Preview Tools before Using: Enhancing Tool Documentation with Multi-Tool Exploration

Anonymous ACL submission

Abstract

Enhancing the task-solving capabilities of large language models (LLMs) through utilizing tools has garnered increasing attention. To enable LLMs to use tools accurately, developers often provide documentation of the tools in the LLMs' context. However, such documentation has various issues, such as incomplete tool descriptions and insufficient descriptions of parameters or responses. To address this problem, we propose *ToolBFS+*, a method to revise tool documentation by exploring the use of tools. *ToolBFS+* adopts a Breadth-First Search (BFS) strategy to explore various tool usage scenarios and collects the information obtained from the exploration to revise the tool documentation, ultimately improving the model's ability to accurately utilize the tools. Extensive experiments on multiple datasets demonstrate that the *ToolBFS+* method can substantially reduce errors, such as the selection of incorrect tools, and improve the capability of LLMs to use tools accurately[1].

1 Introduction

As large language models (LLMs) become increasingly popular (Achiam et al., 2023; Touvron et al., 2023; Kim et al., 2024), the tool learning task is proposed to further enhance the capabilities of LLMs (Shen et al., 2024; Schick et al., 2024; Cai et al., 2023). These methods enhance LLMs' capabilities by enabling them to utilize external tools for real-world interaction, thereby enhancing their ability to address diverse problems (Zhuang et al., 2024; Qin et al., 2023). To help LLMs understand the tools, developers often provide tool documentation within the model's context (Song et al., 2023; Yang et al., 2024). LLMs rely on their in-context learning capabilities to understand the tools from the context and utilize them accurately. Thus, documentation quality is crucial for enabling the LLMs to effectively utilize tools (Hsieh et al., 2023).

[1] https://anonymous.4open.science/ToolBFS

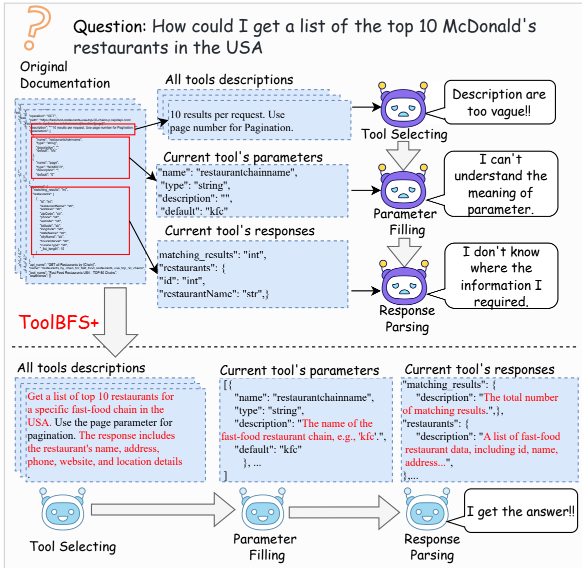

Figure 1: The figure illustrates issues in the original tool documentation, including incomplete tool descriptions, inadequate parameter explanations, and insufficient response details. The documentation can be revised by our *ToolBFS+*to be more comprehensive and accurate for better tool usage.

Typically, when acquiring a tool, the corresponding documentation is provided to facilitate its use (Hsieh et al., 2023). However, this documentation frequently exhibits various issues (Yuan et al., 2024; Qin et al., 2023). As illustrated in Figure 1, the original tool documentation suffers from incomplete descriptions, inadequate parameter explanations, and insufficient response details. These issues can result in errors when the model attempts to utilize the tools.

While manually revising the incomplete documentation could entail substantial costs (Zhuang et al., 2024), some work (Yuan et al., 2024) attempts to have LLMs complete this process by prompting the LLMs to revise the tool documentation. However, solely depending on LLMs to infer the incomplete documentation details without concrete tool-use information (e.g., full tool invocation, including parameters, responses, etc.) may potentially lead to inaccuracies in the documentation. To incorporate concrete tool-use information for a more accurate documentation revision, we propose *ToolBFS+* method that utilizes multi-tool Breadth-First Search (BFS) exploration to enhance the tool documentation. Specifically, our method consists of three stages:

1. Multi-Tool Scenario Generation. Specific tool-use scenarios are crucial for acquiring tool-use information. In this stage, LLMs are prompted to select several functionally related tools, thoroughly understand each tool's functionalities, and subsequently generate a multi-tool scenario for further exploration in the next stage. Multi-tool scenarios refer to specific problems that require the collaboration of multiple tools to solve. These scenarios better mimic the complex demands of the real world.

2. Multi-Tool BFS Exploration. Tool-use information can be plentiful, but some of it may be meaningless or low-quality, providing no valuable information (e.g., empty tool responses caused by incorrect tool parameters). We consider the solutions of the scenarios as tool-use information. To obtain high-quality solutions in generated scenarios, we design the Tool-BFS algorithm to search for solutions through exploration. By viewing the exploration process as a graph search, considering each tool invocation as a node and starting from the root node with no tool invocations, we use a BFS strategy to iteratively search through all possible paths until the correct answer is found. We consider the path from the root node to the node containing the correct answer as a high-quality solution. The BFS strategy ensures both the correctness and efficiency of the solution.

3. Tool Documentation Revision. To incorporate concrete tool-use information from the solutions into the tool documentation, we consider the following two aspects: (1) Using a single node with complete tool invocation, including parameters, responses, etc, to revise the corresponding tool's documentation, filling in the incomplete or unclear content. (2) Viewing the tool selections in the solutions as an experience that can guide which tool to select in specific scenarios. We incorporate this experience into the corresponding tool documentation.

We conduct extensive experiments on RestBench (Song et al., 2023) and ToolBench (Qin et al., 2023). The results show substantial improvements using our proposed documentation enhancement method. These improvements are visible across various tool-use methods and datasets, validating the effectiveness of our approach.

Our main contributions are as follows:

1. We propose *ToolBFS+*, a multi-stage tool documentation enhancement method that can revise the documentation for more accurate tool utilization of LLMs.
2. We design the Tool-BFS algorithm, which allows for a thorough exploration of the scenarios and the discovery of high-quality solutions in a BFS strategy.
3. We conduct extensive experiments on three datasets and show the effectiveness of our *ToolBFS+* method in improving the quality of tool documentation and tool-use ability of LLMs.

2 Related work

Tool-augmented language models. Enhancing the capabilities of LLMs through external tools has been proven effective (Schick et al., 2024; Jin et al., 2024; Li et al., 2023; Patil et al., 2023). Some methods enhance a model's tool-use ability through fine-tuning on specific datasets, involving dataset construction and a training process (Schick et al., 2024; Gao et al., 2024; Tang et al., 2023; Wang et al., 2024, 2022). However, these methods require additional training data, can only be applied to open-source models, and involve high fine-tuning costs (Qin et al., 2023).

Consequently, some researchers aim to optimize the process of utilizing tools to enhance the model's ability to use them. These methods leverage the in-context learning abilities of LLMs by incorporating tool documentation and demonstrations into the context. By designing specific workflows, they enable the model to use tools to accomplish tasks (Yao et al., 2022; Lu et al., 2024; Song et al., 2023). Although this approach eliminates the need for training, it is susceptible to the quality of tool demonstrations and documentation (Yuan et al., 2024; Shi et al., 2023). To address this problem, we propose an approach that enhances the quality of tool documentation, thereby improving the model's ability to use tools.

