilities, shows a significant accuracy advantage. Open-source models, such as Qwen-72b, are narrowing the gap with the top close-source models. General models like DeepSeek-V3 and doubao-1.5-pro perform well in other missions but have a clear weakness in tool utilization. Notably, small specialized models like ToolACE achieve comparable performance to large-scale general models.

Figure 7 illustrates the performance of different scale models in the Qwen2.5-Instruction-Series and Hammer2.1-Series. As shown, there is a positive correlation between model scale and accuracy. Interestingly, specialized models experience a faster decline in accuracy. To explain this phenomenon, more research is needed.

6.2 Impact of Mission Switching

This study examines the impact of mission quantity, mission-type, and mission transition on agent robustness.

Seven models with better overall performance were selected for detailed analysis, including four general models and three specialized models. Figure 8 presents the performance of these models in various subsets of mission quantities. The results indicate that specialized models perform comparably to stronger general models on single mis-

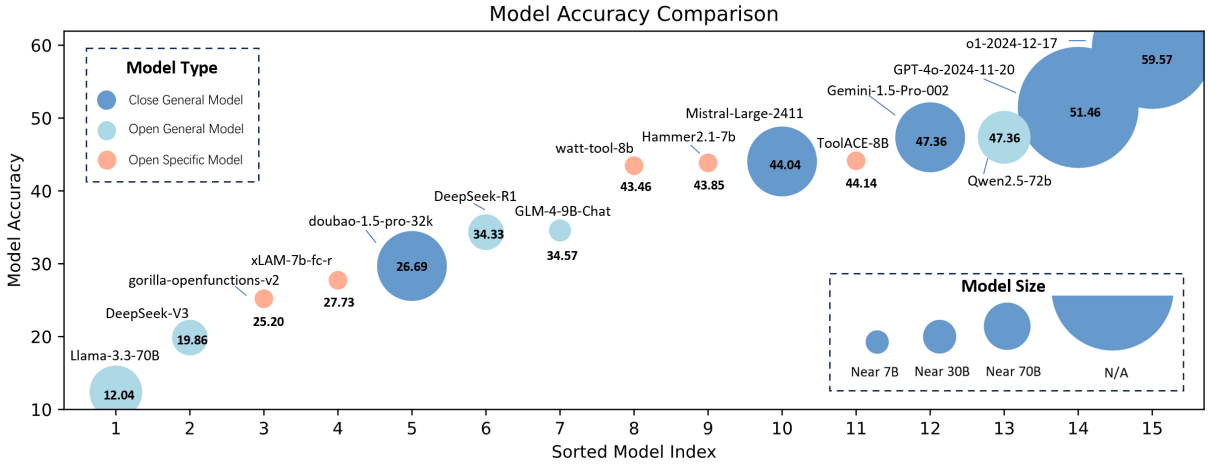

Figure 6: Overall accuracy of agents on the whole benchmark.

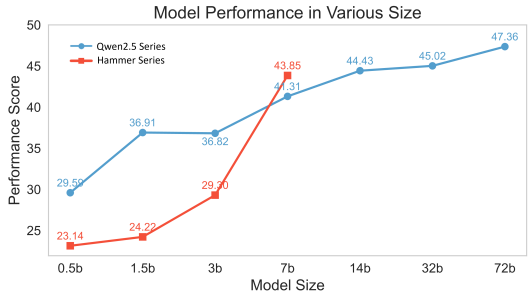

Figure 7: The performance of two series agents.

Figure 8: The impact of various mission number on the agents.

sions but experience a rapid decline in accuracy in multi-mission scenarios. This confirms our hypothesis that current research overlooks the influence of multi-mission. Furthermore, even the most advanced o1 model demonstrates a noticeable decrease in capability when handling multiple missions.

We further analyze the performance of the seven models across different action-type combinations. Following the structure of Figure 2 b), in Figure 9, we visualize the models' performance in the action-type space with heatmaps. Each heatmap pyramid represents a model's performance, with each layer corresponding to a sub-testset and its action-type combinations. Deeper colors signify higher accuracy. Greater color contrast within the same layer, with a larger proportion of lighter areas, indicates poorer robustness of the model. The findings reveal that the best performing o1 model also exhibits the highest robustness. In contrast, the three specialized models show less stability than the general models.

6.3 Impact of Mission Types

Moreover, we divide the test set by mission action-type and analyze the performance of all models, as shown in Table 2. The heatmap reveals several observations: models exhibit varying strengths and weaknesses across different action-types. For instance, most models struggle to determine whether the necessary parameters are missing($A_{clarity}$). Although many models have the ability to handle A_{multi} missions, they still face challenges in handling complex scenarios such as tackling A^{S}_{multi} and A^{S+P}_{multi} missions.

For multi-tool invocation, we introduce two new metrics, with results displayed on the far right of Table 2. The first is the optimal path rate, where the general models perform notably well. Additionally, instead of using hard labels to indicate mission success, we propose accomplished progress metric to assess model capability.

