ACEBench: A Comprehensive Evaluation of LLM Tool Usage

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have demonstrated significant potential in decision-making and reasoning, particularly when integrated with various tools to effectively solve complex problems. However, existing benchmarks for evaluating LLMs' tool usage face several limitations: (1) limited evaluation scenarios, often lacking assessments in real multi-turn dialogue contexts; (2) narrow evaluation dimensions, with insufficient detailed assessments of how LLMs use tools; and (3) reliance on LLMs or real API executions for evaluation, which introduces significant overhead. To address these challenges, we introduce ACEBench, a comprehensive benchmark for assessing tool usage in LLMs. ACEBench categorizes data into three primary types based on evaluation methodology: Normal, Special, and Agent. "Normal" evaluates tool usage in basic scenarios; "Special" evaluates tool usage in situations with ambiguous or incomplete instructions; "Agent" evaluates tool usage through multi-agent interactions to simulate real-world, multi-turn dialogues. We conducted extensive experiments using ACEBench, analyzing various LLMs in-depth and providing a more granular examination of error causes across different data types.

1 Introduction

Large Language Models (LLMs), such as GPT-4 (Achiam et al., 2023), have demonstrated exceptional performance across numerous natural language processing tasks (Naveed et al., 2023; Qu et al., 2025; Mialon et al., 2023). Studies have shown that incorporating tools can significantly expand LLM capabilities, particularly in specialized domains such as mathematics (Das et al., 2024; Bulusu et al., 2024; Gou et al., 2023; Veerendranath et al., 2024), programming (Xu et al., 2024), and reasoning (Chen et al., 2022; Shao et al., 2022; Surís et al., 2023; Yang et al., 2023). On one hand, integrating tools into LLMs can enhance capabilities in multiple domains, for example, ToolTransformer (Schick et al., 2023) enhances the ability of LLMs to solve complex problems by utilizing tools. On the other hand, adopting a tool usage paradigm can improve the robustness of the response and the transparency of the generation, thus increasing the explainability and trust of usersusers (Schick et al., 2023), as well as improving the system's adaptability. As this field continues to evolve, it is essential to comprehensively evaluate all aspects of tool usage, particularly in complex scenarios.

While several studies have focused on evaluating tool usage (Yan et al., 2024; Guo et al., 2024; Wang et al., 2024a; Qin et al., 2023; Wang et al., 2024b; Zhuang et al., 2023; Lu et al., 2024), there are still some shortcomings in the existing tool-use benchmarks. Firstly, existing benchmarks lack multi-turn dialogue evaluation in real-world scenarios. For example, the multi-turn dialogues in BFCL (Yan et al., 2024) and HammerBench (Wang et al., 2024a) are composed of predefined fixed content combinations. Secondly, current tool-use benchmarks (Qin et al., 2023; Guo et al., 2024; Huang et al., 2023; Li et al., 2023) lack fine-grained evaluation and personalized data assessment. Additionally, existing benchmarks (Qin et al., 2023; Guo et al., 2024; Wang et al., 2024b) ignore the assessment of special cases, or the evaluation methods are simplistic (Yan et al., 2024), as user instructions in real life are not always perfect(Wang et al., 2024c). The model's ability to recognize and handle these issues is also crucial for evaluation. Lastly, evaluation costs are high (Qin et al., 2023; Guo et al., 2024), as many studies rely on advanced large models for evaluation.

To address these shortcomings, we propose ACEBench, a comprehensive tool-use benchmark that includes the following categories:

Normal. Consists of fixed question-answer pairs

Table 1: Comparison of benchmarks across different evaluation criteria. "LLM-Free" refers to result evaluation without relying on LLMs. "Robustness" refers to incomplete or unclear user instructions. "Interactiveness" refers to the dynamic interaction between the model and the environment. "Atomic-Level" refers to analyzing from the atomic-level capabilities. "Personalization" refers to the inclusion of personal likes.

Benchmark	LLM-Free	Robustness	Interactiveness	Atomic-Level	Personalization
MetaTool (Huang et al., 2023)	✓	✗	✗	✗	✗
API-Bank (Li et al., 2023)	✓	✗	✗	✗	✗
Stable ToolBench (Guo et al., 2024)	✗	✗	✗	✗	✗
BFCL (Yan et al., 2024)	✓	✓	✗	✗	✗
τ-Bench (Yao et al., 2024)	✓	✗	✓	✗	✗
HammerBench (Wang et al., 2024a)	✗	✓	✗	✗	✗
ACEBench (Ours)	✓	✓	✓	✓	✓

and encompasses a variety of scenarios, including single-turn dialogues, multi-turn dialogues, and personalized scenario data. It also includes evaluations of atomic-level capabilities.

Special. Includes imperfect instructions, such as instructions containing incomplete parameters, incorrectly formatted parameters, or questions irrelevant to the capabilities of the candidate functions.

Agent. Encompasses real-world scenarios, abstracted to construct multi-turn, multi-step tool invocation scenarios, divided into multi-turn and multi-step cases depending on whether the user participates in the dialogue process.

The three categories above cover most of the tool usage scenarios for LLMs, and detailed explanations of each category can be found in Appendix A. Our main contributions are as follows:

- **Comprehensive Benchmark Evaluation.** We propose a comprehensive benchmark for evaluating LLMs' tool usage, covering various scenarios, including more fine-grained evaluation perspectives and assessments under imperfect instructions and providing more stable evaluation metrics.
- **Sandbox Environment and Automated Evaluation System.** We build an end-to-end automated evaluation system and develop a sandbox environment construction scheme for multi-turn, multi-step tool invocation based on real-world scenario abstraction.
- **Extensive Experimental Validation.** Through extensive experiments, we demonstrate our benchmark provides a more comprehensive analysis with greater distinction, offering a clearer evaluation of LLMs' tool usage.

2 Related Works

The emerging trend of leveraging LLMs' tool-use capabilities in real-world applications underscores the need for comprehensive evaluations of their performance and effectiveness. Despite recent advancements, existing benchmarks for evaluating the tool-use capabilities of LLMs still have significant limitations

Stable ToolBench (Guo et al., 2024) addresses the issue of unstable external APIs by employing a virtual API server, but its dependence on large models for evaluation results in high costs and scalability challenges. BFCL (Yan et al., 2024) introduces a benchmark for tool use in multi-turn dialogue scenarios. Yet, it assembles dialogues from fixed content, failing to capture the dynamic and adaptive nature of real-world interactions. Similarly, τ-Bench (Yao et al., 2024) evaluates language agents' ability to engage with human users while adhering to domain-specific rules. Still, its narrow focus on just two scenarios limits its generalizability across diverse tasks. HammerBench (Wang et al., 2024a) improves upon this by incorporating datasets derived from popular mobile applications and merging dialogues to simulate typical question-answer trajectories. However, like BFCL, its multi-turn dialogues are simplistic concatenations of pre-defined content, which do not reflect the complexities of real-world conversational dynamics. In addition, some benchmarks (Qin et al., 2023; Guo et al., 2024) rely on large language models (LLMs) for result evaluation, leading to high costs and unstable operations.

In contrast, our work addresses these limitations by expanding the scope of evaluation to encompass a broader range of tool usage scenarios. We propose a framework that simulates realistic multi-turn dialogue processes and enables end-to-end au-

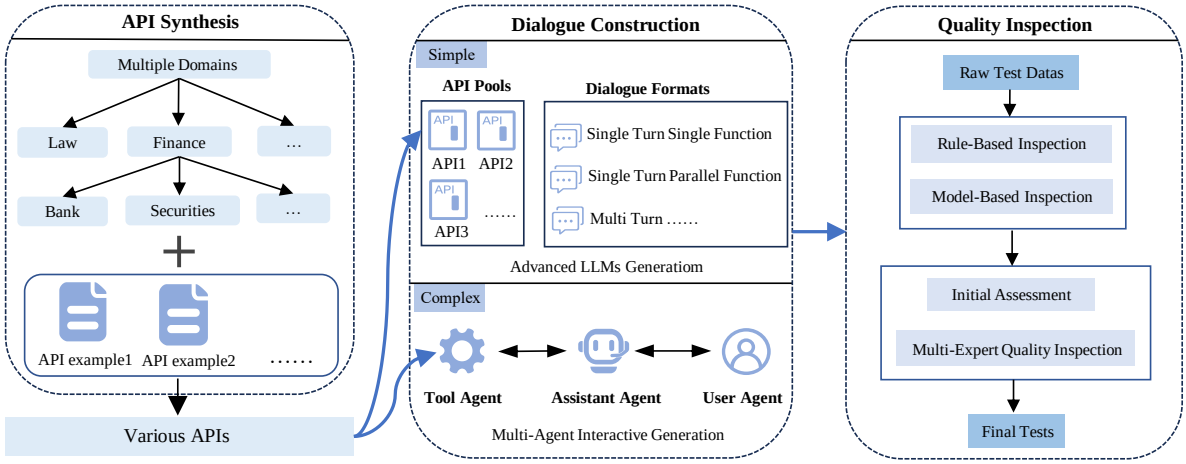

Figure 1: Evaluation dataset construction pipeline: API Synthesis Module (left), Dialogue Generation Module (middle), Quality Inspection Module (right).

tomated assessment, thereby reducing evaluation costs and improving scalability. A comparative analysis of ACEBench against recent benchmarks, as shown in Table 1, demonstrates its effectiveness in overcoming these challenges.