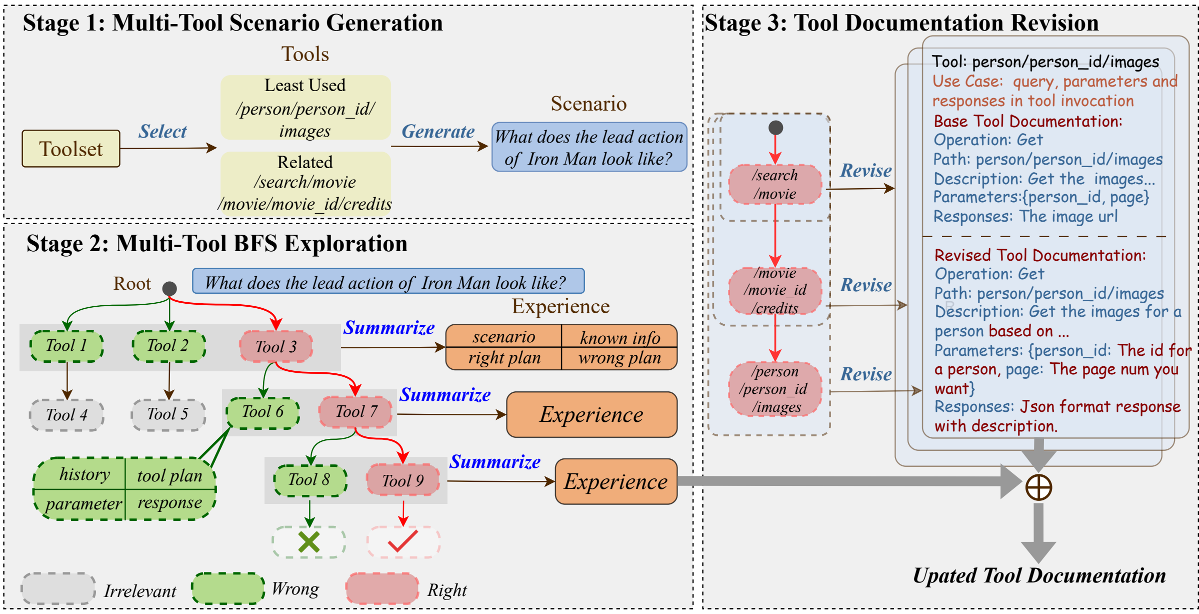

Figure 2: The overall process of our *ToolBFS+* consists of three stages: (1) Multi-Tool Scenario Generation, where functionally related tools are selected and multi-tool scenarios are generated to mimic real-world scenarios; (2) Multi-Tool BFS Exploration, employing a BFS strategy to explore and find high-quality solutions; and (3) Tool Documentation Revision, using solutions from the exploration to revise and enhance tool documentation, improves accuracy of tool usage.

3 Methodology

3.1 Method Framework

In this section, we detail the *ToolBFS+*, enhancing the tool documentation by multi-tool BFS exploration. As shown in Figure 2, our approach consists of three main stages: (1) Multi-Tool Scenario Generation, (2) Multi-Tool BFS Exploration, and (3) Tool Documentation Revision. In the Multi-Tool Scenario Generation stage, we select several functionally related tools from the toolset T and generate a tool-use scenario for these tools. The scenario is then passed to the Multi-Tool BFS Exploration stage for further exploration to get the tool invocation path as the solution. In the Tool Documentation Revision stage, we will revise the tool documentation using the information obtained from the exploration stage.

3.2 Multi-Tool Scenarios Generation

This stage is designed to generate multi-tool scenarios on the toolset $T = \{t_1, t_2, \ldots, t_n\}$. The stage includes two steps: tool selection and scenario generation. During the tool selection step, we initially allow the LLMs to select functionally related tools, but we found that LLMs tend to prefer simpler tools (e.g., *search/movie*, which has simple tool description and require fewer parameters) and overlook more complex ones (e.g., */discover/tv*, which involve multiple parameters and difficult to use). To avoid this, we record the frequency of tool selection and ensure that the least-used tool is selected each time. This is achieved by incorporating the least-used tool into the LLMs' output during the selection step. Subsequently, the LLMs select functionally related tools based on the least-used tool. Then, we prompt LLMs to generate a multi-tool scenario, which involves the coordinated use of selected tools. For example, as shown in stage 1 in Figure 2, the LLMs are provided the least-used tool, */person/person_id/images*. They then select related tools like */search/movie* and */movie/-movie_id/credits*, and generate the scenario: "What does the lead actor of Iron Man look like?", which is passed to the next stage for further exploration.

3.3 Multi-Tool BFS Exploration

After scenario generation, we aim to explore the generated scenarios to obtain high-quality solutions that provide the correct answer and meaningful tool invocations. We also summarize the experience regarding tool selection from the solutions.

Tool-BFS Algorithm. To get solutions, we specially design the Tool-BFS Algorithm. As shown in stage 2 in Figure 2, we prompt the model to explore solutions using a BFS strategy. Specifically, by viewing the exploration process as a graph search, we consider each tool invocation as a node and start from the root node initially containing no tool invocations. Each time, we select a node from the current level to explore downwards until all nodes at the current level have been selected. Then, we proceed to the next level to continue the exploration. Every node records the invoked tools history, the currently invoked tools, parameters, and response. To make the child nodes more diverse and expand the search space, we explicitly inform the model about the next tool plans in the nodes it has generated and encourage it to generate different plans. When LLM give "Final Answer" in the tool plan, we consider that the LLMs have either found the answer or can no longer proceed with the tools. Then, we prompt LLMs to judge whether the current solution and answer are correct by using predefined rules. The rules are manually written and specific to the dataset e.g., "the model should return a reasonable answer"; "fabricating specific parameters during reasoning is not allowed". The algorithm stops when the correct answer is found or the maximum depth limit is reached. We consider the path from the root node to the correct node as the solution. The pseudo-code of the overall algorithm is given in Appendix A.3.

After completing the BFS exploration and finding a solution, we categorize the nodes in the BFS tree into three types: "*Right*", "*Wrong*", and "*Irrelevant*". These respectively refer to the nodes along the solution, the nodes along the wrong paths, and the nodes that do not intersect with the solution. We do not take any action on the *Irrelevant* nodes. Subsequently, we extract the *Right* nodes for later documentation revision.

Experience Summary. The tool selection in the solution can be considered as an experience that guides which tool to select in specific scenarios. The "*Wrong*" nodes also contain meaningful information as they store the mistakes the model tends to make. Therefore, we explicitly store this guidance and mistake as a type of experience within the corresponding tool documentation. This kind of experience consists of four parts: the scenario, known information, the right plan, and the wrong plan. We summarize this experience at the locations where the model generates "*Wrong*" nodes and "*Right*" nodes, indicated by gray blocks in Figure 2. For example, *tool 6* and *tool 7* are summarized as one experience. This experience includes the current scenario "*What does the lead action of Iron Man look like*", the known information (i.e., the execution result of *tool 3*), the right plan (i.e., *tool 7*), and the wrong plan (i.e., *tool 6*). It will be integrated into *Tool 3*'s documentation in next stage.

3.4 Tool Documentation Revision

After generating and exploring different scenarios, we aim to incorporate concrete tool-use information from the solution into the tool documentation in two aspects: (1) using the complete tool invocation from the solution to revise the corresponding tool documentation, filling in the incomplete or unclear content. (2) integrating the experience summarized from the previous stage into corresponding tool documentation as guidance for tool selection.