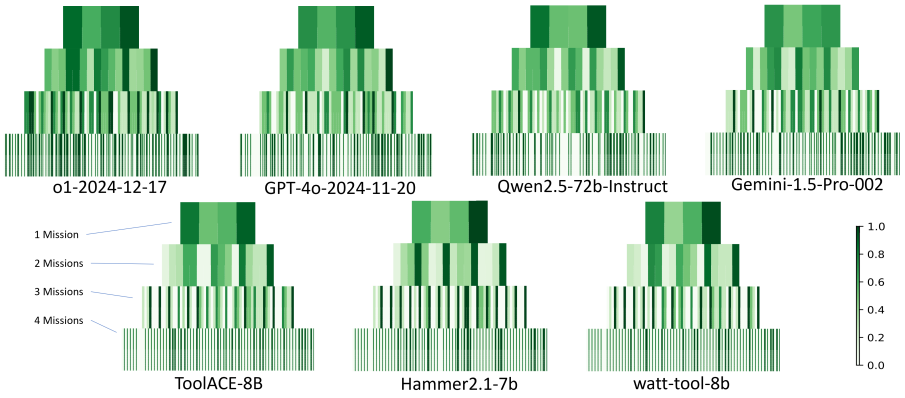

Figure 9: Visualization of the robustness of agents in the mission switching space.

Agent	A_{single}	A_{chat}	A_{clarity}	A^{P}_{multi}	A^{S}_{multi}	A^{S+P}_{multi}	Optimal Path Rate	Accomplished Progress
o1-2024-12-17†	63.28	91.41	45.70	50.32	12.50	19.05	30.15	39.42
GPT-4o-2024-11-20 †	54.69	74.61	35.94	51.59	18.75	23.81	45.56	41.08
Gemini-1.5-Pro-002†	49.61	77.73	35.94	37.58	6.25	8.33	16.58	26.14
Qwen2.5-72b-Instruct‡	56.25	74.61	27.34	45.22	18.75	7.14	19.43	30.29
ToolACE-8B⋆	43.75	87.11	22.66	35.67	0.00	3.57	9.55	24.07
Mistral-Large-2411†	57.03	55.86	31.64	41.40	12.50	16.67	37.69	29.88
Hammer2.1-7b⋆	28.13	91.27	31.25	28.03	6.25	3.57	9.72	19.67
watt-tool-8b⋆	40.63	91.80	23.05	29.94	0.00	0.00	8.38	19.50
GLM-4-9B-Chat‡	30.08	89.84	10.16	12.10	12.50	0.00	12.23	0.00
DeepSeek-R1‡	27.50	68.27	13.39	44.19	33.33	6.06	39.17	33.61
doubao-1.5-pro-32k†	60.16	25.78	5.86	36.94	18.75	9.52	38.53	5.39
xLAM-7b-fc-r⋆	14.45	86.33	5.08	7.64	0.00	1.19	9.55	4.56
gorilla-openfunctions-v2⋆	2.34	90.63	4.30	5.73	0.00	0.00	4.86	3.73
DeepSeek-V3‡	22.09	41.58	7.51	4.81	0.00	4.55	24.13	4.05
Llama-3.3-70B-Instruct‡	29.30	19.92	0.00	0.64	0.00	0.00	12.40	0.00

Table 2: The performance of agents in various type of missions, and the quantitative evaluation results on A_{multi} missions. Here, † and ‡ represent close-source and open-source general model, ⋆ represents specific model.

6.4 Impact of Related Mission

This subsection examines how mission relationship types affect agent performance. As mentioned, all subsequent missions in our benchmark are closely relate to preceding missions, and we have abstracted three types of mission relationships.

Table 3 presents the accuracy of all models in the three types of mission relationship. Long-term memory has the most significant impact on model performance, followed by the absence of core components in the problem(ellipsis).

Agent	Implicit	Ellipsis	Long-Term
o1-2024-12-17†	57.31	54.17	43.58
GPT-4o-2024-11-20†	42.69	52.92	34.64
Gemini-1.5-Pro-002†	46.99	42.08	31.84
Qwen2.5-72b-Instruct‡	40.11	47.08	28.49
ToolACE-8B⋆	38.68	35.83	27.93
Mistral-Large-2411†	35.24	39.17	30.17
Hammer2.1-7b⋆	43.55	34.58	27.93
watt-tool-8b⋆	40.97	32.92	26.26
GLM-4-9B-Chat‡	35.82	26.25	21.23
DeepSeek-R1‡	30.06	32.28	18.67
doubao-1.5-pro-32k†	25.79	28.33	22.91
xLAM-7b-fc-r⋆	30.37	22.92	19.55
gorilla-openfunctions-v2⋆	29.80	21.67	16.20
DeepSeek-V3‡	17.28	18.07	13.39
Llama-3.3-70B-Instruct‡	9.17	13.33	11.17

Table 3: The impact of mission relation types on agent performance.

7 Conclusion

This paper introduces a novel multi-mission benchmark to evaluate the robustness of LLM-based agents. Evaluations reveal that current agents exhibit varying degrees of limitations when addressing multi-mission scenarios. Notably, while specialized agents achieve comparable overall accuracy and single-mission performance to general agents, a significant robustness gap emerges in multi-mission contexts. Moreover, all agents struggle with complex multi-tool invocation missions and have shortcomings in related mission handling. We believe that these findings offer valuable insights for guiding future research on the development of LLM-agents.

Limitations

In evaluating LLM-based agents from a multi-mission perspective, we identify specific limitations in both mission duration and the data generation framework.

Firstly, our study aims to enhance the diversity of test data in terms of mission variation, yet the diversity in the number of missions remains limited. Specifically, our test data comprises up to four missions. This limitation arises because the mission switching space expands exponentially with an increase in the number of missions, leading to a rapid enlargement of the test set size and additional workload. Moreover, we observe a swift decline in the precision of the model's output as the number of missions increases, indicating that there is no immediate need to explore the model's performance across a larger number of missions.

Secondly, the proposed data generation framework employs multiple iterations and human intervention to ensure the quality of multi-turn dialogue production. This approach suffers the limitations of LLMs in accurately following instructions.