3 ACEBench

3.1 Dataset

We construct two versions of the dataset, one in Chinese and the other in English, ensuring an equal distribution of data types across both versions.

3.1.1 Data Generation

The data construction process is divided into two categories: (1) the construction of Agent data, as detailed in Appendix B.1, and (2) the construction of other types of data, which includes two primary steps: API synthesis and dialogue generation.

API Synthesis. We use real APIs from various real-world scenarios as reference during synthesis to enhance authenticity. To ensure the stability of the data, we use synthetic APIs to construct the evaluation dataset, referencing real-world APIs as a guide. We employ a self-evolution approach by building a hierarchical API context tree to ensure the generated APIs cover a wide range of domains and functionalities (Liu et al., 2024b). Initially, we extract relevant information from technical documents to guide the API generation. As the process progresses, the context tree is gradually expanded, ultimately ensuring the depth and breadth of the generated APIs. The left part in Figure 1 illustrates the generation of APIs.

Dialogue Construction. As shown in the middle part in Figure 1, we use two different dialogue generation pipelines built on the constructed API pool from which three to six candidate APIs are selected for each evaluation instance. For most cases, APIs are chosen randomly. However, for instances requiring specific functionality (e.g., similar APIs or multi-turn scenarios), advanced methods, including graph-based sampling (Wang et al., 2024d), are used. Simple cases or those with predefined functionality use a template-based generation, where a single generator produces dialogues to ensure consistency. We employ a multi-agent dialogue pipeline for more complex scenarios, where three agents (user, assistant, and tool) role-play to simulate real-world interactions. Both pipelines are supported by carefully hand-crafted examples to ensure comprehensive coverage and diversity. A detailed description for special data construction is provided in Appendix B.2.

3.1.2 Multi-Stage Data Verification

To address issues like mismatched answers or ambiguous criteria, we have implemented a multi-stage verification process, shown on the right part of Figure 1.

Automated Quality Inspection. The data first undergoes a rule-based quality inspection module, which evaluates four dimensions: clarity of API definitions, executability of function calls, accuracy of dialogues, and consistency of data samples, effectively filtering out formatting and spelling errors. Next, the data enters the model-based quality verification module, which uses LLMs to detect semantic errors, employing a voting mechanism to

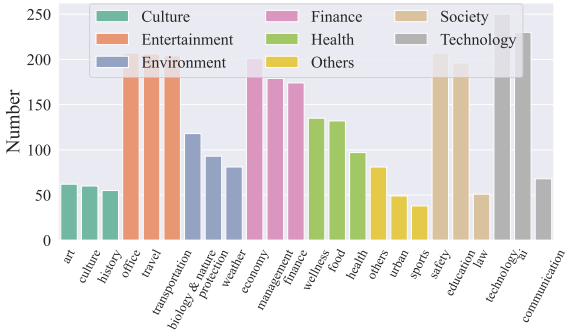

Figure 2: Distribution of APIs in terms of domains (Top 3 subcategories for each category).

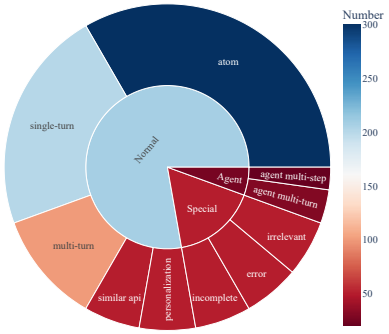

Figure 3: Visualization of the data composition of ACEBench.

ensure consistency in evaluation.

Human Quality Inspection. In the initial evaluation, the dataset remaining after automated quality inspection is assessed by three LLMs to assist human experts in data screening. Valid data is retained, while potentially problematic data is placed in the error candidate pool (approximately 20% of the dataset). These flagged entries undergo a two-step expert review process, where two experts independently assess and suggest modifications, and a third expert consolidates feedback, revising problem statements, API definitions, and answers. The revised data is re-evaluated and manually verified, and three rounds of optimization are performed to ensure a high-quality dataset.

3.1.3 Data Analysis

To demonstrate the breadth and comprehensiveness of ACEBench, we provide a detailed analysis of its test case distributions. Specific examples of each data type can be found in Appendix C.

Domain of APIs. The ACEBench API boasts a comprehensive coverage of 8 major domains and 68 sub-domains, spanning various aspects of daily life, including technology, finance, entertainment, society, health, culture, environment, and others. It offers a rich collection of 4,538 APIs in both Chinese and English. The distribution of these APIs is visualized in the accompanying Figure 2.

Data Composition. ACEBench consists of three categories of test samples: Normal, Agent, and Special, where each category is divided into several subcategories. The data composition is visualized in Figure 3, demonstrating a comprehensive coverage of tool-use capabilities, from simple single-turn tool invocations to complex multi-turn interactions involving users and environments. They include scenarios requiring multiple steps and interactions with the environment, as well as cases where tool calls are infeasible.

Number of turns and arguments. The test data in ACEBench covers a wide range of complexities. Specifically, we statistically analyzed the number of dialogue turns and the number of arguments in the called apis, which are visualized in Figure 4. The results show that the number of dialogue turns ranges from 1 to 8, encompassing most real-world scenarios. These samples with varying numbers of turns and arguments further form a test suite that covers a broader range of difficulties, allowing ACEBench to conduct a more granular evaluation of different models.

3.2 Eval

In this section, we introduce the evaluation process, where different evaluation methods are applied based on the various data types. The inference prompt for evaluation can be found in Appendix D.

3.2.1 Normal Evaluation

As shown in the left part of Figure 5, we evaluate Normal Data by comparing the model's function call outputs with the ground truth. We evaluate using AST parsing, starting by matching the func-

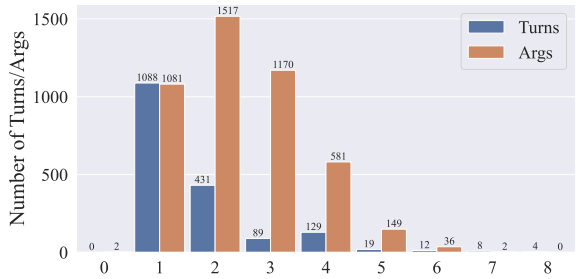

Figure 4: Distribution of dialogue turns and API argument numbers.

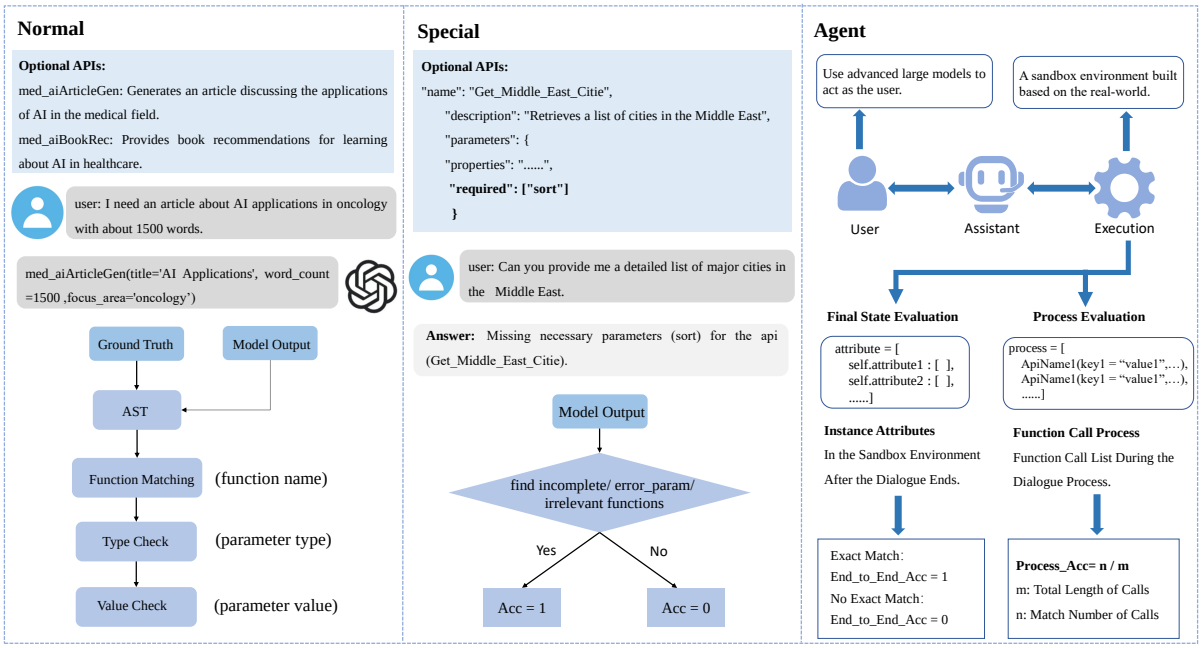

Figure 5: Overview of evaluation process: The left represents 'Normal' evaluation, which involves AST parsing for function and parameter checks. The middle illustrates 'Special' evaluation, focused on error type matching. The right shows 'Agent' evaluation, which assesses final state and intermediate results through user-model interaction.

tion name to the target for exact correspondence, followed by verifying the consistency of parameter types, and finally confirming the accuracy of the parameter values. For cases with multiple possible correct answers, we use a candidate answer pool, and a match with any one of the candidates is considered correct. The Normal Data is evaluated using Accuracy, where a score of 1 indicates a complete match and 0 indicates a mismatch.