For the first aspect, we revise the documentation using nodes at three key points: tool's functionality description, parameters, and responses. For the tool functionality description section, we use the scenarios, required parameters and execution results to revise the tool's functionality description. By doing so, we provide a clear understanding of when and how to use the tool. For the tool parameters section, we utilize specific parameters and responses from tool invocations to clarify the specific functions of each parameter. This helps in providing concrete, clear explanations of how each parameter affects the tool's behavior. For the tool response section, we leverage actual responses from tool invocations to explain the structure and details of the responses. This ensures that LLMs can comprehend the output format and content accurately. By integrating these elements into the LLMs' context, we can revise the documentation more effectively, making it clearer and more comprehensive for LLMs.

For the integration of experience, we designed a mechanism to avoid having too many redundant experiences. This mechanism checks if a new experience already exists in the tool experience set when adding new experiences. Specifically, we prompt the LLMs to check whether the current experience to be added matches or is similar to any experience in the tool's experience set based on known information and scenario. For the use of experience, for example, in Figure 2, when other tool-use methods use *tool3*, we expose the experience set of *tool3* to the LLMs, allowing them to determine if there is any experience that fits the current scenario and known information. If so, the experience is incorporated into the context to guide the selection of tools.

4 Experiment Setup

4.1 Datasets and Evaluation Metrics

Datasets. We conduct experiments on below datasets: (1) RestBench (Song et al., 2023): A benchmark with two sub-datasets: the TMDB dataset, containing 57 real APIs related to movies and actors, and the Spotify dataset, comprising 40 real APIs for operations such as retrieving and playing songs. (2) ToolBench (Qin et al., 2023): A benchmark has a wide range of user requests and many Rapid APIs, with 49 categories available through the RapidAPI hub. We filter a high-quality dataset from Toolbench's I2 category Food subset and named it Toolbench-Food. This dataset is designed to match the format and size of RestBench and contains information related to food recipes and more. We detail the specific dataset construction in Appendix C.

Evaluation Metrics. In evaluating the tool-use methods performance on two RestBench datasets, we use two evaluation metrics following (Song et al., 2023): (1) Correct Path Rate (CP%), which considers a tool call path as correct if the path of the golden answer is a subpath of the model-generated path, and (2) Success Rate (Success%), which assesses whether the model accurately completes the query by human evaluation. For the ToolBench-Food dataset, we follow ToolBench's evaluation metrics. Include (1) Pass Rate (Pass), the proportion of successful instructions completed within a limited budget, and (2) Win Rate (Win), ChatGPT's preference between two solutions (Qin et al., 2023).

4.2 Baselines

We conduct extensive experiments across various tool-use methods to demonstrate the effectiveness of our *ToolBFS+* method. These tool-use methods include: (1) React (Yao et al., 2022), a method that utilizes a chain-of-thought approach within the Thought-Action-Observe framework. (2) Reflect (Gou et al., 2023), which employs a feedback and self-correction mechanism based on tool responses. (3) Chameleon (Lu et al., 2024), a method that directly generates multi-step plans for tool usage and then sequentially executes the plan. (4) RestGPT (Song et al., 2023), which adopts a multi-agent collaboration strategy integrating roles such as planner, selector, caller, and parser. These diverse frameworks allow us to comprehensively evaluate the effectiveness of our proposed method.

For the documentation used in the above tool-use methods, we adopt three settings. (1) Original: Using the documentation provided in the benchmark, which may have various issues. (2) EasyTool: Using documentation that has been improved by EasyTool(Yuan et al., 2024). (3) Ours: Using our tool documentation enhanced by Multi-Tool BFS Exploration.

4.3 Implementation Details

We employ OpenAI's *gpt-3.5-turbo*[2] as the backbone to implement our proposed method and all tool-use methods. Additionally, we conduct experiments on the open-source model Mixtral-8x7B (Jiang et al., 2024) to validate our method's performance on open-source models.

In the Multi-Tool BFS Exploration stage (§ 3.3), we set the width to 3 and the maximum depth to 8. For the tool-use method in exploration in § 3.3, we employ a three-stage process, planning, calling, and parsing, to complete a tool invocation. To determine whether the solution and the answer are correct, we design a set of rules and prompt the model to judge their correctness. We provide the specific details of the judgment in Appendix A.4

5 Result and Analysis

5.1 Main Result

Overall Performance. Table 1 presents our experimental results. Our *ToolBFS+* method enhanced documentation achieves the best performance in all tool-use methods on three datasets. Specifically, on the RestBench-TMDB dataset, our enhanced tool documentation with the React method achieves 64.0 in Correct Path Rate metric and 62.0 in Success Rate metric. This substantially improves performance compared to the Original and EasyTool baselines and approach the performance of RestGPT on the original documentation. The similar improvement is observed on the RestBench-Spotify and ToolBench-Food datasets. The reason is that our multi-tool exploration and documentation revision method improves the quality of the tool documentation by incorporating concrete tool usage information. This improvement is manifested in more comprehensive tool descriptions and more detailed parameter explanations, etc. The enhanced tool documentation provides more information and improves performance for the four tool-use methods mentioned above.

[2] https://openai.com/chatgpt

Method	Docs Type	RestBench-TMDB		RestBench-Spotify		ToolBench-Food	
		CP%	Success%	CP%	Success%	Pass	Win
React (Yao et al., 2022)	Original	56.0	56.0	52.6	43.6	51.85	-
	EasyTool	60.0	59.0	-	-	59.25	53.66
	Ours	64.0	62.0	57.8	54.4	68.52	62.50
Reflect (Gou et al., 2023)	Original	55.0	53.0	50.9	49.1	48.14	47.50
	EasyTool	58.0	56.0	-	-	64.81	59.32
	Ours	60.0	60.0	56.1	52.6	66.67	61.02
Chameleon (Lu et al., 2024)	Original	64.0	63.0	56.1	54.4	46.30	46.88
	EasyTool	69.0	69.0	-	-	59.25	54.29
	Ours	72.0	71.0	61.4	59.6	61.11	56.25
RestGPT (Song et al., 2023)	Original	65.0	63.0	71.9	68.4	62.96	60.34
	EasyTool	74.0	70.0	-	-	64.81	62.06
	Ours	**76.0**	**73.0**	**77.2**	**70.1**	**70.37**	**65.51**

Table 1: The results on three datasets. The CP% represents the Correct Path Rate metric, while the Success% indicates the Success Rate metric. The Pass metric denotes the proportion of successful queries completed. Win is calculated by comparing each model's output to React Original. The best result for each tool-use method is underlined, and the best overall result across all methods is in **bold**.

Method	RestBench-TMDB		RestBench-Spotify	
	CP%	Success%	CP%	Success%
Mixtral-8x7B				
React Original	51.0	40.0	38.6	35.1
React Ours	54.0	46.0	45.6	40.4
RestGPT Original	62.0	49.0	61.4	59.7
RestGPT Ours	70.0	52.0	66.7	63.2

Table 2: The results of using Mixtral-8x7B as the backbone with different tool-use methods. We report **CP%** and **Success%**.

Document Type	RestBench-TMDB		RestBench-Spotify	
	CP%	Success%	CP%	Success%
React				
Original	56.0	56.0	52.6	43.6
Ours	64.0	62.0	57.8	54.4
-*w/o multi-scenario*	61.0	60.0	54.4	50.9
-*w/o exp*	63.0	61.0	54.4	52.6
-*w/o doc*	61.0	61.0	56.1	56.1
RestGPT				
Original	65.0	63.0	71.9	68.4
Ours	73.0	70.0	77.2	70.1
-*w/o multi-scenario*	71.0	67.0	71.9	64.9
-*w/o exp*	72.0	69.0	73.7	66.7
-*w/o doc*	69.0	68.0	75.4	68.4

Table 3: The results of Ablation study on the RestBench-Spotify and RestBench-TMDB datasets.