In summary, these limitations emphasize the need for ongoing development in the field of LLM based evaluations.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint*.

Fanjia Yan Shishir G. Patil Tianjun Zhang Ion Stoica Joseph E. Gonzalez Charlie Cheng-Jie Ji, Huanzhi Mao. a. Gorilla bfvl v3. `https://gorilla.cs.berkeley.edu/leaderboard.html`. Accessed: 2025-01-17.

Fanjia Yan Shishir G. Patil Tianjun Zhang Ion Stoica Joseph E. Gonzalez Charlie Cheng-Jie Ji, Huanzhi Mao. b. Gorilla openfunctions v2. `https://gorilla.cs.berkeley.edu//blogs/7_open_functions_v2.html`. Accessed: 2025-01-17.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. 2024. t-eval: Evaluating the tool utilization capability of large language models step by step. *Annual Meeting of the Association for Computational Linguistics*, pages 9510–9529.

Doubao. Doubao 1.5pro. `https://team.doubao.com/zh/special/doubao_1_5_pro`. Accessed: 2025-02-14.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *International Conference on Machine Learning*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Zishan Guo, Yufei Huang, and Deyi Xiong. 2024. Ctooleval: A chinese benchmark for llm-powered agent evaluation in real-world api interactions. *Annual Meeting of the Association for Computational Linguistics*, pages 15711–15724.

Senyu Han, Lu Chen, Li-Min Lin, Zhengshan Xu, and Kai Yu. 2024. Ibsen: Director-actor agent collaboration for controllable and interactive drama script generation. *Annual Meeting of the Association for Computational Linguistics*.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, et al. 2024a. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. *arXiv preprint*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2024b. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *International Conference on Learning Representations*.

Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. 2024. Mapcoder: Multi-agent code generation for competitive problem solving. *Annual Meeting of the Association for Computational Linguistics*.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, et al. 2024. Embodied agent interface: Benchmarking llms for embodied decision making. *Conference on Neural Information Processing Systems*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *Proceedings*

of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 3102–3116.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024b. Toolace: Winning the points of llm function calling. *arXiv preprint*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2024c. Agentbench: Evaluating llms as agents. *International Conference on Learning Representations*.

meetkai. functionary-meetkai. `https://functionary.meetkai.com/`. Accessed: 2025-01-17.

Mistral. Au large. `https://mistral.ai/en/news/mistral-large`. Accessed: 2025-02-14.

OpenAI. o1 and o1-mini. `https://platform.openai.com/docs/models#o1`. Accessed: 2025-02-14.

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

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024a. Benchmarking agentic workflow generation. *arXiv preprint*.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024b. Autoact: Automatic agent learning from scratch via self-planning. *Annual Meeting of the Association for Computational Linguistics*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. *International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Haiyang Shen, Yue Li, Desong Meng, Dongqi Cai, Sheng Qi, Li Zhang, Mengwei Xu, and Yun Ma. 2024a. Shortcutsbench: A large-scale real-world benchmark for api-based agents. *arXiv preprint*.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2024b. Taskbench: Benchmarking large language models for task automation. *International Conference on Learning Representations Workshop on Large Language Model (LLM) Agents*.

Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. 2024. Direct multi-turn preference optimization for language agents. *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324.

Alkesh K Srivastava and Philip Dames. 2024. Speech-guided sequential planning for autonomous navigation using large language model meta ai 3 (llama3). *arXiv preprint*.

Xingwu Sun, Yanfeng Chen, Yiqing Huang, Ruobing Xie, Jiaqi Zhu, Kai Zhang, Shuaipeng Li, Zhen Yang, Jonny Han, Xiaobo Shu, et al. 2024. Hunyuan-large: An open-source moe model with 52 billion activated parameters by tencent. *arXiv preprint*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint*.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. *Annual Meeting of the Association for Computational Linguistics*, pages 16022–16076.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. *International Conference on Machine Learning*.

Heng-Da Xu, Xian-Ling Mao, Puhai Yang, Fanshu Sun, and He-Yan Huang. 2024. Rethinking task-oriented dialogue systems: From complex modularity to zero-shot autonomous agent. *Annual Meeting of the Association for Computational Linguistics*, pages 2748–2763.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. t-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint*.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. 2024. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *arXiv preprint*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. 2024a. xlam: A family of large action models to empower ai agent systems. *arXiv preprint*.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024b. Agent-pro: Learning to evolve via policy-level reflection and optimization. *Annual Meeting of the Association for Computational Linguistics*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Conference on Neural Information Processing Systems*, 36:50117–50143.

A Diverse Toolset Construction

We generate the toolset based on tool descriptions from public-apis, following the ToolAlpaca approch. This API repository contains 400 tool lists, corresponding to 1600 tools in 50 categories.

In contrast to ToolAlpaca, our approach includes three strategies to enhance tool accuracy and parameter variety. Initially, we utilize LLMs like GPT to refine tool descriptions, addressing the common issue of the absence of constraint parameters in generated tools. For instance, a tool description for querying Spanish weather does not mention Spain in any of its three specific functions, leading to the generated tool cannot validate the query location. Second, we expand parameter types to include complex data structures such as enumerations, arrays, and objects, aligning better with real-world scenarios. Finally, five LLM agent experts review the generated tools. These steps ensure the tools' accuracy and parameter diversity.

B Analysis of the Test Data

Figure 10, 11 and 12 present the proposed dataset from the following three perspectives.

