NesTools : A Dataset for Evaluating Nested Tool Learning Abilities of Large Language Models

Han Han, Tong Zhu, Xiang Zhang, Mengsong Wu, Hao Xiong, Wenliang Chen*
Institute of Artificial Intelligence, School of Computer Science and Technology,
Soochow University, China
{hhan,tzhu7,xzhangxzhang23,mswumsw,hxiongxionghao}@stu.suda.edu.cn
wlchen@suda.edu.cn

Abstract

Large language models (LLMs) combined with tool learning have gained impressive results in real-world applications. During tool learning, LLMs may call multiple tools in nested orders, where the latter tool call may take the former response as its input parameters. However, current research on the nested tool learning capabilities is still under-explored, since the existing benchmarks lack relevant data instances. To address this problem, we introduce NesTools to bridge the current gap in comprehensive nested tool learning evaluations. NesTools comprises a novel automatic data generation method to construct large-scale nested tool calls with different nesting structures. With manual review and refinement, the dataset is in high quality and closely aligned with real-world scenarios. Therefore, NesTools can serve as a new benchmark to evaluate the nested tool learning abilities of LLMs. We conduct extensive experiments on 22 LLMs, and provide in-depth analyses with NesTools , which shows that current LLMs still suffer from the complex nested tool learning task[1].

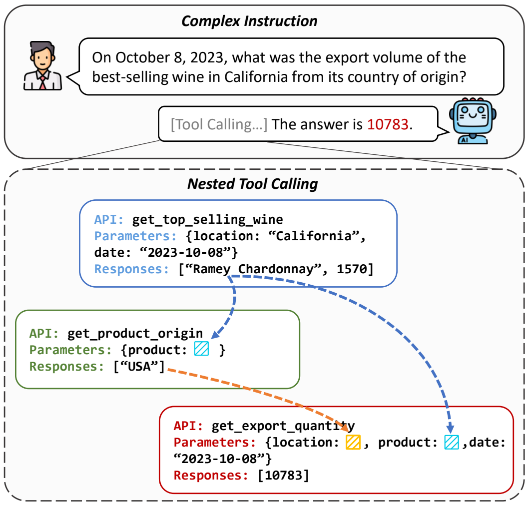

Figure 1: Example of nested tool calling.

1 Introduction

Large Language Models (LLMs) have shown powerful abilities in natural language understanding and reasoning (Achiam et al., 2023; Dubey et al., 2024; Yang et al., 2024; Zhu et al., 2024). To extend such abilities into real-world systems, tool learning (Inaba et al., 2023) has become a promising paradigm to solve complex problems and reduce hallucinations with external APIs, such as the calculator and the search engine (Patil et al., 2023; Schick et al., 2023). In a real-world application, LLMs may interact with multiple tools (Song et al., 2023; Ye et al., 2024; Basu et al., 2024; Huang et al., 2023). As shown in Figure 1, the multi-tool calling process is nested, where the latter tool may take the former one's response as an input parameter. This scenario is prevalent in accomplishing complex tasks with many execution steps.

However, the LLMs' nested tool learning ability is still under-explored, and corresponding benchmarks are absent to provide comprehensive evaluations. Among current tool learning datasets, some datasets completely ignore the nested tool calls (Tang et al., 2023; Patil et al., 2023). Other benchmarks only have a small quantity of nested tool calls (Li et al., 2023; Huang et al., 2024; Wu et al., 2024) or have low qualities with repetitive patterns (Shen et al., 2023) and coarse evaluations (Chen et al., 2024; Qin et al., 2023b). To this end, it is hard to comprehensively assess LLMs' performance on the real-world nested tool learning scenario, and provide insights for further model development.

To address the above challenges, we introduce NesTools , a high-quality nested tool learning benchmark to provide comprehensive evaluations. NesTools offers an innovative automated data

* Corresponding author
[1]Our code and dataset are available at https://github.com/hhan1018/NesTools

Proceedings of the 31st International Conference on Computational Linguistics, pages 9824–9844
January 19–24, 2025.