Both the EasyTool method and our method yield substantial improvements compared to using the original documentation. Compared to EasyTool's documentation, our approach performs better in every experiment setting. This is because, compared to EasyTool, our approach provides more concrete tool-use information, including parameters and concrete examples of responses, when revising documentation. This information can better assist LLMs in revising tool documentation.

Performance with the Open-Source LLMs. Following the above experiment settings, we alternate our backbone LLMs with the open-source model Mistral-8x7B (Jiang et al., 2024) and further validate the effectiveness of our approach. As shown in Table 2, under the RestBench-TMDB setting, our method increases the Success Rate from 40% to 46% using the React method and from 49% to 52% using the RestGPT method, demonstrating the effectiveness with the open-source model.

5.2 Ablation Study

To further illustrate the impact of different components in our method, we conduct the following experiments:

-*w/o multi-scenario*: We replace the multi-tool scenario in § 3.2 with a single-tool scenario (e.g., */search/movie*, search the movie Iron Man, only one tool is needed), then explore the single-tool scenario and revise the documentation.

-*w/o exp*. We remove the experience in § 3.4 Tool Documentation Revision to test if it affects tool selection.

-*w/o doc*: We implement this by retaining the experience but replacing the enhanced tool documentation with the original tool documentation in § 3.4.

We conduct ablation experiments on RestBench's two datasets using the React and RestGPT methods with the ablated documentation. Our full method still retains the best performance on two metrics. The *-w/o multi-scenario* setting significantly reduced effectiveness across datasets and methods. This indicates that exploring multi-tool scenarios is crucial for enhancing tool documentation, as the information obtained from single-tool scenarios is insufficient to revise tool documentation.

On the TMDB dataset, the *-w/o doc* setting causes the most significant performance decline, while on the Spotify dataset, the *-w/o exp* setting causes the most significant performance decline. This is because the original TMDB dataset has relatively limited documentation content, with more incomplete tool descriptions and insufficient explanations for parameters and responses. However, the quality of the Spotify documentation is relatively high, and experience guidance plays a major role in documentation enhancement.

5.3 Case Study

We conduct a comprehensive case study and find that our enhanced documentation allows the model to better understand the tools, enabling it to select correct tools, fill in the correct parameters, and parse required information from the responses. We also provide examples to illustrate the difference between our enhanced documentation and original documentation in tool-use methods performance. More details can be found in Appendix B.

6 Discussion

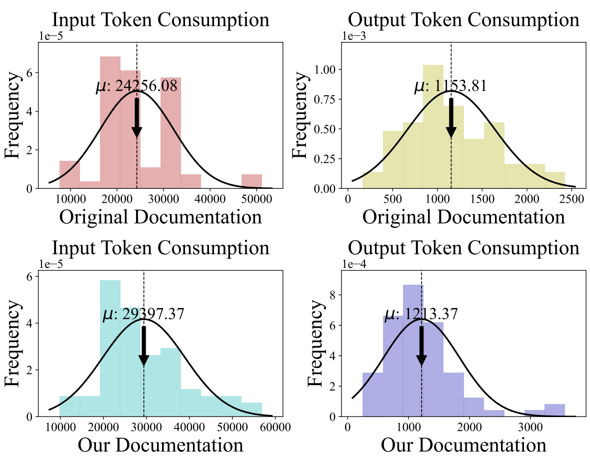

Figure 3: The efficiency analysis of the RestGPT method on different documentation, where we count the distribution of input and output token consumption and the average consumption μ.

Efficiency analysis. Due to the intensive inference costs associated with tool-use methods (Song et al., 2023), we further explore whether our documentation method would result in efficiency reductions. Using the same settings as Table 1, we compare the token consumption between using the original documentation and our enhanced documentation on the RestBench-TMDB dataset.

In Figure 3, we present histograms showing the frequency and mean value μ of token consumption for input and output tokens using different documentation. Notably, output token calculations are typically more complex and time-consuming than input token calculations (Vaswani et al., 2017). Our findings indicate that the number of output tokens remains almost consistent with the original documentation, and there was no significant increase in input token consumption. This suggests that our method achieves significant performance improvements while maintaining a similar token consumption as the original documentation.

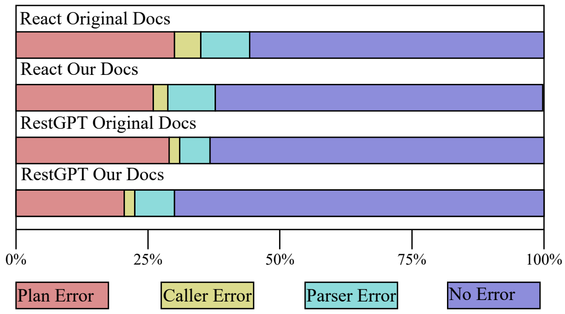

Figure 4: Error statistics in React and RestGPT with different documentation.

Error analysis. We conduct an error analysis experiment to further analyze how our enhanced documentation improves the model's performance. This experiment, conducted under the experimental settings of Table 1, checks the distribution of different errors in React and RestGPT by using different documentation. In Figure 4, errors are categorized into Plan Error, Caller Error, and Parser Error. Specifically, Planning Errors denote the selection of an incorrect tool, Caller Errors involve the use of wrong parameters, and Parser Errors arise when the required information is not parsed from the tool responses.

Figure 4 illustrates that our method substantially reduces the Planning Error compared to using the original documentation. This improvement stems from the revised tool descriptions and guidance of experience. Furthermore, improvements are also

Method	Correct	ΔLen	Tool-Calling Success
React	74.47%	2.36	86.99%
DFSDT	87.23%	2.42	87.80%
Tool-BFS	89.36%	2.19	95.65%

Table 4: The solution analysis of three different search strategies on the ResetBench-TMDB dataset.

Document Type	Selector		Caller		Parser	
	GPT4	Human	GPT4	Human	GPT4	Human
RestBench-Tmdb						
Original	1.70	1.72	2.04	2.03	2.16	2.16
Ours	2.00	1.92	2.13	2.11	2.59	2.51
RestBench-Spotify						
Original	1.70	1.65	1.58	1.49	1.82	1.75
Ours	1.92	2.04	1.68	1.59	2.25	2.14

Table 5: The direct evaluation results of documentation quality on the RestBench-TMDB and RestBench-Spotify datasets.

observed in correctly filling parameters and parsing responses, which benefits from more clear and comprehensive parameter and response descriptions.

Analysis of the BFS strategy. To demonstrate the superiority of the BFS approach in finding high-quality solutions, we compare three different search strategies: React, DFSDT, and Tool-BFS. React (Yao et al., 2022) is a tool-use method that follows the Thought-Action-Observation framework. DFSDT (Qin et al., 2023) enhances LLMs with the Depth First Search-based Decision Tree (DFSDT) to select tools to solve tasks. We conduct experiments under the generated multi-tool scenarios of RestBench-TMDB. To evaluate the quality of the solutions, we design three metrics: **Correct**, **Δ Len**, and **Tool-Calling Success**, which respectively represent the percentage of finding the correct answer (i.e., the method mentioned in § 3.2 that prompts LLMs to determine the correctness of the current answer), the average length of solutions, and the rate of successful tool usage in the solutions. This evaluation is grounded in the simple intuition that high-quality solutions should ensure correctness and efficiency.

The results in Table 4 demonstrate that React underperforms compared to search-based methods in terms of **Correct**. Both DFSDT and Tool-BFS methods show similar performance for **Correct**. However, the Tool-BFS method greatly surpasses the DFS method regarding **ΔLen** and **Tool-Calling Success**, validating the efficiency of our approach in utilizing tools to seek high-quality solutions.