3.2.2 Special Evaluation

As shown in the middle part of Figure 5, the core goal of evaluating Special Data is to determine whether the model can accurately identify and point out the issue. Specifically, when the instruction is missing required parameters, the model needs to identify and indicate the missing parameter; when there is an error in the parameter, the model needs to correctly identify the error parameter; and when the task is irrelevant to the functions, the model needs to recognize that the function cannot process the request. In any of these cases, if the model correctly identifies and addresses the issue, the Accuracy is 1; otherwise, the Accuracy is 0.

3.2.3 Agent Evaluation

As shown in the right part of Figure 5, we evaluate the Agent Data by assessing the model's ability to use tools in real-world interactions with humans, focusing on both result and process evaluation. There are two evaluation metrics:

End-to-End Accuracy is evaluated by comparing the instance attributes of the corresponding class with the target. If all attributes match exactly, the Accuracy is 1; otherwise, the Accuracy is 0.

Process Accuracy is determined by the consistency between the actual function call process and the ideal process. It is expressed as $\frac{n}{m}$, where m represents the ideal function call process, and n represents the degree of match between the actual and ideal processes.

3.2.4 Overall Accuracy

The Overall Accuracy is computed as a weighted sum of the accuracies for the Normal, Special, and Agent data types, where the weights are determined by the square roots of their respective sample sizes. The details can be found in Appendix E.

4 Experiments

In this section, we present a comprehensive set of experiments designed to evaluate the performance of LLMs on ACEBench.

Experimental Setup. In our evaluation, we examine seven closed-source LLMs, including the GPT-4 series (Achiam et al., 2023), Qwen-Max (Yang et al., 2024), Gemini-1.5-Pro (Team et al., 2024),

Table 2: Comprehensive evaluation of different models on ACEBench for Chinese and English combined (%).

Model	Normal						Special	Agent	Overall
	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Summary			
Closed-Source Large Language Models									
GPT-4o	93.4	84.5	77.0	85.0	83.0	87.6	93.0	63.8	**85.4**
GPT-4-Turbo	93.2	84.8	77.5	86.0	86.0	88.0	86.7	67.5	**84.5**
Qwen-Max	91.2	80.5	68.0	83.0	83.0	84.2	74.0	64.3	**78.4**
GPT-4o-Mini	86.5	76.0	66.5	77.0	78.0	79.9	79.0	33.3	**72.5**
Gemini-1.5-Pro	84.5	76.8	64.5	80.0	78.0	79.0	78.7	25.5	**70.7**
Claude-3-5-Sonnet	76.9	72.5	62.5	71.0	72.0	72.9	77.4	39.5	**68.9**
Doubao-Pro-32k	79.8	55.5	58.0	76.0	66.0	70.7	55.0	25.0	**59.4**
Open-Source Large Language Models									
Qwen2.5-Coder-32B-Instruct	90.2	81.0	71.0	83.0	81.0	84.1	80.7	60.8	**79.6**
DeepSeek-V3	91.5	84.0	77.0	83.0	83.0	86.5	73.0	34.5	**74.8**
Qwen2.5-72B-Instruct	86.8	80.3	69.5	83.0	81.0	82.1	75.7	45.0	**74.7**
Llama-3.1-70B-Instruct	82.5	68.3	63.5	79.0	68.0	75.5	38.3	42.3	**60.4**
Qwen2.5-7B-Instruct	76.0	60.3	58.5	72.0	67.0	69.4	47.0	13.8	**54.8**
DeepSeek-Coder-V2-Lite-Instruct	75.2	57.8	46.5	72.0	65.0	66.4	40.3	2.0	**49.5**
Qwen2.5-Coder-7B-Instruct	76.0	63.8	57.5	74.0	68.0	70.1	22.3	15.5	**48.9**
Watt-Tool-8B	85.7	69.3	55.5	79.0	64.0	75.6	6.0	2.8	**45.7**
Hammer2.1-7B	73.7	57.5	40.0	62.0	55.0	62.8	14.7	16.8	**42.9**
Llama-3.1-8B-Instruct	51.9	39.8	28.0	66.0	46.0	46.6	21.0	5.3	**33.4**
Phi-3-Mini-128k-Instruct	57.2	39.3	23.0	58.0	32.0	46.5	18.7	0.8	**32.0**
xLAM-7B-r	43.5	22.0	19.0	61.0	0.0	33.7	2.7	8.8	**21.6**
Llama-3.2-3B-Instruct	38.7	15.3	9.0	42.0	32.0	29.6	9.4	0.0	**19.6**
Hammer2.1-3B	22.4	11.5	3.5	40.0	20.0	18.7	1.0	1.5	**11.3**

Claude-3.5-Sonnet (Anthropic, 2024), and Doubao-Pro-32K (ByteDance, 2025). Additionally, a wide range of open-source language models are assessed, such as the Qwen2.5 series (Yang et al., 2024), Llama3 series (Dubey et al., 2024), Phi-3-Mini (Abdin et al., 2024), Deepseek-V3(Liu et al., 2024a), and DeepSeek-Coder-V2 (Zhu et al., 2024). Furthermore, four tool-learning-enhanced models were evaluated: Hammer2.1-3B, Hammer2.1-7B (Lin et al., 2024), xLAM-7B-r (Liu et al., 2024c), and Watt-Tool-8B (Watt-AI, 2024).

4.1 Main results and analysis

The comprehensive experimental results for the Chinese and English datasets are presented in Table 2, with detailed results for each language provided in Appendix F. We can draw the following important conclusions:

General Conclusion on Model Performance. The overall best performance remains dominated by closed-source models, such as the GPT-4 series. However, the performance gap between certain open-source models, such as Qwen2.5-Coder-32B-Instruct, Qwen2.5-72B-Instruct and DeepSeek-V3, and their closed-source counterparts is progressively narrowing. This trend suggests that open-source models are steadily catching up to closed-source models, driven by advancements in model architecture and training methodologies.

Performance Variations Between Top Open-source and Closed-source Models. As shown in Figure 6, the performance of these top models is generally similar in basic ability tasks, with the GPT-4 series models leading in most tasks, demonstrating strong intelligence and adaptability. However, closed-source models outperform open-source models due to more refined internal optimizations, access to larger datasets, and potentially larger scales. A noticeable performance gap emerges in mid-to-high-level tasks, such as Special and Agent, as these tasks require higher-level abilities, such as handling abnormal situations, complex reasoning, strong multi-turn interaction and decision-making capabilities.

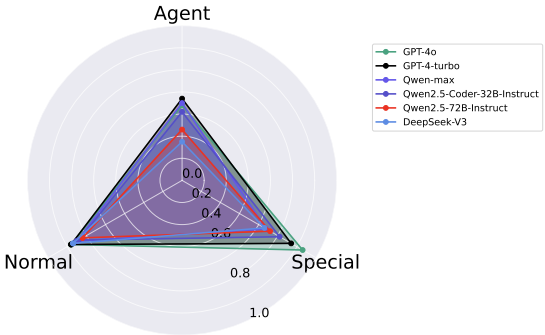

Figure 6: Performance of Top-3 closed/open-source models.

Loss of Generalization in Fine-Tuned Models.

Table 3: The Accuracy of different models on Special Data (%).

Model	Incomplete	Error	Irrelevant
Llama-3.1-8B-Instruct	29.0	20.0	14.0
Watt-Tool-8B	7.0	1.0	10.0
Qwen2.5-7B-Instruct	26.0	36.0	79.0
xLAM-7B-r	1.0	3.0	4.0
Llama-3.2-3B-Instruct	13.0	12.0	3.0
Hammer2.1-3B	0.0	3.0	0.0

As shown in Table 3, models fine-tuned on specific datasets, such as Watt-Tool-8B (Watt-AI, 2024), xLAM-7B (Liu et al., 2024c), and Hammer2.1-7B(Lin et al., 2024), exhibit a significant decline in performance on the Special dataset. This decline can primarily be attributed to the fact that while fine-tuning enhances a model's performance on specialized tasks, it can also lead to a loss of generalization, making the model less effective on new or broader instruction-following tasks.

Performance Limitations of Large Models in Complex Tasks. As shown in Table 4, most models exhibit an end accuracy of less than 50% on Agent data tasks. This can be attributed to the fact that completing such tasks in dynamic environments, which simulate real-world multi-turn interactions, requires more than just performing individual tool operations. The model must also integrate contextual information during tool usage and account for the interdependencies between tool calls, which significantly increases task complexity. Furthermore, these tasks demand advanced reasoning and adaptability, which even large models may struggle with due to the challenges of maintaining consistency across long-term interactions and responding to the evolving nature of the task.

Table 4: Performance evaluation of different models on Agent Data: **PA** represents Process Accuracy, **EA** represents End-to-End Accuracy (%).