		NesTools (Ours)	T-Eval (Chen et al., 2024)	API-Bank (Li et al., 2023)	ToolBench (Qin et al., 2023b)	UltraTool (Huang et al., 2024)	BFCL (Patil et al., 2023)
Tools	**Amount**	3,034	50	73	16,464	436	1,618
	Avg. params (required)	2.24	1.24	1.97	1.01$^{\diamond}$	4.22	2.11
Instances	**Amount**	1,000	553	485	126,486	1,000	2,000
	Multiple-tool callings	1,000	553	122	85,330 $^{\diamond}$	867	490
	Nested-tool callings	830	N/A$^{\spadesuit}$	50	N/A$^{\spadesuit}$	227	0
	Avg. tool call	3.04	5.81	1.53	unknown	2.38	unknown
Multiple-tool scenario?		✔	✔	✔	✔	✔	✔
Evaluation of nesting?		✔	✘	✘	✘	✘	✘

Table 1: Comparison of tool learning datasets for evaluation. $^{\spadesuit}$ The tool calling procedure is carried under a multi-step setting and the exact number cannot be obtained. $^{\diamond}$ The statistics refer to Seal-Tools(Wu et al., 2024).

construction scheme, generating large-scale and more diverse examples of nested tool learning compared to existing datasets. The entire dataset construction process includes tool & instance generation, query generation, and data review & refinement. To ensure an accurate and consistent evaluation, we carefully select and cross-verify 1,000 data entries. For more comprehensive evaluations, we assess models on four deterministic dimensions: tool selection, tool calling order, parameter filling, and nested parameter filling. Compared to existing benchmarks in Table 1, our dataset focuses on the nested tool learning task and provides large-scale tools with more nested calls. In addition, our fine-grained assessment dimensions on nesting tool calls could provide a set of more comprehensive tool learning evaluations that are closely aligned with real-world scenarios.

We conduct extensive experiments on NesTools with 22 popular LLMs, including proprietary models and open-weight models. We provide thorough analyses in terms of the nesting depth, nesting structure, scaled model sizes, and robustness effects. The results show that although models benefit from size scaling, they still suffer from the simple tool selection, and the performance would degrade when tools are deeply nested. Our core contributions are as follows:

- We propose a novel automatic data construction pipeline to easily generate large-scale nested tool learning datasets.
- We introduce NesTools , a high-quality dataset with large-scale diverse examples for comprehensive nested tool learning evaluations.
- We conduct extensive experiments on 22 LLMs to verify their effectiveness and generalization abilities, providing detailed analyses and insights for LLMs in the field of nested tool learning.

2 Related Work

Tool learning Early works (Yao et al., 2023; Schick et al., 2023; Paranjape et al., 2023) have incorporated straightforward tools such as search engines and calculators to enhance LLMs' access to up-to-date information and precise mathematical reasoning. Following this, API-bank (Li et al., 2023) constructs several tools and tool-use dialogues. ToolBench (Qin et al., 2023b) employs real-world APIs to construct datasets capable of addressing a broader spectrum of user queries. T-eval (Chen et al., 2024) collects tools from common domains, leveraging a multi-agent paradigm to resolve solution annotations. UltraTool (Huang et al., 2024) starts with real-world queries that may require the construction of new tools.

Tool Evaluation Conducting reasonable and effective evaluations of tool learning capabilities for Large Language Models (LLMs) is essential (Qin et al., 2023a; Qu et al., 2024). API-Bank (Li et al., 2023) evaluates from perspectives of the correctness of API calls and the quality of LLMs' responses. ToolBench (Qin et al., 2023b) uses coarse-grained evaluation metrics including pass rate and win rate backed up by ChatGPT. UltraTool (Huang et al., 2024) evaluates from perspectives of planning, tool creation, and tool usage. Although these benchmarks include a limited number of nested tool calling, researchers have not undertaken a comprehensive analysis and evaluation of this phenomenon. Consequently, the performance of LLMs in handling nested tool calling remains under-explored.

To address this gap, we introduce NesTools , a dataset specifically constructed to feature a multitude of nested tool calling scenarios across various