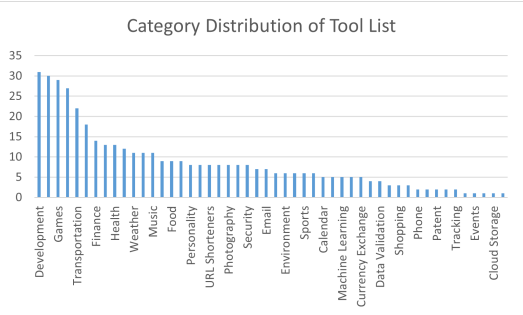

Figure 10: Category distribution of tools.

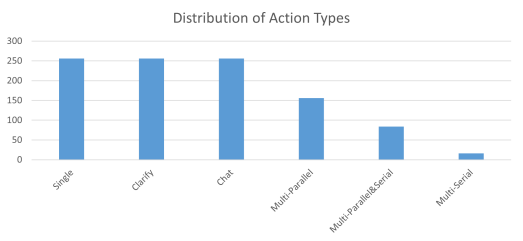

Figure 11: Distribution of action-types.

B.1 Data Examples

We present two more examples of mission execution corresponding to the examples in Section 5. Figure 13 illustrates the execution of the optimal path, while Figure 14 shows a non-optimal path execution.

C Details of Proposed Evaluation Method

1. Initialize graph G, indegree table, visitation table, current path, and all paths.

2. Perform topological sorting and depth-first traversal based on parallel combination and permutation.

2.1 For each search, find all nodes with an indegree of 0 and arrange all possible combinations based on the number of nodes. Specifically, since nodes with an indegree of 0 are independent, they can be combined arbitrarily. When the number of nodes in a combination is greater than 1, it indicates that these nodes can be called in parallel. It is this method that allows our algorithm to enumerate all possible paths, including parallel and serial-parallel calls, as opposed to being limited to serial calls only, compared to naive topological sorting.

2.2 Traverse each combination, add the combination to the current path, and update the indegree and visitation tables.

2.3 Continue with depth-first traversal until the number of nodes in the path matches the number

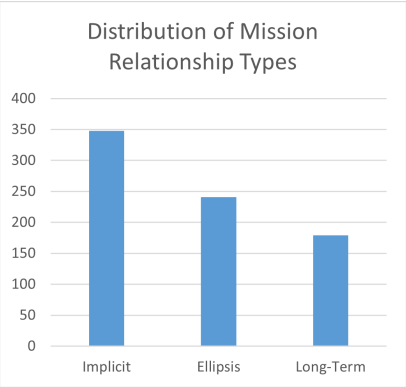

Figure 12: Distribution of three mission relationship types.

of nodes in the annotated answer, completing the generation of one path, and add it to all paths.

2.4 Repeat the above steps until the traversal is complete.

3. Based on the path length, divide into the optimal path and the suboptimal path.

D Further Analysis of Agent Performance

In addition to the analytical perspectives mentioned in the main text, we analyze the error types of the agents.

We categorize errors into tool errors and parameter errors. Specifically, we further divide parameter errors into parameter name hallucinations, parameter value hallucinations, and parameter value errors. Table 4 lists these error classifications. Stronger agents show a relatively lower proportion of tool errors. Although parameter name hallucinations occur less frequently, they are serious and widespread. The most common parameter error occurs when the agent extracts parameter values.

Table 4: The distribution of agent errors. Here, 'Hallu.' is short for hallucination.

Agent	Tool Errors	Parameter Errors		
		Name Hallu.	Value Hallu.	Value Err
o1-2024-12-17	83.33	0.24	5.07	11.36
GPT-4o-2024-11-20	75.87	0.20	8.05	15.49
Gemini-1.5-Pro-002	85.15	0.19	3.34	11.32
Qwen2.5-72b-Instruct	80.90	0.37	6.31	12.43
ToolACE-8B	90.56	0.17	1.75	7.52
Mistral-Large-2411	78.19	0.35	6.46	15.01
watt-tool-8b	90.68	0.17	3.63	5.53
GLM-4-9B-Chat	92.99	0.15	2.99	3.88
DeepSeek-R1	95.77	0.00	2.11	2.11
doubao-1.5-pro-32k	82.35	0.28	10.69	6.67
xLAM-7b-fc-r	96.36	0.27	1.35	1.89
gorilla-openfunctions-v2	98.83	0.00	0.26	0.90
DeepSeek-V3	96.57	0.00	0.90	2.53
Llama-3.3-70B-Instruct	90.53	0.33	2.45	6.69

E Part Roles Prompt of Agents

E.1 Role Prompt of Mission Generation

We show the role prompt of single mission generation in Figure 15.

E.2 Role Prompt of Planner

We show the role prompt of Planner in Figures 16, 17, 18, 19, 20, 21 and 22.

E.3 Role Prompt of Tool

We show the role prompt of Tool in Figures 23.

E.4 Role Prompt of AI

We show the role prompt of AI in Figures 24.