Model	Multi Turn		Multi Step	
	EA	PA	EA	PA
GPT-4-Turbo	50.0	66.0	85.0	89.5
DeepSeek-V3	31.5	54.5	37.5	53.0
Claude-3-5-Sonnet	21.5	41.5	57.5	76.5
DouBao-Pro-32k	20.0	45.5	30.0	47.5
Qwen2.5-7B-Instruct	15.0	28.0	12.5	15.5
Hammer2.1-7B	8.5	33.5	25.0	42.5

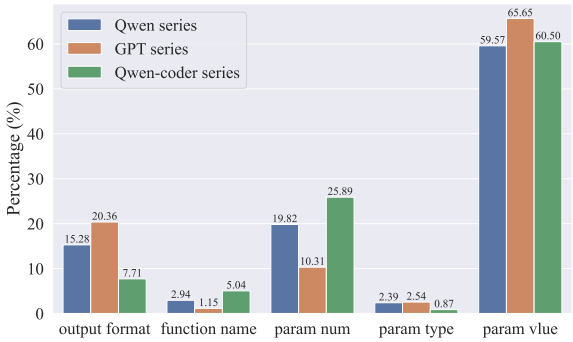

Figure 7: Error type distribution across different model series on Normal Data.

4.2 Error Analysis

Error Analysis of Normal Data. As shown in Figure 7, we observe from the error type distribution on Normal data that param value error dominate across all models. This highlights the models' difficulty in generating specific values, likely due to limited contextual understanding and the complexity of numerical distributions. Output format error is the second most common, suggesting room for improvement in generating code that follows predefined formats and syntactic rules. These issues may stem from inconsistencies in training data and the models' limited ability to learn rule-based generation. In contrast, function name and param type errors are less frequent, indicating that the models excel in matching function calls and handling data types. While the models show strong function invocation abilities, further improvements are needed in numerical generation and format compliance. Specific error examples for Normal data can be found in Appendix G.1.

Error Analysis of Special Data. As shown in Table 5, we identified two main types of model errors: The first type is "Error Detection", which refers to the model's complete failure to detect issues in the user's instructions or its inability to identify problems according to the prompt's formatting requirements. The second type is "Error Correction," where the model detects the problem but provides unclear feedback. For example, the model might indicate that there is an issue, but fails to specify which parameter values are incorrect or what critical information is missing. Results show that most errors in special-type scenarios are caused by "Error Detection", highlighting a critical gap in the model's problem-detection capabilities. This suggests that the model needs to learn not only simple

tool invocation but also how to identify corresponding issues under imperfect instructions. Specific error examples for Special data can be found in Appendix G.2.

Table 5: Error type distribution across different model series on Special Data.

Model	Error Detection	Error Correction
Watt-Tool-8B	188	4
Hammer2.1-7B	172	7
Phi-3-mini-128k-instruct	143	15
Qwen2.5-3B-Instruct	130	36
xLAM-7B-r	195	1
Llama-3.1-8B-Instruct	145	6
Hammer2.1-3B	197	0
Llama-3.2-3B-Instruct	166	9

Error Analysis of Agent Data. Our analysis identifies three primary causes of Agent errors. First, function call errors occur when the model fails to select the appropriate function or provide parameters that do not meet the required specifications, reflecting a lack of understanding of tool-use capabilities and parameter constraints. Second, rule violations arise when the model disregards predefined scene rules, skipping necessary steps or breaking key task logic, highlighting deficiencies in its comprehension and execution. Finally, information mismanagement results from the model's inability to correctly record or process contextual information during multi-turn interactions, leading to outputs that diverge from expectations. As shown in Figure 22, we illustrate an error caused by missing information.

4.3 Further Analysis

Scaling Law. We evaluated the performance of Qwen2.5-Coder (7B, 14B, 32B) and Qwen2.5-Instruct (3B, 7B, 14B, 32B, 72B) on the ACEBench dataset. As shown in Figure 8, the experimental results demonstrate that performance improves significantly across various tasks as the model size increases, with particularly strong results observed in high-complexity tasks. However, it is worth noting that as the model size continues to grow, the rate of performance improvement begins to slow down, especially between the 32B and 72B models. This indicates that while increasing the model parameters brings substantial performance gains initially, the marginal benefits of scaling up further decrease, making additional improvements more challenging.

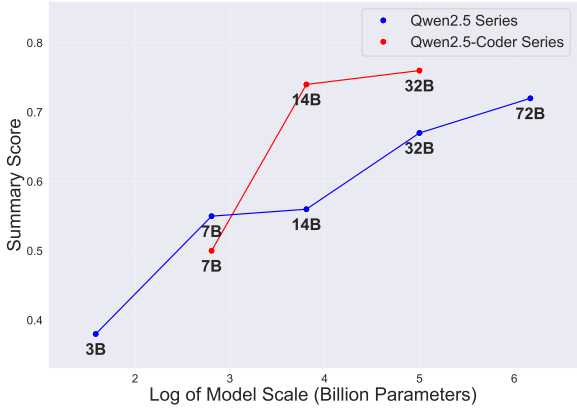

Figure 8: Scaling Law of LLMs on ACEBench.

Impact of Code Training. Based on the experimental results in Figure 8, we find that the score of Qwen2.5-Coder-7B is lower than the non-coder version, while the scores of Qwen2.5-Coder-14B and Qwen2.5-Coder-32B are higher than those of the non-coder versions of the same size. This indicates that code tuning becomes more impactful in improving model performance as the model size increases. For larger models (such as 14B and 32B), code tuning significantly enhances tool usage, especially in tasks that follow predefined rules and steps. However, for smaller models (such as 7B), despite code tuning, the differences between models are minimal, suggesting that smaller models have limited capacity and cannot fully leverage the advantages of code tuning.

5 Conclusion

This paper introduces ACEBench, a comprehensive tool-use benchmark designed to evaluate the tool-use capabilities of Large Language Models (LLMs), including data from three types: normal, special, and agent. It addresses key limitations of existing evaluation benchmarks, such as the lack of multi-turn dialogue assessments in real-world scenarios, the absence of fine-grained evaluations for parameter-type function calls, and the high costs associated with using large models for evaluation. The experimental results indicate that models fine-tuned on specific tool-use datasets to struggle with generalization when faced with complex or imperfect instructions, and code capabilities enhance the tool-use performance of large models. Through extensive experiments, we demonstrate the effectiveness of ACEBench in providing deeper insights into the tool-use abilities of various models.

Limitations

We acknowledge several limitations in our evaluation of ACEBench for assessing the tool-use capabilities of large language models. Firstly, while our test data is generated by large language models and various measures have been taken to ensure its authenticity and diversity, a gap remains when compared to data from real-world applications. This discrepancy may impact the evaluation of the model's performance in real-world scenarios. Secondly, for the Agent data, the design of evaluation scenarios relies on manual construction, which somewhat limits the diversity and coverage of the evaluation framework.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

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 arXiv:2303.08774*.

Anthropic. 2024. Claude 3.5 sonnet. Accessed: January 2025.

Arya Bulusu, Brandon Man, Ashish Jagmohan, Aditya Vempaty, Jennifer Mari-Wyka, and Deepak Akkil. 2024. Mathviz-e: A case-study in domain-specialized tool-using agents. *arXiv preprint arXiv:2407.17544*.

ByteDance. 2025. Doubao product page. Accessed: January 2025.

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

Debrup Das, Debopriyo Banerjee, Somak Aditya, and Ashish Kulkarni. 2024. Mathsensei: A tool-augmented large language model for mathematical reasoning. *arXiv preprint arXiv:2402.17231*.

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*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

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. *arXiv preprint arXiv:2403.07714*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

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. *arXiv preprint arXiv:2304.08244*.

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 arXiv:2410.04587*.

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 arXiv:2409.00920*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. 2024c. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *arXiv preprint arXiv:2406.18518*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. 2024. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343.

T. Schick, J. Dwivedi-Yu, R. Dessì, et al. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Zhihong Shao, Fei Huang, and Minlie Huang. 2022. Chaining simultaneous thoughts for numerical reasoning. *arXiv preprint arXiv:2211.16482*.

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898.

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 arXiv:2403.05530*.

Vishruth Veerendranath, Vishwa Shah, and Kshitish Ghate. 2024. Calc-cmu at semeval-2024 task 7: Pre-calc–learning to use the calculator improves numeracy in language models. *arXiv preprint arXiv:2404.14355*.

Jun Wang, Jiamu Zhou, Muning Wen, Xiaoyun Mo, Haoyu Zhang, Qiqiang Lin, Cheng Jin, Xihuai Wang, Weinan Zhang, and Qiuying Peng. 2024a. Hammerbench: Fine-grained function-calling evaluation in real mobile device scenarios. *arXiv preprint arXiv:2412.16516*.

Pei Wang, Yanan Wu, Zekun Wang, Jiaheng Liu, Xiaoshuai Song, Zhongyuan Peng, Ken Deng, Chenchen Zhang, Jiakai Wang, Junran Peng, et al. 2024b. Mtu-bench: A multi-granularity tool-use benchmark for large language models. *arXiv preprint arXiv:2410.11710*.

Wenxuan Wang, Juluan Shi, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, and Michael R Lyu. 2024c. Learning to ask: When llms meet unclear instruction. *arXiv preprint arXiv:2409.00557*.

Z. Wang, X. Zeng, W. Liu, and et al. 2024d. Toolflow: Boosting llm tool-calling through natural and coherent dialogue synthesis. *arXiv preprint arXiv:2410.18447*.

Watt-AI. 2024. Watt-tool 8b. Accessed: January 2024.

Shuyuan Xu, Zelong Li, Kai Mei, and Yongfeng Zhang. 2024. Core: Llm as interpreter for natural language programming, pseudo-code programming, and flow programming of ai agents. *arXiv preprint arXiv:2405.06907*.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Berkeley function calling leaderboard. `https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html`. Accessed: 2024-01-16.