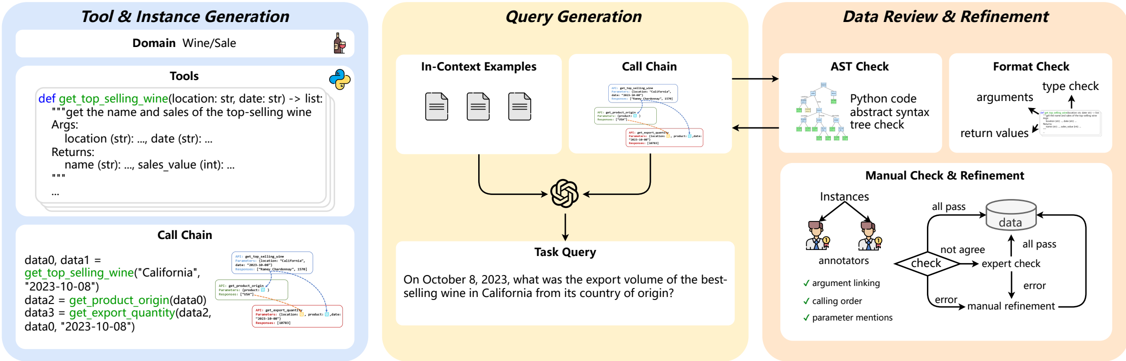

Figure 2: The dataset construction process of NESTOOLS .

domains. The purpose of this dataset is to systematically evaluate the performance of LLMs in nested tool learning abilities.

3 Dataset Construction

Leveraging large language models (LLMs) for data generation has become quite common in various domains (Wang et al., 2023; Yu et al., 2024). It is very necessary to explore a feasible and comprehensive data generation scheme for nested tool calling.

To obtain NESTOOLS , we propose an automatic dataset generation pipeline with manual reviews. As shown in Figure 2, the whole process starts with tool and instance generation (§3.1), followed by query generation (§3.2), and data review & refinement (§3.3). Lastly, we provide a comparative analysis for NESTOOLS with other datasets (§3.4).

3.1 Tool & Instance Generation

In real-world applications with a limited number of tools, call chains containing nested calls are relatively restricted and homogenized, and there are not too many non-duplicated and diverse call chains. To get a wide range of tools and instances, existing benchmarks mostly use a conventional two-round generation scheme, including filtering or constructing a portion of the tools first, and then filtering or generating some instances (Qu et al., 2024; Chen et al., 2024). However, the proportion of generated samples containing nested instances under this scheme is very small because LLMs cannot synthesize as many candidate tools as possible to obtain potential nested call chains under the limitations of the context. If relying on manual brainstorming, this is very costly and difficult to scale and ensure diversification.

How to increase the frequency of nested calling instances? LLMs are relatively unfamiliar with nested tool calling. However, they are very familiar with nested functions in code because the pretraining process has already injected large amounts of code knowledge into LLMs (Dubey et al., 2024). The two types of nesting can be very similar to some extent, with the common code pattern being to define a number of Python functions and then execute the calls in the main function. In this task, we ask GPT-3.5 to generate fixed-format functions as tools with annotation for necessary tool information, and then generate a call chain to execute the tools in the main function.

To ensure that the automated generation tools can cover a wider range of domains and minimize repetition rates as much as possible, we collect the domains from Seal-Tools (Wu et al., 2024), including 146 domains and 5,860 sub-domains, serving as a reference guide for the generation of the subsequent tools and instance.

When generating tools, we adopt a template with functions in Python code, including comments that describe the tool, required parameters, optional parameters, and return parameters. Initially, we attempt a nested call pattern using "func1(func2(...), *args)". However, this style is insufficient for more complex nested examples, as it results in lengthy expressions and lacks clarity in referencing specific return parameters when dealing with multiple return values. The improved generation pattern is as follows: "data0, ... = func1(...), data2, ... = func2(data0, ...)". This approach accommodates more complex nesting and indicates specific return values, as illustrated in Figure 2.

It is worth mentioning that the in-context examples are very important, there are different nested structures for different nested calls, to ensure the diversity and rationality of the generation, we carefully select a portion of high-quality samples to form the sample pool, and rotate these examples during the iterative generation process. At the same time, to minimize homogeneous generation, we set the generation temperature to 0.95 to seek for more diversified generation. The prompt and example can be found in Appendix A.1.

3.2 Query Generation

While the tool and call chain are generated, we reserve a comment to initially generate some raw queries that refer to the logic and flow of the call chain, although they may have flaws such as missing necessary parameter values. We then provide the entire actual call chain and the raw query to allow GPT-3.5 to generate the complete task, a step that allows the big model to populate all necessary parameters. Finally, we let GPT-4 (Achiam et al., 2023) optimize and rewrite the query to make it more complete and closer to the real requirements. Corresponding prompts can be found in Appendix A.2 and Appendix A.3.