Fine-grained analysis of quality of documentation. Current evaluation metrics lack a direct assessment of tool documentation quality, i.e., evaluating the tool documentation itself rather than indirectly evaluating the performance over other tasks. To address this, we design a direct evaluation method for documentation quality.

Existing tool-use methods utilize tool documentation mainly in three stages: planning, invocation, and parsing (Yao et al., 2022; Song et al., 2023; Qin et al., 2023). We assess tool documentation quality across these three stages: Planning: The model selects a tool based on tool descriptions and user queries. Therefore, we rate documentation based on the comprehensiveness and conciseness of the tool functional description. Invocation: The model fills in parameters based on tool's parameter section. Hence, We rate documentation based on the completeness and clarity of parameter descriptions. Parsing: The model parses the required execution results based on the response description in the tool documentation. Thus, we rate documentation based on structural accuracy (e.g. standard and complete json format) and completeness of response descriptions. For accuracy, we use both GPT-4 and human evaluations.

We use a three-point scale for the rating with detailed rules provided in Appendix E. As illustrated in Figure 5, both sets of evaluations indicate a consistent preference for our enhanced documentation across all three metrics, demonstrating the effectiveness of our approach in improving quality of tool documentation.

7 Conclusion

In this work, we propose a method called *ToolBFS+*, which improves the quality of tool documentation by integrating specific tool usage information obtained through multi-tool BFS exploration, thereby improving the performance of different tool-use methods. Our method consists of three stages: (1) Multi-Tool Scenario Generation, (2) Multi-Tool BFS Exploration, and (3) Tool Documentation Revision. By conducting BFS exploration on the generated multi-tool scenarios, we obtain high-quality solutions. We then integrate the concrete tool usage information from these solutions into the corresponding tool documentation, thereby enhancing the tool documentation. Extensive experiments on various tool-use methods and datasets demonstrate the superiority of our *ToolBFS+* method.

Limitations

Our current implementation of Multi-Tool BFS Exploration, based on LLMs, as discussed in § 3.3, may encounter efficiency challenges. Specifically, the process can involve exploring irrelevant nodes to arrive at high-quality solution. However, it's important to note that this documentation is intended for one-time generation with no subsequent modifications. In the future, we plan to integrate heuristic path search methods to enhance the efficiency of our exploration process.

Ethics Statement

This paper proposes a method to enhance tool documentation by addressing existing issues in the tool documentation to improve the model's ability to use the tools. All the tools used in our experiments are provided by open-source platforms, including TMDB, Spotify, and Rapid API.

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 arXiv:2303.08774*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 18030–18038.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. 2023. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Qiao Jin, Yifan Yang, Qingyu Chen, and Zhiyong Lu. 2024. Genegpt: Augmenting large language models with domain tools for improved access to biomedical information. *Bioinformatics*, 40(2):btae075.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36.

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. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2024. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36.

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

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*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, pages 31210–31227. PMLR.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. 2023. Restgpt: Connecting large language models with real-world applications via restful apis. *arXiv preprint arXiv:2306.06624*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. Llms in the imaginarium: Tool learning through simulated trial and error. *Preprint*, arXiv:2403.04746.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2024. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36.

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

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

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2024. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36.

A ToolBFS+ Method Details

A.1 Tool Selection and Scenario Generation Prompt

Table 6 presents the prompt we used to select the tool and generate the scenario. At the end of the prompt is the least-used tool we select for it.

A.2 Node Cases and Experience Cases

Table 7 presents a node example in the Tool-BFS algorithm. Table 8 presents an experience example gained from the exploration stage.

A.3 Tool-BFS Algorithm

Algorithm 1 is the pseudo-code of our Tool-BFS Algorithm.

A.4 Judgment Prompt

During our Multi-Tool BFS exploration phase, to determine whether the currently obtained answer is correct without a ground truth answer, we prompt the model to judge the correctness of the current answer by pre-defining rules. The specific judgment prompt is shown in Table 9.

Using LLMs to judge the correctness of the answer has been widely adopted in tool learning tasks (Qin et al., 2023). Due to the complexity of tool learning tasks, there is often a lack of labels to serve as evaluation standards. Therefore, many works employ LLMs to determine the correctness of the answers (Qin et al., 2023; Lu et al., 2024). Besides, data with reasonable answers is relatively easy to collect. For instance, during user interactions, the model can provide answers using different tools, and users can give positive or negative feedback. By gathering this type of data with positive feedback, we can eliminate the need for scenario generation in Multi-Tool Scenario Generation stage and judge the correctness of the answers found in the Multi-Tool BFS Exploration stage based on user-satisfactory answers.

A.5 Revising Prompt

We present three main types of prompts utilized in stage 3, which are for revising tool descriptions Table 10, revising tool parameters Table 11, and revising tool responses Table 12.

B Case Study

We present examples of using different documentation-based React methods on RestBench-TMDB in Table 13, and examples of using different documentation-based RestGPT methods on RestBench-Spotify in Table 14, 15. In the React examples, the model accurately selects the tool during the Action 2 stage due to our more precise tool description. Similarly, in the RestGPT examples, the model's caller module correctly fills in the parameters because of our more detailed parameter description

C Details of ToolBench-Food

C.1 Dataset Building Process

ToolBench (Qin et al., 2023) is a benchmark that encompasses a wide range of user requests and numerous REST APIs, offering 49 categories available through the RapidAPI hub. However, many requests in ToolBench are unachievable, and many of the APIs are of low quality, making it impossible to function properly (Qin et al., 2023). To obtain a high-quality dataset for validating our method, we select a high-quality dataset from Toolbench's I2 category Food subset and named it Toolbench-Food, which is similar in size to RestBench.

To ensure that each request is achievable, we filter out the data that provided the correct answers by checking the DFSDT output for each request. DFSDT is a tool usage scheme based on the Depth-First Search (DFS) proposed in (Qin et al., 2023). We then extract the tools used by these requests to compile a toolset. In the end, we filter out 54 requests and 41 tools. Table 16 shows several examples of the dataset.

D Details of RestGPT

D.1 Success Rate Evaluation

For the evaluation of the Success Rate, we adopt the human evaluation method following RestGPT (Song et al., 2023) to assess whether the model has fulfilled the user query. We recruit three graduate students from related fields to evaluate the Success Rate.

E Fine-Grained Analysis of Tool Documentation Detail

We recruit three graduate students from related fields to evaluate the quality of the tool documentation. The prompts used for evaluation were the same for both GPT-4 and human evaluators. Table 17 18 19 show the prompts we used.