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 arXiv:2412.15115*.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

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

A Detailed Descriptions of Test Cases

A.1 Data Categories Description

We divide the benchmark into three main categories: Normal, Special, and Agent. Below is a detailed description of each category.

Normal Data

The Normal Data consists of fixed question-answer pairs, where each question corresponds to a correct function call. It is categorized into the following categories: Single-Turn, Multi-Turn, Similar APIs, Preference, and Atom.

Single-Turn: There is only one round interaction between the user and the assistant, and based on the number of function calls in the response, it is divided into single-turn single function calls and single-turn parallel function calls.

Multi-Turn: There are multiple interactions between the user and the assistant. The conversation can be categorized into two types: **(switch)** The conversation progresses by changing topics. **(adjust)** The conversation evolves by refining or modifying the original question.

Similar APIs: The candidate APIs exhibit significant similarity, particularly focusing on the same topic. This similarity presents a challenge for the assistant, requiring it to effectively distinguish between the APIs and accurately select the most appropriate ones.

Preference: Besides the candidate APIs, the assistant is provided with supplementary user profile data. This type of information necessitates the assistant's ability to mine user-specific factors, such as past interactions, interests, or other personalized attributes, to generate argument values.

Atom: Atom Data refers to a set of APIs that contain only specific parameter types, such as candidate functions where the parameters exclusively involve numbers, lists, etc. This design is intended to explore whether the type of function parameters affects the model's ability to handle data filling. We have divided the Atom data into five types: number, enum, list, bool, and object.

Special Data

The Special Data refers to situations where the model is unable to resolve the problem posed in the instruction using the candidate functions(Wang et al., 2024c). It is categorized into the following categories: Incomplete, Error, and Irrelevant.

Incomplete: Refers to situations where the key information required for the function call is missing in the query, such as the absence of "required" parameters.

Error: Refers to situations where the instruction contains parameters or names that do not meet the required format or constraints, such as matching a specific pattern or being selected from a predefined list, causing the function call to fail.

Irrelevant: Refers to situations where the instruction exceeds the function's capabilities, meaning none of the candidate functions can resolve the issue.

Agent Data

Agent Data refers to situations where completing a task in an environment built based on real-world scenarios typically requires multiple steps. In such cases, an advanced language model simulates the user role, replicating interactions found in real-world scenarios. This approach aims to evaluate the model's ability to perform in complex interactions. We define a "multi-step" scenario as one in which the user appears only once in the entire conversation, while a "multi-turn" scenario is one where the user appears multiple times throughout the conversation.

B The Construction of Data

B.1 Agent Data

As shown in Figure 9, the construction of Agent data can be summarized in the following steps:

First, through an in-depth analysis of real-world scenarios, extract key task requirements and modularize them into sub-scenarios in different domains (such as flight booking, food delivery platforms, and financial services), clearly define the specific functional objectives of each module.

Second, implement logical abstraction for each function through code, designing core processes such as user authentication, cost calculation, payment processing, and information recording. This ensures the code logic's scalability and robustness while comprehensively addressing exception handling.

Third, design interaction rules based on specific scenario requirements to standardize the interaction process between users and the Agent, such as verifying account, merchant information, and balance status in food delivery orders, and providing

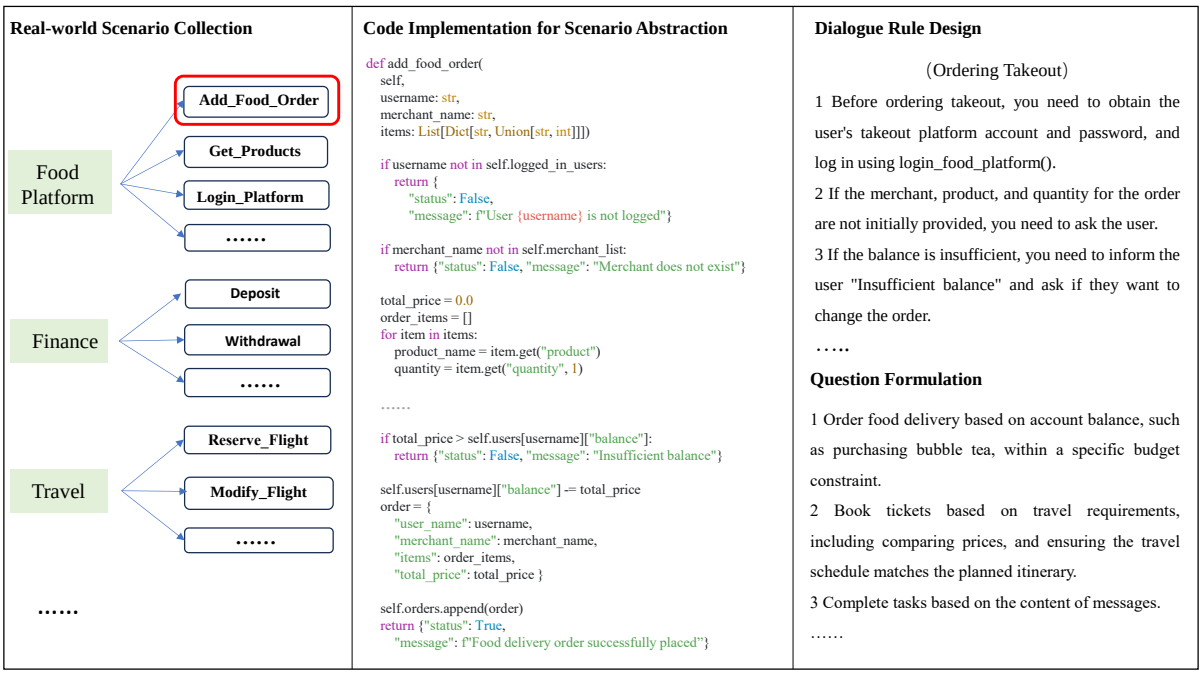

Figure 9: The construction of Agent Data. The left shows scenario sampling from real-world cases, the middle demonstrates the implementation of code tailored to specific scenarios, and the right presents examples of dialogue rules and question design for the scenarios.

user guidance in exceptional cases.

Finally, combine real-world demands to design question formats and solutions, enabling the model to accurately meet user task requirements and achieve the desired outcomes.

B.2 Special Data

Irrelevant

The irrelevant data refers to situations where the instruction exceeds the function's capabilities. And the construction method for Irrelevant data is similar to that of Normal data, where we simply remove the correct API from the candidate APIs of the common data.

Incomplete

The special data with incomplete instructions primarily refers to situations where key information is missing from the user's instructions, causing the function to be called incorrectly. Our main prompt for constructing incomplete data is shown in Figure 10. From the generated dialogue, we can extract the corresponding data and reference answers (the missing data). Next, we can convert the conversation we obtained into data. A specific example is shown in Figure 11.

Please refer to the example and continue the dialogue based on the given tool definition and the beginning of the conversation. **The requirements are as follows:**
1 The user's request is meaningful, requiring the use of one tool, and the tool will be called once.
2 Calling the tool to fulfill the user's request still lacks one or more required parameters.
.....
Here is an example for reference

<tool_definition>

```
{
"name": "calculate_triangle_area",
"description": "Given the base and height of a
              triangle, calculate its area.",
"required": ["base", "height"]
}
```

<dialogue_example>

```
[User]: I want to calculate the area of a
triangle with a base of 5 and a height of 10.
[Assistant]:
[Thought]  Missing parameters: base|height
[Response]:
Please provide both the base and the
height of the triangle.
[User]: The base is 5, and the height is 10.
[Assistant]:
<tool_usage>calculate_triangle_area
|{"base": 5, "height": 10}</tool_usage>
```

Here is the dialogue continuation you need to write:
<Tool Definition>
{tool_definition}
.....

Figure 10: Main prompt of Incomplete Data construction.

candidate function

```
{
  "name": "book_flight",
  "description": "Flight booking ...",
  "arguments": {},
  "required": ["from_city_name",
   "to_city_name", "depart_date"]
}
```

obtained dialouge

```
<user> I plan to travel to Beijing.
Help me book a flight to Beijing.
</user> [Thought] Missing parameters:
from_city_name
[Response] May I know the departure
city you want to book?
<user> Shenzhen
```

converted data

```
[question] I plan to travel to Beijing.
Help me book a flight to Beijing.
[answer] Miss necessary parameter
(from_city_name) from (book_flight)
```

Figure 11: Specific example of Incomplete Data construction.

Please refer to the example and continue the dialogue based on the given tool definition and the beginning of the conversation. **The requirements are as follows:**
1 The user's request is specific and meaningful, requiring one tool, and the tool should be called once.
2 The user's first question contains incorrect parameters (i.e., it does not conform to the function definition's pattern or format requirements).
.....
Here is an example for reference
<tool_definition>

```
{"name": "TennisTeamDetails",
"description": "Retrieve detailed
information about a tennis team .",
"parameters": {
   "properties": {
     "team_name": {......
    "pattern": "^[a-zA-Z\\s]+$"}}}}
```

<dialogue_example>

```
<user> Can you retrieve the details of
     the tennis team named 'Team@1234'?
</user> [Thought] Error Parameter: 'Team@1234'
[Response] The team name you provided does
          not meet the naming criteria.
<user> Oh, I made a mistake.
   The team name is TeamABC.
</user> <tool_usage>TennisTeamDetails
|{"team_name": "TeamABC"}</tool_usage>"
```

Here is the dialogue continuation you need to write:
<Tool Definition>
{tool_definition}
.....