3.3 Data Review & Refinement

To ensure the data quality, we perform strict reviewing steps and make further refinements. We first perform automatic checks. Given that nested tool calling contains multiple tools where each tool contains multiple return values, it is necessary to ensure that each return value in the call chain has a variable to receive it. As a result, we conduct the format check to filter out abnormal output formats and irregularities in generation. It is worth noting that in addition to the format error, the automated screening includes an abstract syntax tree (AST) check to ensure the consistency of tools and call chains. This process could align the return values and parameters between the candidate tools and the call chains.

After the automated check, we perform a manual review for further quality control. For the call chains, we mainly check whether the values of the parameters are in accordance with the description information in the golden tools. Besides, the most critical aspect of this reviewing step is to determine whether the nested parameters are filled in correctly. It requires manual tracing back to the parent node tool that returns the parameter, and carefully comparing the parent node's return value with the textual description of the parameter at this position. Potential errors include the parameter being a non-nested parameter but being filled as a nested one, or being a nested parameter but being filled with an incorrect return value from a previous tool.

For the task query review, we mainly check whether the non-nested parameters in the call chain are mentioned in the query, whether the query description conforms to the logic of the call chain, and whether the low-quality samples are optimized to meet real-world requirements. We employ two annotators to tag NesTools from two perspectives: consistency between the call chain and the tools, and whether the query meets the requirements. Besides, the annotators are required to refine the data instances if there are errors. If there is a disagreement between the two annotators, an expert would help make a final annotation. The overall annotation process shows a high agreement of 0.96. More details can be found in Appendix A.4.

3.4 Dataset Summary

As shown in Table 1, we compare NesTools in detail with the currently available datasets, and with the necessary number of tools and instances guaranteed, NesTools has a clear advantage in terms of nested tool calling, which not only covers a higher percentage of nesting and includes a new framework of evaluation LLMs for nested tool calling.

4 Experiments

4.1 Experimental Setup

In order to evaluate the LLM's understanding of the whole call chain of nested tool calls, we use the pattern of prompting for evaluation. For the parameter of the nested tools, the model is constrained to generate a placeholder like "API_call_{number}" to fill the parameter of nested position in the evaluation prompt.

For each sample, in order to simulate the scenario when the LLM faces similar tools, we utilize the gte-large retriever [2] to retrieve 5 similar tools for each gold tool from the tool pool. To minimize the occurrence of ambiguous tools, we control for the fact that the name of each retrieved tool must meet a certain Levenshtein distance requirement with the names of all gold tools.

We conduct experiments on NesTools with the latest proprietary and open-weight LLMs.