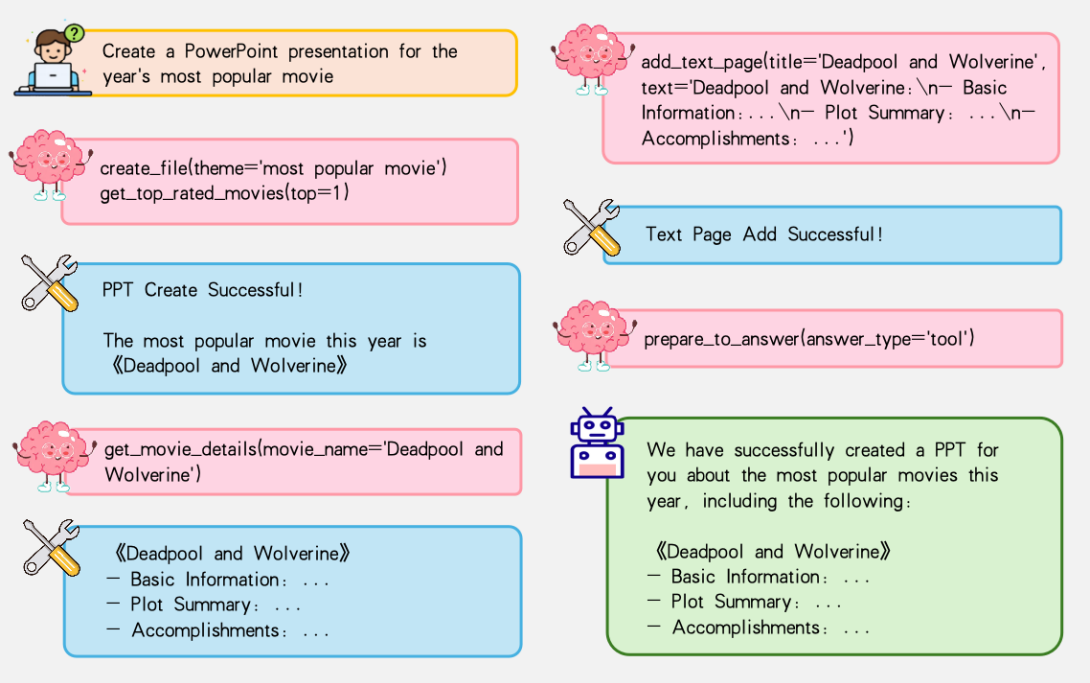

Figure 13: An Optimal Path Example.

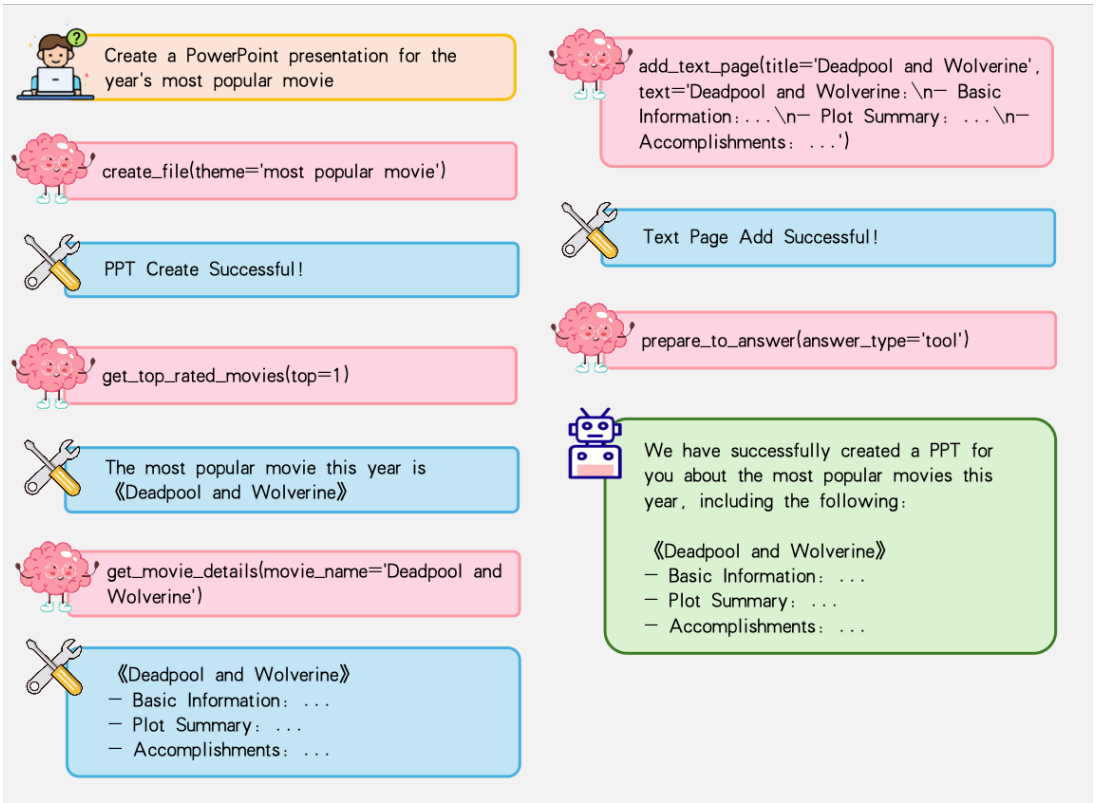

Figure 14: A Suboptimal Path Example.

Single Tool Invocation Mission Generation Prompt.

Please act as a user interacting with a super intelligent agent.

This super intelligent agent has access to a range of external tools and can use these tools to solve the missions you propose.

Next, please propose 5 missions that you need the super intelligent agent to solve based on the

All 5 missions must require the use of $\{\{\{tool\}\}\}$ from the [Tool List] to be completed, and each mission should only require a single call to $\{\{\{tool\}\}\}$.

The missions should be specific and diverse.

Finally, please output the final result according to the [Format] without generating any extra text.

The required parameters for tool $\{\{\{tool\}\}\}$ are: $\{\{\{tool\_required\}\}\}$, and the optional parameters are: $\{\{\{tool\_no\_required\}\}\}$.