Figure 12: Main prompt for Error Data construction.

Error

The special data which has error instructions mainly refers to situations where the instruction contains parameters or names that do not meet the required format or constraints, and the construction is shown in 12. Next, we can convert the conversation we obtained into data. A specific example is shown in 13.

candidate function

```
{ "name": "FootballTeamDetails",
  "description": "Retrieve information
  about a football team by its name.",
  "parameters": {
      "team_name": {......
        "pattern": "^[a-zA-Z\\s]+$"}}}}
```

obtained dialouge

```
<user> I want to know information about
the football team football$156.
</user> [Thought] Error Parameter:
         football$156.
[Response] The team name you provided
does not meet the naming criteria.
<user> ......
```

converted data

```
[question] I want to know information
about the football team footbaoo$156.
[answer] There is incorrect value
(football$156) for the (team_name).
```

Figure 13: Specific example of Error Data construction.

C Examples of Dataset

C.1 Normal Examples

Single-Turn. The example of Normal Single-Turn Data is shown in Figure 14.

Multi-Turn. The example of Normal Multi-Turn Data is shown in Figure 15.

Preference. The example of Normal Preference Data is shown in Figure 16.

Similar APIs. The example of Normal Similar APIs Data is shown in Figure 17.

Atom. The example of Normal Atom Data is shown in Figure 18.

C.2 Special Examples

Incomplete. The example of Special Incomplete Data is shown in Figure 19.

Error. The example of Special Error Data is shown in Figure 20.

Irrelevant. The example of Special Irrelevant Data is shown in Figure 21.

C.3 Agent Examples

The example of Agent Data is shown in Figure 22 and Figure 23.

D Evaluation Inference Prompts

D.1 Normal Prompt

The main evaluation inference prompt for Normal Data is shown in Figure 24 and Figure 25.

D.2 Special Prompt

The main evaluation inference prompt for Special Data is shown in Figure 26.

D.3 Agent Prompt

An example of the evaluation inference prompt for Agent Data in a specific scenario is shown in Figure 27.

E Formula for Overall Accuracy

The formula for calculating the **Overall Accuracy** can be expressed as:

$$\text{All Acc} = A \cdot \text{Acc}_{\text{Normal}} + B \cdot \text{Acc}_{\text{Special}} + C \cdot \text{Acc}_{\text{Agent}}$$

where the coefficients A, B, and C are defined as:

$$A = \frac{\sqrt{n_{\text{Normal}}}}{\sqrt{n_{\text{Normal}}} + \sqrt{n_{\text{Special}}} + \sqrt{n_{\text{Agent}}}}$$

$$B = \frac{\sqrt{n_{\text{Special}}}}{\sqrt{n_{\text{Normal}}} + \sqrt{n_{\text{Special}}} + \sqrt{n_{\text{Agent}}}}$$

$$C = \frac{\sqrt{n_{\text{Agent}}}}{\sqrt{n_{\text{Normal}}} + \sqrt{n_{\text{Special}}} + \sqrt{n_{\text{Agent}}}}$$

where $n_{\text{Normal}}, n_{\text{Special}}, n_{\text{Agent}}$ are the sample sizes for the **Normal**, **Special**, and **Agent** data types, and Accuracy refers to the accuracy of each respective category.

F Detailed Results

The accuracy evaluation of different models on English Data is shown in Table 6, and the accuracy evaluation of different models on Chinese Data is shown in Table 7.

G Error Examples

G.1 Error Examples of Normal Data

Wrong Function Name. An example of wrong function name is shown in Figure 28.
Wrong Param Type. An example of wrong param type is shown in Figure 29.
Wrong Param Value. An example of wrong param value is shown in Figure 30.
Wrong Output Format. An example of wrong output format is shown in Figure 31.

G.2 Error Examples of Special Data

Error Detection Figure 32 shows an example of error detection.
Error Correction Figure 33 shows an example of error correction.

(question)

Please assess the system with asset ID 12345, vulnerability ID 67890, with a severity level of high, and a potential threat that may lead to data leakage.

(candidate function)

```
{
"name": "Threat_Vulnerability_Assessment_Tool",
"description": "Assess potential security
threats and provide mitigation suggestions",
"parameters": ......
}
{
"name": "userPrivacySettingsManager",
"description": "Manage user's privacy",
"parameters": ......
}
```

Figure 14: An example of Normal Single-Turn Data.

(conversation)

User: I want to analyze the pest and disease risks faced by my rice crop, which is currently in the tillering stage.
system: Please provide the current temperature and humidity conditions.user: The temperature is 28°C, and the humidity is 80%.
tool:

```
[PestInfestationRiskAnalyzer(
    crop_information={"crop_type": "rice", "growth_stage": "tillering"},
    climate_conditions={"temperature": 28, "humidity": 80})]
```

system: Based on the analysis, your rice is facing a moderate risk of pests and diseases under the current climatic conditions. Please take necessary preventive measures.
User: I also want to know the maintenance requirements for my production equipment. The device ID is Device123.
system: Please provide the sensor readings of the device, including temperature, vibration, and humidity.
User: The device currently reads a temperature of 75°C, vibration level of 5, and surrounding humidity of 50%.

(candidate function)

```
{
  "name": "PestInfestationRiskAnalyzer",
  "description": "Analyze the pest and disease risk faced by the farmland ",
  "parameters": ......
}
{
  "name": "Predictive_Maintenance_AI_Tool",
  "description": "Predicting the maintenance needs of equipment using sensor data",
  "parameters": ......
}
{
  "name": "PersonalityTraitAnalyzer",
  "description": "Analyze an individual's personality traits",
  "parameters": ......
}
```

Figure 15: An example of Normal Multi-Turn Data.

(question)
Can you update my preferred notification method and also check if my current email address and home location are properly updated in the system?

(candidate function)

```
{
  "name": "updateOrderStatusAlerts",
  "description": "Sends automated alerts to users regarding the status of their current orders.",
  "parameters": ......
}
{
  "name": "submitProductReview",
  "description": "Allows users to submit a review for a product they have purchased.",
  "parameters": ......
}
{
  "name": "updateUserProfile",
  "description": "Updates the user's profile information based on provided data.",
  "parameters": ......
}
```

(profile)

```
{
  "basic_features": {
    "UserName": "Michael Smith",
    "UserEmail": "mike.smith@example.com",
    "UserHomeLocation": "Los Angeles, CA",
    "UserBirthday": "1978-04-23",
    "UserLanguage": "Spanish",
    "UserTimeZone": "PST",
    .......
  },
  "user_history": {
    "shopping": [
      "Searched for 'Nike running shoes' on app",
      "Added Nike Air Max to cart",
      "Checked coupon availability for Nike products",
      "Filtered search by 'Outdoor Equipment' category",
      "Selected 'High spending' filter for items over $500",
      ......
    ],
    "takeout": [
      "Ordered Chicken Fajitas on the takeout app for lunch",
      "Opted to receive promotional deals via phone calls",
      "Chose Debit Card ending in 5678 for payment on the takeout app",
      ......
    ]
  }
}
```

Figure 16: An example of Normal Preference Data.

(question)

My baby has had a visible vein on her nose for 5 days, and she's been crying a lot with a decreased appetite. Can you help?

(candidate function)

```
{
  "name": "baby_health_check_A",
  "description": "Checks the common reasons for baby's persistent vein visibility on the nose
     and suggests actions. This API considers factors like skin thinness, crying, or overexertion",
  "parameters": ......
}
{
  "name": "baby_health_check_B",
  "description": "Examines baby's vein visibility and recommends seeing a doctor.
      Focuses on persistent visibility and associated symptoms",
  "parameters": ......
}
```

Figure 17: An example of Similar API Data.

(question)

I need a design for my new website. It's for a technology company focusing on user engagement.

(candidate function)

```
{
  "name": "WebDesignAssistant_generateDesign",
  "description": "Generates a website design based on industry and user experience focus.",
  "parameters": {
    "type": "object",
    "properties": {
      "industry": {
        "description": "The industry for which the website is being designed.",
        "type": "string",
        "enum": [
          "Technology",
          "Healthcare",
          "Education",
          "Finance"
        ],
        "default": "Technology"
      },
      "userExperience": {......}
    },
    "required": ["industry", "userExperience"]
  }
}
{. . . . . }
```

Figure 18: An example of Atom (enum) Data.

(question)

I'm considering relocating my business to the Middle East. Can you provide me with a list of major cities?

(candidate function)

```
{"name": "Get_Middle_East_Cities",
 "description": "Retrieves a list of cities in the Middle East, sorted by overall score by default.",
  "parameters": {
    "properties": {
      "sort": {
        "description": "The sorting order for the list of cities.",
        "type": "string",
        "enum": ["asc", "desc"],
      }
    },
    "required": ["sort"]
  }}
```

Figure 19: An example of Special Incomplete Data.

(question)

Can you retrieve the tennis team details named 'Team@1234'?"

(candidate function)

```
{"name": "TennisTeamDetails",
  "description": "Retrieve detailed information about a tennis team by its name.",
  "parameters": {
    "type": "object",
    "properties": {
      "team_name": {
        "pattern": "^[a-zA-Z\\s]+$",
      }
    },
    "required": ["team_name"]
  }}
```

Figure 20: An example of Special Error Param Data.