[2]https://huggingface.co/Alibaba-NLP/gte-large-en-v1.5

Model	Selection			Order			Parameter			Nested Param			Avg.
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	
Proprietary Models													
GPT-4o	**85.1**	**86.2**	**85.7**	**72.9**	**74.3**	**73.6**	**75.5**	**78.6**	**77.0**	**73.5**	**69.3**	**71.3**	**76.9**
GPT-4o-mini	77.3	72.6	74.9	60.6	56.8	58.6	65.3	64.6	64.9	58.8	48.1	52.9	62.8
GPT-3.5	69.0	71.2	70.1	47.9	50.4	49.1	56.7	62.0	59.2	46.5	41.7	44.0	55.6
Claude-3.5	79.6	82.8	81.2	64.3	68.2	66.2	70.6	75.9	73.2	68.3	65.5	66.8	71.8
Open-Weight Models													
LLaMA3.1-8B	65.8	71.7	68.6	44.3	50.7	47.3	44.3	51.1	47.5	40.8	42.7	41.7	51.3
LLaMA3.1-70B	82.3	81.9	82.1	68.9	68.8	68.8	57.7	60.3	59.0	67.6	61.1	64.1	68.5
LLaMA3.1-405B	**87.9**	**87.2**	**87.5**	**77.3**	**77.4**	**77.3**	66.4	68.5	67.4	**78.8**	**72.7**	**75.7**	**77.0**
Mistral-7B v0.2	49.6	44.1	46.7	22.0	21.8	21.9	35.2	33.0	34.1	21.9	14.1	17.1	29.9
Mixtral-8x7B	62.4	64.3	63.4	38.0	41.3	39.6	46.3	49.9	48.0	35.6	35.1	35.3	46.6
Mixtral-8x22B	70.4	75.1	72.7	49.6	54.7	52.0	59.4	66.8	62.9	53.0	52.3	52.6	60.1
Qwen2-0.5B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Qwen2-1.5B	39.6	18.1	24.8	12.4	4.4	6.5	28.4	14.2	18.9	0.0	0.0	0.0	12.5
Qwen2-7B	59.4	58.9	59.1	34.9	35.5	35.2	41.1	43.1	42.1	29.5	14.1	19.1	38.9
Qwen2-57B	63.8	68.9	66.2	40.2	45.5	42.7	47.6	54.3	50.7	37.8	29.0	32.8	48.1
Qwen2-72B	78.4	79.2	78.8	62.9	64.2	63.6	**67.0**	**69.9**	**68.4**	57.9	58.6	58.3	67.3
Qwen1.5-0.5B	10.2	0.2	0.3	0.0	0.0	0.0	2.2	0.1	0.1	0.0	0.0	0.0	0.1
Qwen1.5-1.8B	21.2	3.4	5.8	2.5	0.4	0.8	9.8	1.7	2.9	0.0	0.0	0.0	2.4
Qwen1.5-7B	55.0	35.5	43.2	27.8	18.7	22.3	38.0	27.1	31.7	21.2	5.2	8.4	26.4
Qwen1.5-14B	61.8	56.9	59.3	37.0	35.3	36.1	46.4	45.8	46.1	34.2	21.4	26.3	41.9
Qwen1.5-32B	65.2	61.1	63.1	41.7	40.7	41.2	52.3	51.3	51.8	45.6	27.2	34.1	47.5
Qwen1.5-72B	66.0	59.7	62.7	43.1	40.0	41.5	49.4	47.4	48.4	31.6	21.5	25.6	44.6
Qwen1.5-110B	69.7	65.0	67.3	49.0	46.7	47.8	56.3	54.8	55.6	44.8	31.8	37.2	52.0

Table 2: The main results of NESTOOLS . Avg. is the average F1 score of all evaluation metrics. **Bold** represents the best score among all models.

For the proprietary LLMs, we selected four representative LLMs: GPT-4o, GPT-4o-mini and GPT-3.5 from OpenAI and Claude3.5 from Anthropic[3]. For the open-weight LLMs, we choose the representative ones: LLaMA3.1 (Dubey et al., 2024), Qwen1.5 (Bai et al., 2023), Qwen2 (Yang et al., 2024), Mistral (Jiang et al., 2023) and Mixtral (Jiang et al., 2024). We perform detailed experiments on each of their sizes.

The detailed evaluation prompt is shown in Appendix B.2.

4.2 Evaluation Metrics

For NESTOOLS , to standardize the evaluation, we devise metrics for the following aspects: correctness of tool selection, correctness of the order of tool calls, correctness of parameter filling, and correctness of nested parameter filling. Each evaluation is a deterministic P/R/F1 metric.

Selection P/R/F1 measures the accuracy of LLM's tool selection, which is a common evaluation dimension.

Order P/R/F1 measures the accuracy of LLM's judgment of the previous and subsequent tools. Since nested tool calling includes the use of the return value of the previous tool, it is necessary to evaluate the order of the tools before and after.

Parameter P/R/F1 measures the correctness of LLM's filling parameters, which is also a common evaluation dimension.

Nested Param P/R/F1 is specific to nested calls and evaluates the accuracy of parameters at nested locations in the call chain.

Other metrics and details are listed in Appendix B.1.

[3]The version for GPT-4o is gpt-4o-2024-08-06, for GPT-4o -mini is gpt-4o-mini-2024-07-18, for GPT-3.5 is gpt-3.5-turbo-0125 and for Claude3.5 is claude-3-5-sonnet-20240620.

4.3 Main Results

In the field of tool learning, for a long time, the GPT-4 series has occupied the position of the best performance, and open-weight LLMs are once out of reach (Huang et al., 2024; Chen et al., 2024). However, as shown in Table 2, we are pleasantly surprised to find that LLaMA3.1-405B slightly outperforms GPT-4o on NESTOOLS , achieving the best performance among all evaluated LLMs. This signifies that open-weight and proprietary LLMs are currently very competitive in tool learning.