[Requirements]="""
1. The description of the user's mission must include information on all the required parameters needed to call $\{\{\{tool\}\}\}$. For other optional parameters, please add them as you see fit, using natural language.
2. The user's missions should use different types of sentence structures: imperative, declarative, interrogative, etc.
3. The user's missions should include different tones: colloquial, formal, polite, direct, etc.
4. Ensure that the length of the user's missions varies, gradually increasing from short to long.
5. Ensure that the user's missions involve different themes/instances, different scenarios, and different roles.
6. Extract common entities that appear in all descriptions from the [Tool List] and ensure that these entities appear in the user's missions.
7. Do not explicitly specify the tool $\{\{\{tool\}\}\}$ in the user's missions.
"""

[Tool List]="""
$\{\{\{tool\}\}\}$
"""

[Format]="""
```
{
    "mission 1": "xxx",
    "mission 2": "xxx",
    "mission 3": "xxx",
    "mission 4": "xxx",
    "mission 5": "xxx",
}
```
"""

Figure 15: Single Tool Invocation Mission Generation Prompt.

Planner Decision Generation Prompt Part-1.

Please act as a Planner within a super intelligent agent.

You have access to a series of external tools, and you can solve user missions by invoking these external tools, as detailed in the [Tool List].

You are responsible for assessing the completion status of the current user mission and providing thoughts, plans, and actions to be executed.

If the Checker_Planner indicates 'no' for correct, it means there is an issue with the decision you made in the previous round. In this case, you should regenerate your decision based on the analysis provided by the Checker_Planner.

However, please be mindful not to include explanations of previously generated incorrect results in your Thoughts!

In your Plan, be sure not to mention the use of the prepare_to_answer tool and the ask_user_for_required_parameters tool. Instead, describe these actions in natural language, as the prepare_to_answer and ask_user_for_required_parameters tools are not to be exposed.

Refer to the [Planner Output Format] for the output format.

[Environmental Information]="""
{{{env_info}}}
"""

Figure 16: Planner Decision Generation Prompt Part-1.

Planner Decision Generation Prompt Part-2.

[Planner Output Format]="""
Planner:
{
"Mission_Finish": "Whether the user mission is completed, fill in 'yes' if completed, 'no' if not completed",
"Thought": "Based on the [Requirements] and [Environmental Information], follow the steps below to give the internal thought process when solving the user mission. You must provide an analysis of the required and optional parameters for each tool that needs to be called.
First step, decompose the mission, first analyze whether a tool needs to be called to complete it, and whether there is a suitable tool in the [Tool List].
If a tool needs to be called, which tool(s) should be used to complete the user mission, whether one or multiple tools should be called.
If multiple tools are involved, please provide an analysis of the serial and parallel nature of multiple tools.
Second step, provide an analysis of the required and optional parameters for the first tool that needs to be called (now), in the following order.
1. First, list the required and optional parameters for each tool that needs to be called.
2. Based on the context and user mission, analyze the required parameters, check which information for each tool's required parameters is provided, and explain which are provided and which are missing to ask the user.
3. Finally, analyze the optional parameters. If the user has provided information for optional parameters, briefly explain the situation; otherwise, there is no need to explain.
Note:
1. The analysis process should not be too lengthy; it needs to be concise and clear.
2. Do not have too much redundant content that is repetitive of the Plan.",
"Plan": "Based on the [Requirements], [Environmental Information], Thought, context, and user mission, provide a planning scheme.
Note:
1. When involving multiple tool calls, provide the overall plan and the plan for the first action during the first Plan, and provide the plan for the current step in subsequent dialogues.
2. The Plan is a general explanation of the Thought. The Plan does not need to set the values of the tool parameters; it only needs to explain which tools should be called to complete what missions, only the purpose of calling the tools.
3. The format of the Plan needs to be consistent with the example given in the [Requirements].
4. Do not have too much redundant content that is repetitive of the Thought.",
"Action_List": [
{
"name": "Based on the [Requirements], provide the action to be taken, i.e., the selected tool name",
"arguments": "Based on the [Requirements], [Environmental Information], and [Tool List], provide the input parameters for the action to be taken, i.e., the tool's input parameters. Note: 1. Optional parameters not specified by the user do not need to be provided. 2. Use the JSON format in terms of format, use a dictionary object, do not use strings, and there is no need to provide comments for the parameters",
"tool_call_purpose": "The purpose of the tool call"
}
]
}
"""

Figure 17: Planner Decision Generation Prompt Part-2.

Planner Decision Generation Prompt Part-3.

[Requirements]="""
*** Special Attention ***
1. When making a decision, please ensure that the tool you invoke from the [Tool List] is suitable for solving the user's mission based on the definition of the tools in the list. Do not force the use of inappropriate tools to solve the user's mission; instead, call the appropriate tool from the [Tool List] according to the user's mission.

2. Ensure that the Action_List you provide does not contradict the Plan you have set out. The order of tools in the given Action_List should be consistent with the sequence planned in the Plan.

3. For optional parameters, you only need to fill them in if the user has provided a value that is different from the default or if there is no default value. Otherwise, there is no need to include them in the arguments.

*** The prepare_to_answer tool needs to be called in the following two scenarios: ***
1. If you believe that the user's mission can be completed, then call the prepare_to_answer tool to provide a summary response, with the answer_type parameter set to 'tool'.

2. If you believe that the user's mission does not require the use of any tools from the [Tool List] or that there is no suitable tool to solve the user's mission and it can be answered directly, then call the prepare_to_answer tool, with the answer_type parameter set to 'chat'.