(question)

Could you help me find available restaurants in New York City?

(candidate function)

```
{
  "name": "Get_Weather_Report",
  "description": "Retrieve the current weather report for a specified location",
  "parameters": ......
}
{
  "name": "GetTravelDestinationInfo",
  "description": "Retrieves information about a specific travel destination",
  "parameters": ......
}
```

Figure 21: An example of Special Irrelevant Data.

Question: You are Jack, and you need to order some takeout for dinner. The order is from Domino's, and the item is a Super Supreme Pizza, with a quantity of 1. If your balance is sufficient, also order a Cheese Tea from HeyTea. Your takeout platform account is "Jack", and the password is "password654".

I want to order some takeout.

Please provide your takeout platform account and password.

.....

[turn_on_wifi()]
[login_food_platform(username='Jack', password='password654')]

The WiFi is turned on, and user Jack has successfully logged in.

Which restaurant's takeout you would like to order?

The restaurant is Domino's, the item is a Super Supreme Pizza.

[add_food_order(username='Jack', merchant_name=' Domino's , items=[{'product': 'Super Supreme Pizza', 'quantity': 1}])]

.....

If the balance is sufficient, I will also order a Cheese Tea from HeyTea.

[check_balance(user_name='Jack')]

32.0 yuan.

However, the balance is insufficient.

Result Eval End_to_End Accuracy = 0

Expected Attribute

```
{
 "orders":
 [
  {
     "product": "Super Supreme Pizza",
     "quantity": 1,
     "price_per_unit": 88.0
  },
  {
     "product": "Cheese Tea",
     "quantity": 1,
     "price_per_unit": 25.0
  }
 ],
 "total_price": 113.0
}
```

Real Attribute

```
{
 "orders":
 [
    {
     "product": " Super Supreme Pizza ",
     "quantity": 1,
     "price_per_unit": 88.0
    }
 ],
  "total_price": 88.0
}
```

(After ordering the pizza, the remaining balance is sufficient to order another cup of Cheese tea.)

Process Eval Process Accuracy = 4/6

Expected Process

turn_on_wifi(…)
login_food_platform(…)
add_food_order(…)
check_balance(…)
get_products(…)
add_food_order(…)

Real Process

login_food_platform(…)
turn_on_wifi(…)
login_food_platform(…)
add_food_order(…)
check_balance(…)

Figure 22: An example of Agent Data: A partially correct representation.

Question: Your name is Grace, and you want to send Frank a message that says: 'Are you free tomorrow to have lunch with me?' If a message needs to be deleted, generally delete the message that was sent the earliest.

I want to send a message to Frank.

Who are you and what is the content of the message you want to send?

I am Grace, and the message I want to send is: 'Are you free tomorrow to have lunch with me?'

[send_message(sender_name='Grace', receiver_name='Frank', message='Are you free tomorrow to have lunch with me?')]

The capacity is full. You need to ask which message to delete.

Which message would you like to delete?

Please delete the message that was sent the earliest.

[get_earliest_message_id()]

The earliest message ID is 3.

[delete_message(message_id=3)]

[send_message(sender_name='Grace', receiver_name='Frank', message='Are you free tomorrow to have lunch with me?')]

Result Eval End_to_End Accuracy = 1

Expected Attribute

```
{
 "message":
 [ ……
  "7": {
       "sender_id": "USR102",
       "receiver_id": "USR101",
       "message": "Are you free tomorrow
to have lunch with me?"
       }
 ]
}
```

Real Attribute

```
{
 "message":
 [ ……
  "7": {
       "sender_id": "USR102",
       "receiver_id": "USR101",
       "message": "Are you free tomorrow
to have lunch with me?"
       }
 ]
}
```

Process Eval Process_Accuracy = 1

Expected Process

get_earliest_message_id (…)
delete_message (...)
send_message (…)

Real Process

send_message (…)
get_earliest_message_id (…)
delete_message (...)
send_message (…)

Figure 23: An example of Agent Data: A fully accurate representation.

You are an AI assistant with the role name "assistant". Based on the provided API specifications and conversation history from steps 1 to t, generate the API requests that the assistant should call in step t+1. The API requests should be output in the format [ApiName(key1='value1', key2='value2', ...)], replacing ApiName with the actual API name, key1, key2, etc., with the actual parameter names, and value1, value2, etc., with the actual parameter values. The output should start with a square bracket "[" and end with a square bracket "]". If there are multiple API requests, separate them with commas, for example:[ApiName(key1='value1',key2='value2',...),ApiName(key1='value1',key2='value2', ...), ...]. Do not include any other explanations, prompts, or API call results in the output. If the API parameter description does not specify otherwise, the parameter is optional (parameters mentioned in the user input need to be included in the output; if not mentioned, they do not need to be included). If the API parameter description does not specify the required format for the value, use the user's original text for the parameter value. If the API requires no parameters, output the API request directly in the format [ApiName()], and do not invent any nonexistent parameter names.

{time}

Role Descriptions:
user: User
assistant: The AI assistant role that makes API requests
tool: Provides the results returned from tool calls

API Specifications:
{function}

Figure 24: The inference prompt for Nomal (except Prefernce) Data.

You are an AI assistant, and your role is called assistant. **Based on the given API description, dialogue history 1..t, and character profile**, generate the API requests that the assistant should call in step t+1. The API requests should be output in the format [ApiName(key1='value1', key2='value2', ...)], where ApiName is replaced with the actual API name, and key1, key2, etc., are replaced with the actual parameter names, and value1, value2 are replaced with the actual parameter values. The output should start with a "[" and end with a "]". If there are multiple API requests, they should be separated by commas, e.g., [ApiName(key1='value1', key2='value2', ...), ApiName(key1='value1', key2='value2', ...), ...]. Do not output any other explanations, hints, or results of the API calls in the output. If the API parameter description does not specify special instructions, the parameter is optional (parameters mentioned in the user input or character profile should be included in the output, and if not mentioned, they should not be included). If the API parameter description does not specify the format for the parameter value, it should be taken from the user's original text or character profile. If the API requires no parameters, the API request should be output as [ApiName()], with no fabricated parameter names.

Character Profile:
{profile}

Role Descriptions:
user: User
assistant: The AI assistant role that makes API requests
tool: Provides the results returned from tool calls

API Specifications:
{function}

Figure 25: The inference prompt for Nomal (Prefernce) Data.

You are an AI system with the role name "assistant". Based on the provided API specifications and conversation history from steps 1 to t, generate the API requests that the system should call in step t+1. Below are two specific scenarios:
1. When the information provided by the user is clear and unambiguous, and the problem can be resolved using the list of candidate functions:
- If the API parameter description does not specify the required format for the value, use the user's original text for the parameter value.
- API requests should be output in the format [ApiName(key1='value1', key2='value2', ...), ApiName(key1='value1', key2='value2', ...), ...], replacing ApiName with the actual API name, key1, key2, etc., with the actual parameter names, and value1, value2, etc., with the actual parameter values. The output should start with a square bracket "[" and end with a square bracket "]". At this time, the output must not contain any other content.
2. When the information provided by the user is unclear, incomplete, or incorrect, or the user's question exceeds the capabilities of the provided functions, you need to clearly point out these issues. The following is your strategy:
(1) If the user's instructions include the key details required to call the API, but the type or form of the parameter values does not match the API's definitions, ask in-depth questions to clarify and correct the details. The output format should be: ["There is incorrect value (value) for the parameters (key) in the conversation history."]
(2) If the user's instructions lack the key details required by the API, ask questions to obtain the necessary information. The output format should be: ["Missing necessary parameters (key1, key2, ...) for the api (ApiName)"], replacing key1, key2 with the names of the missing parameters and ApiName with the actual API name.
(3) If the user's request exceeds the current capabilities of your APIs, inform them that you cannot fulfill the request. The output format should be: ["Due to the limitations of the function, I cannot solve this problem."]
Note: The above steps have a priority order. You need to first determine whether scenario (1) applies. If it does, output according to the requirements in (1). Pay attention to distinguishing between scenarios (1) and (2).

{time}

Role Descriptions:
user: User
assistant: The AI assistant role that makes API requests
tool: Provides the results returned from tool calls

API Specifications:
{function}

Figure 26: The inference prompt for Special Data.

The current time is June 11, 2024, 16:00 (Beijing Time). As a simulated mobile assistant agent, you can help users **send text messages, add reminders, and order takeout.**

Text messages
Sending Text Messages (1)Before sending a text message, the agent must first obtain the sender and recipient of the message.(2)When the memory is full and needs to delete messages, you need to ask the user: "Memory is full, which message would you like to delete?"
Viewing Text Messages (1)Before viewing text messages, the agent must first log into the device via login_device().(2)Before viewing text messages, the agent must first obtain the sender and recipient of the messages.(3)After viewing text messages, the agent needs to ask the user if they want to add the message content to a reminder.(4)After viewing text messages, the agent needs to ask the user if they want to reply to the message.(5)If the message content involves takeout, the agent needs to ask if the user wants to order takeout based on the message content.

Reminders
Adding Reminders(1)Before adding a reminder, you should obtain the content and title of the reminder. The reminder time defaults to the current time.(2)If the reminder to be added is the content of a specific message, the agent needs to first view the message content.
Viewing Specific Reminders by Title:After viewing a specific reminder by title, you need to ask the user if they want to complete the tasks within it.