Algorithm 1: Tool-BFS Algorithm

```
Input: Scenario S, Tool set T, Executor E, Model M, Maximum depth max_depth, Width
       width
Output: Correct solution path or None
queue ← Queue();
initial_node ← Node(depth=0, history=[], plan=[], response=[]);
queue.push(initial_node);
while not queue.empty() and queue.depth ≤ max_depth do
    current_node ← queue.pop();
    if is_api_call(current_node.plan) then
        current_node.response ← execute_api_call(current_node.plan, E, M);
    else if is_final_answer(current_node.plan) then
        if is_correct_answer(current_node.plan, S, M) then
            return;
    next_nodes ← [];
    for i ← 1 to width do
        plan ← generate_next_plan(T, next_nodes, M);
        if plan in [node.plan for node in next_nodes] then
            continue;
        new_node ← Node(depth=current_node.depth + 1, history=current_node.history +
          current_node, plan=plan, response=[]);
        next_nodes.push(new_node);
        queue.push(new_node);
return None;
```

```
You are an API tester whose job is to design usage scenarios for existing API
libraries to test their specific functionalities.
I will provide all the API information. Each API includes its operations, paths, and
 basic descriptions.
When constructing a usage scenario, you need to first select APIs and then build the
 scenario based on the chosen APIs.
Please note that each time you must select between two and four APIs, and the first
API is mandatory.
For the scenarios you build, please avoid using unknown information such as "this
movie" "the music" or 'artist' Instead, replace them with specific movies or music
titles, such as "Inception", "The Eminem Show" or "Eminem".
The scenario you constructed should be relatively complex  and it requires the use
of at least two API tools to address the task.
You cannot generate scenarios that cannot be solved by the API I provide.
Make sure your question is a solvable one, and I won't add any additional
information to your question
------
Here are some examples:
{icl_examples}
------
Here are the APIs information:
General description of the API toolset: {description}
APIs:
{apis}
The API 1 must be selected. When selecting other APIs, prioritize choosing ones that
 are related to API 1.
Every API you select must be used in the scenarios you have built.
When you choose an API, try to choose APIs that are related to each other.
For example, an API requires the return result of another API as input.
Some APIs may be independent. In this case, you only need to use this API to build
usage scenarios. You don't need to select another API.
Starting below, you must follow this format:
Selected APIs: The selected APIs information
Operation: The operation of the first API
Path: The path of the first API, it must be the API 1 in the above APIs, the content
 of API 1 should be identical to the one listed above. Please do not fabricate any
information.
Description: The description of the first API
Operation: The operation of the second API
Path: The path of the second API, it could be any API in the above APIs
Description: The description of the second API
...up to 4 APIs
Constructed scenarios: The constructed scenarios, just generate one scenario
Now begin your mission!
-------
Selected APIs:
Operation: {operation}
Path: {path}
Description: {api_description}
```

Table 6: Tool Select and Scenario Generation Prompts in Our Method

```
{
    "planner_history": [
     [
        "Search for the movie "The Dark Knight".",
        "GET /search/movie?query=The Dark Knight",
        "Successfully called GET /search/movie?query=The Dark Knight to search for
        the movie "The Dark Knight". The total number of results is 31 spread across
         2 pages."
     ],
    ],
    "depth": 2,
    "plan": {
        "plan": "Get the credits for the movie "The Dark Knight" to find the
        director.",
        "api_plan": "GET /movie/155/credits"
    },
    "api_calling": {
        "Operation: GET,
        "Input": {
            "URL": "https://api.themoviedb.org/3/movie/155/credits",
            "parameters": None
            "description": "Get the cast and crew details for the movie 'The Dark
            Knight.'",
            "output_instructions": "What is the name of the director of the movie?"
        }
    },
    "response": {
        "id": 155,
        "cast": [
             {
                "adult": false,
                "gender": 2,
                "id": 3894,
                "known_for_department": "Acting",
                "name": "Christian Bale",
                "original_name": "Christian Bale",
                "popularity": 39.477,
                "profile_path": "/v2Oks7DTbZcKzSi2Pw58C7SSLzM.jpg",
                "cast_id": 35,
                "character": "Bruce Wayne / Batman",
                "credit_id": "52fe4220c3a36847f8005d17",
                "order": 0
             }...
    "execution_res": "Successfully called GET /movie/155/credits to get the cast and
     crew details for the movie 'The Dark Knight'. The name of the director of the
    movie is Christopher Nolan."
}
```

Table 7: BFS Tree Node Example in RestBench-TMDB Dataset

```
{
    "scenario": "Get the genres of the most popular movie currently.",
    "known_information": "The ID of the most popular movie currently.",
    "right_plan": "I have the ID of the most popular movie currently, so I need to
    get the genres of the movie.",
    "right_api_plan": "Use the GET /movie/{movie_id} method to get the genres of the
     movie.",
    "wrong_plan": "I have the ID of the most popular movie currently, now I need to
    know the details of the movie.",
    "wrong_api_plan": "Use the GET /movie/{movie_id}/credits method to get the
    details of the movie."
}
```

Table 8: Experience Example in RestBench-TMDB Datasets

```
You are an evaluator tasked with determining whether the reasoning process I
provided has been completed successfully. If it has, please explain why and then
output 'yes', otherwise, explain why and then output 'the answer is "no"'
Please note that if a task is successfully completed, its reasoning process must be
correct, and the final result should be formatted as follows:
Final Answer: The final answer of the query is "..."  (It indicates that a task has
been successfully completed.)


Here are some examples:
{icl_examples}

API information:
{api_info}

Judgement Rules:

1: In the API calling step, placeholders "{{}}" should not be present. They need to
be formatted with specific content.
2: For the ID information in the inference path, it must be obtained or referenced
from the API response, rather than being fabricated by the model itself.For example,
 if the model uses GET /albums/{{albums_id}}/tracks, the albums_id  must be obtained
 from the API response, such as GET /search
3: If the question is about a specific piece of information and one of the available
 APIs has an API for that specific piece of information, then the specific API
should be used to solve the problem rather than the more general API. For example,
if the query is about the tracks of a albums, the model should use the API GET /
albums/{{albums_id}}/tracks to get the tracks of the album, rather than using /
search API or /albums/{{albums_id}}.
4: In judging, you need to consider the correctness of the inference process and the
 final answer.
5: The order of execution of the api should be in the order indicated.
6: Additional unrelated actions are not allowed

No matter how the model explain its reasoning steps in the inference stage, the
above rules are must be followed.


You must follow the format below:
Query: The query the model is trying to answer.
Plan step 1: The first step of the reasoning process.
API calling 1: The API calling in the first step.
...
Final Answer: The final answer of the query is "..."
Judgement: The judgement of the reasoning process. Firstly explain why the reasoning
 process is correct or incorrect, and then output "yes" or "no".

Begin !!!
Query: {query}
{planner_history}
{plan}
Judgement:
```

Table 9: Prompt for Tool Selection and Scenario Generation

```
You are an API documenter responsible for writing a series of functional
descriptions for RESTful APIs to assist users in querying the relevant APIs.
I will provide you with a description of the existing api but this description may
be incomplete or noisy, please rewrite the description of the current api based on
some of the scenarios and call examples I have provided you with as well as the
original api information
Please update the current description of the API based on the given information to
better assist users in querying the API.
Note that you should keep the description as informative as possible while reducing
the text length.
Please briefly explain what parameters the api requires and what to include in the
response, as well as the considerations for the use of the api.
The original document may contain some irrelevant or redundant information, so pay
attention to filtering
You only need to add what you think is important to the document, not explain the
response completely.
Each rewritten document should be no more than 50 words.
If you think the original information is accurate enough, then you don't need to
change it, just output the original conten


Here are some examples:
{icl_examples}
If you think the original description is accurate enough, just repeat the original
description.

Starting below, you must follow this format:
Operation: The operation of the API
Path: The path of the API
Description: The original description of the API
Parameters: The parameters of the API, maybe None
Responses: The responses of the API, maybe None
Scenario Examples: The scenario examples of the API
Rewritten Description: Your rewritten description of the API

Now begin your mission!
{api_doc}
Scenario Examples: {scenario}
Rewritten Description:
```