Note:
1) The absence of a suitable tool in the [Tool List] to solve the user's mission does not mean that you lack the ability to answer. Please respond based on the context information and the knowledge you possess. Do not excessively refuse to answer, nor imagine knowledge you do not have. Only refuse to answer when you cannot respond based on the context information and your own knowledge.

2) The absence of a suitable tool in the [Tool List] to solve the user's mission also includes the following situation:

First, analyze the common entities that appear in each tool. For example, some tools can only query data related to a certain entity A. If the user asks about entity B, it also means that there is no suitable tool.

For instance:
- If the tools in the [Tool List] can only query and analyze population data for Denmark, and the user asks for population data for Sweden, then you should also call the prepare_to_answer tool.

- If the tools in the [Tool List] can only query weather data for China, including current and historical weather, and the user asks for weather data for the United States, then you should also call the prepare_to_answer tool.

Figure 18: Planner Decision Generation Prompt Part-3.

Planner Decision Generation Prompt Part-4.

*** There are four scenarios in which the ask_user_for_required_parameters tool needs to be invoked: ***
1. If you believe that a user's mission requires the use of a tool from the [Tool List], but the user's mission is missing some required parameters from the tool, and the user needs to provide the necessary information, then invoke the ask_user_for_required_parameters tool. Please do not hallucinate parameters.

2. Please note that you are unable to deduce the values of some tool parameters on your own and will need to invoke the ask_user_for_required_parameters tool to ask the user. Please do not hallucinate parameters.

For example:
1) For the timestamp parameter, you do not have the ability to deduce the timestamp based on time. However, you can deduce other time-related parameters (start_time, end_time, etc.) on your own based on [Environmental Information], without needing to invoke the ask_user_for_required_parameters tool.
2) For ID-type parameters (station_id, product_id, etc.), you do not have the ability to deduce the corresponding ID based on the name.

3. Based on the context of the conversation, if you have already asked the user once to provide the necessary information but the user still has not provided all the required parameters, then please continue to invoke the ask_user_for_required_parameters tool.

4. If the user provides incomplete parameter values, such as the tool parameter being an IP address (ip_address), but the user provides an incomplete IP address (e.g., 192.22), please continue to use the ask_user_for_required_parameters tool to ask the user for the complete IP address.

Finally, if you confirm the need to invoke the ask_user_for_required_parameters tool, provide the inquiry plan in the format: "Ask the user to provide xxx, in order to invoke the xxx tool to xxx" in the Plan.

Figure 19: Planner Decision Generation Prompt Part-4.

Planner Decision Generation Prompt Part-5.

*** There are eight scenarios in which multiple tools need to be invoked: ***
If a user mission involves invoking multiple tools, please first analyze the dependency relationships between the multiple invocation tools. For tools that do not have invocation dependencies, perform concurrent invocations, and for tools that do have invocation dependencies, perform serial invocations. Specifically, you can handle each of the following eight scenarios separately:

Concurrent invocation scenarios:

1. If you determine that the user mission requires multiple invocations of the same tool A, but with different parameters for each invocation of tool A, then please invoke tool A concurrently and provide the concurrent invocation plan in the Plan in the format: "Concurrently invoke tool A N times for xxx."

2. If you determine that the user mission requires the invocation of different tools, such as tools A and B, and there is no dependency between tool A and B, then please invoke tools A and B concurrently, and provide the concurrent invocation plan in the Plan in the format: "Concurrently invoke tool A for xxx, while invoking tool B for xxx."

Serial invocation scenarios:

3. If you determine that the user mission requires the invocation of different tools, such as tools A, B, and C, and there are dependencies between these tools, then please invoke tools A, B, and C serially, and provide the serial invocation plan in the Plan in the format: "First, invoke tool A for xxx. Then, invoke tool B for xxx. Next, invoke tool C for xxx. Now, I will invoke tool A for xxx."

Serial invocation has the following two dependency scenarios:

3.1. Parameter dependency: For example, before invoking tool C, it is necessary to first invoke tool B to obtain the result as an input parameter, and before invoking tool B, it is necessary to first invoke tool A to obtain the result as an input parameter. Therefore, you need to first complete the invocation of tool A to obtain the result, use it as the input parameter for invoking tool B, and after obtaining the result from tool B, use it as the input parameter for invoking tool C, i.e., please invoke tools A, B, and C serially.

3.2. Logical dependency: Even if there is no parameter dependency between the invocation of tools A, B, and C, but there is a logical dependency, such as logically needing to invoke tool B before tool C, and tool A before tool B, then please also invoke tools A, B, and C serially.

Figure 20: Planner Decision Prompt Generation Part-5.

Planner Decision Generation Prompt Part-6.

Combined serial and concurrent invocation scenarios:

4. If you determine that the user mission requires the invocation of different tools, such as tools A, B, and C, and tool C depends on the invocation of tools A and B, but there is no dependency between tools A and B, then please invoke tools A and B concurrently, followed by the serial invocation of tool C, and provide the combined serial and concurrent invocation plan in the Plan in the format: "Concurrently invoke tools A and B for xxx and xxx, respectively. Then, invoke tool C for xxx. Now, I will concurrently invoke tools A and B for xxx and xxx."

5. If you determine that the user mission requires the invocation of different tools, such as tools A, B, and C, and tools B and C depend on the invocation of tool A, but there is no dependency between tools B and C, then please first invoke tool A serially, followed by the concurrent invocation of tools B and C, and provide the combined serial and concurrent invocation plan in the Plan in the format: "First, invoke tool A for xxx. Then, concurrently invoke tools B and C for xxx and xxx, respectively. Now, I will invoke tool A for xxx."