Order takeout
Ordering Takeout(1)Before ordering takeout, the agent needs to obtain the user's takeout platform account and password, and log in using login_food_platform().(2)If the merchant, product, and quantity for the order are not initially provided, you need to ask the user.(3)When encountering takeout from different merchants, you need to order them one by one.(4)If the balance is insufficient, you need to inform the user "Insufficient balance" and ask if they want to change the order.

Function Calls
When a function call is needed, please strictly adhere to the above format requirements:
(1)[ApiName(key1='value1', key2='value2', ...)], Please remember that the function call must start with [and end with]
(2)You need to promptly feedback the task execution status to the user and do not repeatedly call the same function. When you believe the current task is completed, respond with "finish conversation" to end the dialogue.

Figure 27: The inference prompt for Agent Data in a specific scenario.

Table 6: Accuracy evaluation of different models on English Data (%).

Model	Normal						Special	Agent	Overall
	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Summary			
Closed-Source Large Language Models									
GPT-4o	90.0	78.0	68.0	80.0	78.0	82.5	92.7	56.0	**81.1**
GPT-4-Turbo	90.7	80.5	69.0	80.0	88.0	84.2	82.0	62.5	**80.3**
Qwen-Max	88.0	75.0	61.0	74.0	82.0	79.7	74.0	60.0	**75.1**
GPT-4o-Mini	84.3	73.5	59.0	74.0	72.0	76.4	76.7	27.5	**68.9**
Gemini-1.5-Pro	82.3	73.0	61.0	74.0	72.0	75.7	77.3	26.0	**68.5**
Claude-3-5-Sonnet	66.7	64.0	46.0	58.0	68.0	62.2	72.7	44.0	**62.2**
Doubao-Pro-32k	75.3	58.0	52.0	70.0	54.0	66.3	50.7	26.5	**56.0**
Open-Source Large Language Models									
Qwen2.5-Coder-32B-Instruct	86.0	73.5	59.0	76.0	72.0	77.4	80.0	50.0	**73.9**
DeepSeek-V3	88.0	77.5	63.0	76.0	78.0	80.3	72.7	34.0	**71.1**
Qwen2.5-72B-Instruct	81.3	74.5	64.0	76.0	80.0	76.8	74.0	37.5	**70.0**
Llama-3.1-70B-Instruct	83.7	71.5	61.0	74.0	66.0	75.6	29.3	41.0	**57.9**
Qwen2.5-7B-Instruct	70.3	57.0	49.0	62.0	58.0	62.8	49.3	15.0	**51.8**
Qwen2.5-Coder-7B-Instruct	73.3	63.5	52.0	70.0	58.0	66.6	25.3	18.5	**48.1**
DeepSeek-Coder-V2-Lite-Instruct	71.7	58.0	50.0	62.0	60.0	64.0	39.3	2.5	**47.9**
Watt-Tool-8B	84.7	71.5	57.0	70.0	62.0	74.8	2.0	1.5	**44.0**
Hammer2.1-7B	71.3	62.5	43.0	64.0	52.0	62.9	3.3	15.0	**39.6**
Phi-3-Mini-128k-Instruct	66.3	49.0	31.0	58.0	32.0	54.0	12.0	0.0	**34.4**
MLlama-3.1-8B-Instruct	51.0	49.5	28.0	60.0	56.0	48.1	15.3	6.5	**32.9**
xLAM-7B-r	61.7	42.0	32.0	66.0	0.0	48.7	4.0	10.0	**30.8**
Llama-3.2-3B-Instruct	31.7	21.5	9.0	34.0	32.0	26.4	8.7	0.0	**17.6**
Hammer2.1-3B	32.7	14.0	7.0	36.0	32.0	25.5	0.7	1.5	**15.2**

Table 7: Accuracy evaluation of different models on Chinese Data (%).

Model	Normal						Special	Agent	Overall
	Atom	Single-Turn	Multi-Turn	Similar API	Preference	Summary			
Closed-Source Large Language Models									
GPT-4o	96.7	91.0	86.0	90.0	88.0	92.7	93.3	71.5	**89.6**
GPT-4-Turbo	95.7	89.0	86.0	92.0	84.0	91.7	91.3	72.5	**88.6**
Qwen-Max	94.3	86.0	75.0	92.0	84.0	88.7	74.0	68.5	**81.7**
GPT-4o-Mini	88.7	78.5	74.0	80.0	84.0	83.4	81.3	39.0	**76.0**
Claude-3-5-Sonnet	87.0	81.0	79.0	84.0	76.0	83.5	82.0	35.0	**75.6**
Gemini-1.5-Pro	86.7	80.5	68.0	86.0	84.0	82.2	80.0	25.0	**72.8**
Doubao-Pro-32k	84.3	53.0	64.0	82.0	78.0	75.0	59.3	23.5	**62.8**
Open-Source Large Language Models									
Qwen2.5-Coder-32B-Instruct	94.3	88.5	83.0	90.0	90.0	90.8	81.3	71.5	**85.3**
Qwen2.5-72B-Instruct	92.3	86.0	75.0	90.0	82.0	87.3	77.3	52.5	**79.3**
DeepSeek-V3	95.0	90.5	91.0	90.0	88.0	92.6	73.3	35.0	**78.5**
Llama-3.1-70B-Instruct	81.3	65.0	66.0	84.0	70.0	75.3	47.3	43.5	**62.9**
Qwen2.5-7B-Instruct	81.7	63.5	68.0	82.0	76.0	75.9	44.7	12.5	**57.8**
DeepSeek-Coder-V2-Lite-Instruct	78.7	57.5	43.0	82.0	70.0	68.8	41.3	1.5	**51.1**
Qwen2.5-Coder-7B-Instruct	78.7	64.0	63.0	78.0	78.0	73.5	19.3	12.5	**49.6**
Watt-Tool-8B	86.7	67.0	54.0	88.0	66.0	76.3	10.0	4.0	**47.4**
Hammer2.1-7B	76.0	62.5	37.0	60.0	58.0	62.7	26.0	18.5	**46.1**
Llama-3.1-8B-Instruct	52.7	30.0	28.0	72.0	36.0	45.0	26.7	4.0	**33.8**
Phi-3-Mini-128k-Instruct	48.0	29.5	15.0	58.0	32.0	38.9	25.3	1.5	**29.5**
Llama-3.2-3B-Instruct	45.7	9.0	9.0	50.0	32.0	32.7	10.0	0.0	**21.6**
xLAM-7B-r	25.3	2.0	6.0	56.0	0.0	18.7	1.3	7.5	**12.3**
Hammer2.1-3B	12.0	9.0	0.0	44.0	8.0	11.8	1.3	1.5	**7.4**

(question)
I want to understand the symmetry in Escher's Ẅaterfallärtwork. Please provide a detailed symmetry analysis.

(answer)

```
{"Escher_Artwork_Analysis_Tool": {
    "artwork_analysis": [
      {"artwork_id": "Waterfall",
       "analysis_type": "Symmetry Analysis"}]}}
```

(model output)
{"FontAdvisor_recommendDisplayFonts":{.....}}

Figure 28: An error example of Wrong Function Name on Normal Data.

(question)
Can you check if our current automation system with system ID 2435 is compatible with the new device model XR10?

(answer)
{"AutomationSystemCompatibilityCheck":{ "systemID":2435,"newDeviceModel":"XR10"}}

(model output)
{"AutomationSystemCompatibilityCheck":{ "systemID":"2435", "newDeviceModel":"XR10"}}

Figure 29: An error example of Wrong Param Type on Normal Data.

(question)
I am currently working on a research paper about the state of forests, and I need an assessment of the vegetation diversity in the Amazon rainforest. Please provide me with a detailed report.

(answer)

```
{"EcoDiversityMonitor_assessVegetation": {
    "area_identifier": "Amazon rainforest",
    "detailed_report": true}}
```

(model output)

```
[user,area_identifier='Amazonrainforest',detailed_report=True]
```

Figure 31: An error example of Wrong Output Format on Normal Data.

(question)
I would like to assess the GPS accuracy of my vehicle. The device ID is GPS-7524. Sorry, I made a mistake earlier—it should be GPS-7526.

(answer)
{ "SignalAccuracyEvaluator":{ "device_id":"GPS-7526" }}

(model output)
{"SignalAccuracyEvaluator":{ "device_id":"GPS-7524"}}

Figure 30: An error example of Wrong Param Value on Normal Data.

(question)
I‘m considering relocating my business to the Middle East. Can you provide me with a detailed list of major cities there, sorted by internet speed?

(answer)
Missing necessary parameters (sort) for the API (Get_Middle_East_Cities)

(model output)

```
{
  "Get_Middle_East_Cities": {
    "sort": "desc",
    "sort_by": "internet_speed",
    "size": 10,
    "page": 1
  }
}
```

(The question did not mention the sort method)

Figure 32: An error example of Error Detection on Special Data.

(question)
I‘m considering relocating my business to the Middle East. Can you provide me with a detailed list of major cities there, sorted by internet speed?

(answer)
Missing necessary parameters (sort) for the api (Get_Middle_East_Cities)

(model output)
Missing necessary parameters (sort by) for the api (Get_Middle_East_Cities)
(Find missing necessary parameters, but not correctly identified)

Figure 33: An error example of Error Correction on Special Data.