Table 10: Prompt for Revising Tool Functionality Description

```
You are an API documentation personnel, and your task is to write description
information for the parameters in the current API documentation.
I will provide you with basic information about the API and partial descriptions of
the existing parameters, but the original descriptions may not be comprehensive
enough.
Therefore, I will also provide you with some real use case examples. Based on the
given case examples, you should update the parameter descriptions to better assist
users in querying the API.
Each rewritten description should be no more than 10 words.
If you think the original information is accurate enough, then you don't need to
change it, just output the original content.

The updated parameter description of the API should be the following format:
For example:

{icl_examples}
If you think the original description is accurate enough, just repeat the original
description.
Starting below, you must follow this format:
Operation: The operation of the API
Path: The path of the API
Description: The original description of the API
Parameters: The parameters of the API, maybe None
Responses: The responses of the API, maybe None
Scenario Examples: The scenario examples of the API
Updated Parameter Description: The updated parameter description of the API



Now begin your mission!
{api_doc}
Scenario Examples: {scenario}
Updated Parameter Description:
```

Table 11: Prompt for Revising Tool Parameters

```
You are an API documentation personnel and your task is to write descriptive
information for the response formats in the current API documentation.
I will provide you with basic information about the API, and in addition, I will
give you some real-world usage examples.
Based on the given usage examples, I need you to write descriptive information for
each of the response values. The specific requirements are as follows:
The response format is a dictionary nested within another dictionary. I would like
you to add a description field to each dictionary to describe what the current value
 represents.
If you think the original information is accurate enough, then you don't need to
change it, just output the original content.

For example:
{icl_examples}

If you think the original description is accurate enough, just repeat the original
description.
Starting below, you must follow this format:
Operation: The operation of the API
Path: The path of the API
Responses: The responses of the API, maybe None
Scenario Examples: The scenario examples of the API
Updated Responses: The updated response schema of the API


Now begin your mission!
{api_doc}
Scenario Examples: {scenario}
Updated Responses:
```

Table 12: Prompt for Revising Tool Responses

```
React with Original Documentation:
Query: give me a poster of the second season of Big Bang Theory
Thought 1:Search for the TV show "Big Bang Theory"
Action 1:GET /search/tv
Action Input 1:
{   "Operation": "GET",
    "url": "https://api.themoviedb.org/3/search/tv",
    "params": {
    "query": "Big Bang Theory" }}
Observation 1: {..."id":1418,"origin_country":["US"],"original_language":"en","
original_name":"The Big Bang Theory","overview":"Physicists Leonard and Sheldon find
 their nerd-centric social circle with pals Howard and Raj expanding when aspiring
actress Penny moves in next door."...}
Thought 2:I have found "The Big Bang Theory" TV show. Now, I need to get the details
 of the second season to find the poster.
Action 2:GET /tv/1418/season/2
Action Input 2:
{   "Operation": "GET",
    "url": "https://api.themoviedb.org/3/tv/1418/season/2",}
Observation 2: Observation 2: {..."production_code":"3T7351","runtime":22,"
season_number":2,"show_id":1418,"still_path":"/y9cFxQyFRBOewwVvkMl6SaH2om0.jpg","
vote_average":7.7,"vote_count":48,...}
Thought 3:I have found the details of the second season of "The Big Bang Theory" and
 the poster for the first episode is available. I will now provide the poster for
the second season of "The Big Bang Theory."
Action 3:Final Answer: ![The Bad Fish Paradigm](https://image.tmdb.org/t/p/original/
y9cFxQyFRBOewwVvkMl6SaH2om0.jpg)
-------------------------------------------------------------------------------------
React with Our Documentation:
Query: give me a poster of the second season of Big Bang Theory
Thought 1:Search for the TV show "Big Bang Theory" to get its ID.
Action 1:GET /search/tv?query=Big%20Bang%20Theory
Action Input 1:
{"Operation": "GET",
    "url": "https://api.themoviedb.org/3/search/tv",
    "params": {
    "query": "Big Bang Theory"}}
Observation 1: {..."id":1418,"origin_country":["US"],"original_language":"en","
original_name":"The Big Bang Theory","overview":"Physicists Leonard and Sheldon find
 their nerd-centric social circle with pals Howard and Raj expanding when aspiring
actress Penny moves in next door."...}
Thought 2:I have found the ID for "The Big Bang Theory" TV show, which is 1418. Now,
 I need to get the images for the second season of this show.
Action 2:GET /tv/1418/season/2/images
Action Input 2:Action Input 2:
{   "Operation": "GET",
    "url": "https://api.themoviedb.org/3/tv/1418/season/2/images",}
Observation 2: {"id":3733,"posters":[{"aspect_ratio":0.666,"height":2233,"iso_639_1
":"en","file_path":"/2NBwUBZ4clwj6qO9fBinfxiB0dR.jpg","vote_average":5.454,"
vote_count":3,"width":1488},{"aspect_ratio":0.666,"height":1426,"iso_639_1":"en","
file_path":"/tULsqpXMTZ85BdFjNR9ScxBSbec.jpg","vote_average":5.322,"vote_count":5,"
width":950},{"aspect_ratio":0.667,"height":2268,"iso_639_1":"hu","file_path":"/
eFIj7bDVKRTssMMs67EkiLbZ0o9.jpg","vote_average":5.312,"vote_count":1,"width
":1512},...}
Thought 3:I have found the posters for the second season of "The Big Bang Theory".
Here are some of the images:
1. ![Poster 1](https://image.tmdb.org/t/p/original/2NBwUBZ4clwj6qO9fBinfxiB0dR.jpg)
2. ![Poster 2](https://image.tmdb.org/t/p/original/tULsqpXMTZ85BdFjNR9ScxBSbec.jpg)
3. ![Poster 3](https://image.tmdb.org/t/p/original/eFIj7bDVKRTssMMs67EkiLbZ0o9.jpg)
Final Answer: These are some of the posters for the second season of "The Big Bang
Theory".
```

Table 13: React with different documentation on RestBench-TMDB

```
RestGPT with Original Documentation:
Query: Add the first song from my music library to the playback queue and enable
single song repeat mode
Planner: Get the first song from my music library
API Selector: GET /me/tracks
Caller: Operation: GET
Input: {
    "url": "https://api.spotify.com/v1/me/tracks",
    "params": {
        "limit": "1"
    },
    "description": "Retrieve a list of the songs saved in the current Spotify user's
     'Your Music' library.",
    "output_instructions": "What is the name and id of the first track saved in the
    user's library?"
}
Parser: Name: Whitney Houston - I Wanna Dance with Somebody (Who Loves Me)
ID: 2tUBqZG2AbRi7Q0BIrVrEj
Caller: Execution Result: Successfully called GET /me/tracks. The name of the first
track saved in the user's library is "Whitney Houston - I Wanna Dance with Somebody
(Who Loves Me)" and the ID is 2tUBqZG2AbRi7Q0BIrVrEj.
Planner: Add the track "Whitney Houston - I Wanna Dance with Somebody (Who Loves Me)
" (2tUBqZG2AbRi7Q0BIrVrEj) to the playback queue
API Selector: POST /me/player/queue to add the track "Whitney Houston - I Wanna
Dance with Somebody (Who Loves Me)" (2tUBqZG2AbRi7Q0BIrVrEj) to the playback queue
Caller: Operation: POST
Input: {
    "url": "https://api.spotify.com/v1/me/player/queue",
    "data": {
        "uri": "spotify:track:2tUBqZG2AbRi7Q0BIrVrEj"
    },
    "description": "Add an item to the end of the user's current playback queue.",
    "output_instructions": "No specific output instructions provided."
}
Parser: The URI of the track added to the queue is spotify:track:2
tUBqZG2AbRi7Q0BIrVrEj
Caller: Execution Result: Successfully called POST /me/player/queue to add the track
 "Whitney Houston - I Wanna Dance with Somebody (Who Loves Me)" with the URI spotify
:track:2tUBqZG2AbRi7Q0BIrVrEj to the playback queue.
Planner: Enable single song repeat mode
API Selector: PUT /me/player/repeat with the parameter "repeat-track"
Caller: Operation: PUT
Input: {
    "url": "https://api.spotify.com/v1/me/player/repeat",
    "description": "Set the repeat mode for the user's playback to repeat the
    current track."
}
Parser: Error: Required parameter state missing.
```