6. If you determine that the user mission requires the invocation of different tools, such as tools A and B, and there is a dependency between tools A and B, and tool A needs to be invoked multiple times, then please first invoke tool A concurrently multiple times, followed by the serial invocation of tool B, and provide the combined serial and concurrent invocation plan in the Plan in the format: "First, concurrently invoke tool A N times for xxx. Then, invoke tool B for xxx. Now, I will concurrently invoke tool A N times for xxx."

7. If you determine that the user mission requires the invocation of different tools, such as tools A and B, and there is a dependency between tools A and B, and tool B needs to be invoked multiple times, then please first invoke tool A serially, followed by the concurrent invocation of tool B multiple times, and provide the combined serial and concurrent invocation plan in the Plan in the format: "First, invoke tool A for xxx. Then, concurrently invoke tool B N times for xxx. Now, I will invoke tool A for xxx."

Special scenarios:

8. The tools prepare_to_answer and ask_user_for_required_parameters cannot be invoked concurrently with other tools and need to be invoked serially.

Figure 21: Planner Decision Generation Prompt Part-6.

Planner Decision Generation Prompt Part-7.

Please also note:

1. The dependency relationship between tool invocations refers to the necessity of completing the call to Tool A before running the call to Tool B.

2. For multiple invocations of the same tool, it is necessary to carefully analyze the dependency relationship of each call, noting that even two calls to the same tool may be interdependent.

3. If you state in your Thought and Plan that tools need to be called in sequence, then the number of tools to be called in your given Action_List cannot exceed one, otherwise, there will be a logical contradiction!

4. If you cannot ensure that parallel calls to multiple tools A, B, C will not have parameter dependencies and logical dependencies, then please call multiple tools A, B, C in sequence!

*** Special Circumstances ***

In the following three cases, there is no need to call the ask_user_for_required_parameters tool:

1. If a tool's parameter is a country's ISO code, and the user's mission mentions a specific country, such as China, you can directly deduce China's ISO code and fill it in.

2. If a tool's parameter is a longitude or latitude value, and the user's mission mentions a specific location, such as Beijing, you can directly deduce the approximate longitude and latitude values for Beijing and fill them in.

3. If a tool's parameter is a time-related parameter (such as start_time, end_time, or other parameters that include year, month, and day) and not a timestamp type, you can deduce it based on the current time in the [Environmental Information] and fill it in. At the same time, you need to explain in your Thought how you deduced the value of the time-related parameter based on the current time.

*** Other Notes: ***

1. Be sure not to provide comments for parameters, as providing parameter comments will cause JSON to fail to parse.
"""

{{{all_tool_required_info}}}

[Tool List]="""
{{{tools}}}
"""

Figure 22: Planner Decision Generation Prompt Part-7.

Tool Feedback Generation Prompt.

Please act as an external tool, Tool, within a super intelligent agent. These external tools can be used to solve user missions, as detailed in the [Tool List].

Based on the tool name and input parameters output by the super intelligent agent's Planner, simulate the execution results of the tool.

If there are multiple tools in the Action_List provided by the Planner, please simulate each one separately, ensuring the number matches the Action_List, and store the results in the Observation_List. Refer to the [Tool Output Format] for the output format.

[Environmental Information]="""
{{{env_info}}}
"""

[Tool Invocation Result Requirements]="""
1. Validate the HTTP method and parameters in the request according to the OpenAPI specification.
2. Generate a response that strictly follows the format specified in the OpenAPI specification and ensure it is in JSON format.
3. The response should contain real data, avoiding the use of placeholders.
4. Handle edge cases by providing appropriate error responses.
5. For requests without length limitations, such as the GET method, ensure the response returns 3 to 5 samples, and be careful not to use ellipses like // xxx, ... to omit sample information, as it must conform to JSON format, otherwise it will cause JSON parsing errors!
6. Try to simulate responses in English.
"""

[Tool List]="""
{{{tools}}}
"""

[Tool Output Format]="""
Tool:
{
    "Observation_List": [
        {
            "status_code": "Refer to [Tool Invocation Result Requirements] for the HTTP response status code",
            "response": "Refer to [Tool Invocation Result Requirements] to simulate the result of the action execution. Ensure your response is in JSON format, contains real data, and complies with the OpenAPI specification format."
        }
    ]
}
"""

Figure 23: Tool Feedback Generation Prompt.

AI Feedback Generation Prompt.

Please act as an Agent assistant within a super intelligent agent, which has a series of external tools. The Planner within the super intelligent agent can solve user missions by calling external tools, as detailed in the [Tool List].

You are responsible for interacting with the user. Based on the results returned by the Planner and Tool, combined with the user mission and the context of the conversation, you provide answers, and only your answers will be displayed to the user.

Refer to the [Agent Assistant Output Format] for the output format.

[Environmental Information]="""
{{{env_info}}}
"""

[Agent Assistant Output Format]="""
Agent Assistant: According to the [Requirements], reply to the most recent round of content starting with "User:" in the context conversation information (do not repeat this sentence).
"""

[Requirements]="""
1. The reply must start with "Agent Assistant:".
2. Summarize the user mission from the most recent round starting with "User:" based on the context conversation information.
3. Use markdown format, and be sure to pay attention to the layout to make it look neat, with two line breaks between paragraphs.
4. Pay special attention! If the Observation given by the Tool is a list, and each item in the list has its own ID, such as xxx_id or xxxId, then when summarizing the reply, please retain these IDs for each item and inform the user!
5. Reply in English.
"""

{{{all_tool_required_info}}}

[Tool List]="""
{{{tools}}}
"""

Figure 24: AI Feedback Generation Prompt.