Table 14: RestGPT with original documentation on RestBench-Spotify

```
RestGPT with Our Documentation:
Query: Add the first song from my music library to the playback queue and enable
single song repeat mode
Planner: Get the first song from my music library
API Selector: GET /me/tracks
Caller: Operation: GET
Input: {
    "url": "https://api.spotify.com/v1/me/tracks",
    "params": {
        "limit": "5"
    },
    "description": "Retrieve the updated list of saved songs in the user's library
    with a limit of 5.",
    "output_instructions": "Extract the names and ids of the saved songs."
}
Parser: The name of the artist is Whitney Houston and the id is 6
XpaIBNiVzIetEPCWDvAFP
The name of the artist is Whitney Houston and the id is 6XpaIBNiVzIetEPCWDvAFP
The name of the artist is Lana Del Rey and the id is 00FQb4jTyendYWaN8pK0wa
The name of the artist is Lana Del Rey and the id is 00FQb4jTyendYWaN8pK0wa
The name of the artist is Lana Del Rey and the id is 00FQb4jTyendYWaN8pK0wa
Caller: Execution Result: Successfully called GET /me/tracks. The names and ids of
the saved songs are as follows:
1. Artist: Whitney Houston, ID: 6XpaIBNiVzIetEPCWDvAFP
2. Artist: Lana Del Rey, ID: 00FQb4jTyendYWaN8pK0wa
Planner: Add the song by Whitney Houston (6XpaIBNiVzIetEPCWDvAFP) to the playback
queue
API Selector: POST /me/player/queue to add the song by Whitney Houston (6
XpaIBNiVzIetEPCWDvAFP) to the playback queue
Caller: Operation: POST
Input:
{
    "url": "https://api.spotify.com/v1/me/player/queue",
    "data": {
        "uri": "spotify:track:6XpaIBNiVzIetEPCWDvAFP"
    },
    "description": "Adding a specific song to the playback queue.",
    "output_instructions": "No specific output needed."
}
Parser: The URI of the song added to the playback queue is spotify:track:6
XpaIBNiVzIetEPCWDvAFP
Caller: Execution Result: Successfully called POST /me/player/queue to add the song
by Whitney Houston (6XpaIBNiVzIetEPCWDvAFP) to the playback queue. The URI of the
song added to the playback queue is spotify:track:6XpaIBNiVzIetEPCWDvAFP.
Planner: Enable single song repeat mode
API Selector: PUT /me/player/repeat to enable single song repeat mode
Caller: Operation: PUT
Input: {
    "url": "https://api.spotify.com/v1/me/player/repeat",
    "data": {
        "state": "track"
    },
    "description": "Set repeat mode for playback to single song repeat mode."
}
Parser: The player has been set to repeat mode.
Caller: Execution Result: Successfully called PUT /me/player/repeat to enable single
 song repeat mode
Planner: Final Answer: I have added the first song from your music library to the
playback queue and enabled single song repeat mode.
```

Table 15: RestGPT with our documentation on RestBench-Spotify

```
My family and I are planning a beer tasting trip to Italy and we need a
comprehensive list of beers available in different regions. Can you provide me with
that? Additionally, I would like to explore the local cuisine. Can you give me
recipes for Italian wedding soup? Lastly, I'm curious about the nutritional
information of milk. Can you fetch that for me?
---------------------------
I'm a food blogger and I'm looking for new recipe ideas. Can you provide me with
creative recipes for tacos? Also, fetch the details of the available chicken dishes
from KFC Chickens API.
---------------------------
"I'm a food blogger and I'm looking for unique and creative recipes to share with my
 audience. Can you recommend some interesting chicken recipes using the Recipe_v4
API? Additionally, I would like to know the nutritional information for milk. Please
 provide me with the energy, protein, carbohydrate, and fat content of milk.
---------------------------
I'm organizing a cocktail workshop for beginners and I want to teach them some easy
and delicious cocktail recipes. Can you suggest some cocktail recipes that use
ingredients like vodka, rum, and tequila? It would be great if you could also
provide some tips and tricks for making the perfect cocktails.
```

Table 16: Several examples of ToolBench-Food

```
You are an evaluator assessing the quality of a tool description. Your task is to
read the given description  of the tool and provide a score based on the following
criteria:
1. Information Completeness: How comprehensive is the description? Does it cover all
 essential aspects of the tool, including its main features, benefits, and use cases
?
2. Text Length: How concise is the description? Is it to the point without
unnecessary details or redundancy?

Your score should be a number between 1 and 3, where 1 is the lowest and 3 is the
highest. Please provide a brief explanation for your score.
The score 1 means that the description is incomplete, lacks essential information,
or is too verbose.
The score 2 means that the description is somewhat complete but could be improved in
 terms of information coverage or text length.
The score 3 means that the description is highly informative, well-structured, and
concise.

Please evaluate the description based on these criteria and provide a score for the
given tool description.


Tool: {tool}
Description: {description}
Score:
```

Table 17: Prompt for evaluating tool descriptions

```
You are an evaluator tasked with assessing the quality of the parameter section in a
 tool document. Your goal is to read the provided section and assign a score based
on the following criteria:

1. Completeness and Accuracy: How comprehensive and accurate is the description of
the parameters? Does it cover all necessary parameters, their types, ranges, and any
 constraints accurately?
2. Clarity and Understandability: How clear and understandable is the description of
 the parameters? Can readers easily comprehend the purpose and usage of each
parameter?

Your score should be a number between 1 and 3, where 1 is the lowest and 3 is the
highest. Please provide a brief explanation for your score.
The score 1 means that the parameter section is incomplete, inaccurate, or difficult
 to understand.
The score 2 means that the parameter section is somewhat complete and clear but
could be improved in terms of accuracy or clarity.
The score 3 means that the parameter section is highly informative, accurate, and
easy to understand.

Please evaluate the parameter section based on these criteria and provide a score
for the given tool document.

Tool: {tool}
Parameter Section: {parameters}
Score:
```

Table 18: Prompt for evaluating tool parameters

```
You are an evaluator tasked with assessing the quality of the response section in a
document. Your objective is to read the provided section and assign a score based on
 the following criteria:

1. Structural Accuracy: How well-structured is the response section? Does it follow
a logical sequence and provide a clear overview of the tool's response mechanism?
2. Comprehensive Parameter Explanations: How comprehensive and accurate are the
explanations of the parameters in the response section? Do they cover all necessary
details, including how each parameter affects the response and any dependencies
between parameters?

Your score should be a number between 1 and 3, where 1 is the lowest and 3 is the
highest. Please provide a brief explanation for your score.
The score 1 means that the response section is poorly structured, lacks clarity, or
provides incomplete parameter explanations.
The score 2 means that the response section is somewhat structured and informative
but could be improved in terms of clarity or completeness.
The score 3 means that the response section is well-structured, clear, and provides
comprehensive parameter explanations.

Please evaluate the response section based on these criteria and provide a score for
 the given tool document.

Tool: {tool}
Response Section: {response}
Score:
```

Table 19: Prompt for evaluating tool responses