We observe that the LLaMA3.1 series exhibits significant advantages in Selection and Order. Notably, LLaMA3.1-8B outperforms LLMs like Mixtral-8x7B and Qwen2-57B, which are substantially larger in size. Qwen2-72B reaches the best performance in parameters among open-weight LLMs. Mixtral-8x22B has the highest format accuracy among open-weight LLMs, second only to GPT-4o and Claude-3.5 in all LLMs.

However, in terms of parameter filling, the LLaMA3.1 series shows an unusual disadvantage. Analysis of sample cases reveals that LLaMA3.1 is less sensitive to parameter types, often generating incorrect parameter types in JSON outputs, even though the extracted answer information may be correct. This issue is most pronounced in the LLaMA3.1 series. LLaMA3.1-70B is most severely affected, resulting in a loss of nearly 10 percentage points in parameter metric. It is worth noting that such problems are significantly less prevalent in other LLMs, and they are entirely absent in Claude-3.5. This suggests that compared to other LLMs, the LLaMA series may have undergone relatively less training focused on JSON format alignment.

Nested tool calling presents a certain level of difficulty, and the stronger the LLM's capabilities, the better it performs in resolving nested tool calling. Filling nested parameters requires the model to truly understand the call chain and identify which specific return value corresponds to each nested parameter. Judging from the metrics of parameters and nesting, LLM's performance in nesting is generally lower than that in parameters, but this gap is narrowing as the LLM's overall performance improves.

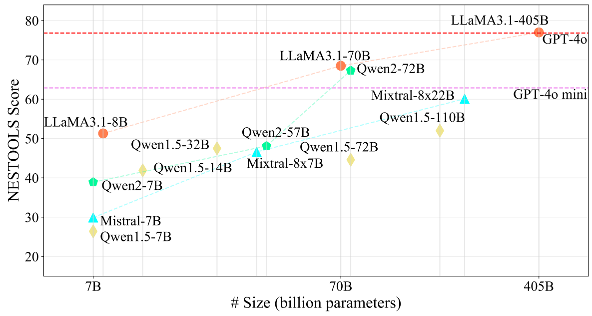

Figure 3: Model scaling results on NESTOOLS .

5 Further Analysis

5.1 Scaling Analysis

Scaling law is common in LLMs. As shown in Figure 3, we can conclude that most LLMs such as Qwen2, LLaMA3.1, Mistral, and GPT-4o series follow the scaling law in NESTOOLS . The larger the size, the stronger its performance in tool calling, which is consistent with the previous research conclusions. However, it is worth mentioning that the Qwen1.5 series does not fully follow the scaling law. Compared with the initially released Qwen1.5 series, Qwen team released Qwen1.5-32B two months later. We speculate that this process may include more high-quality data and more optimized iterative training.

5.2 Nesting Depth Analysis

Different call chains have different call depths. We consider the entire call chain's nesting depth to be 1 if the instance contains no nested calls. For nested tool calling instances, each tool is at a specific nesting depth. Tools that do not use the return value of previous tools are at depth 1. If a tool uses the return value of a previous tool, its depth is the previous tool's depth plus one, and so on. For an instance, its overall nesting depth matches the deepest level among the tools.

We analyze the performance of LLMs on NESTOOLS with different nesting depths. As shown in Figure 4, although the performance of the different LLMs varies, each of them shows a tendency that the deeper the nesting depth, the worse the performance of LLMs. This undoubtedly illustrates the difficulty of nested tool calling and its importance for multiple tool calling.

This trend can be attributed to several factors. Firstly, as the nesting depth increases, the complexity of the task grows exponentially. Each additional layer of nesting introduces new dependencies and

Setting	Selection			Order			Parameter			Nested Param			Avg.
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	
base	**65.8**	71.7	**68.6**	**44.3**	50.7	**47.3**	**44.3**	51.1	**47.5**	**40.8**	42.7	**41.7**	**51.3**
w/o type	62.7	**73.6**	67.7	40.5	**51.4**	45.3	42.4	52.7	47.0	35.7	43.1	39.0	49.8
w/o conjunction	59.8	73.4	65.9	35.7	48.3	41.0	41.1	**53.4**	46.5	33.2	**43.8**	37.7	47.8

Table 3: Robustness analysis conducted on LLaMA3.1-8B, base represents the results of the original setting, w/o type represents masking the type of response parameters of each tool, and w/o conjunction represents removing the conjunctions of the query.

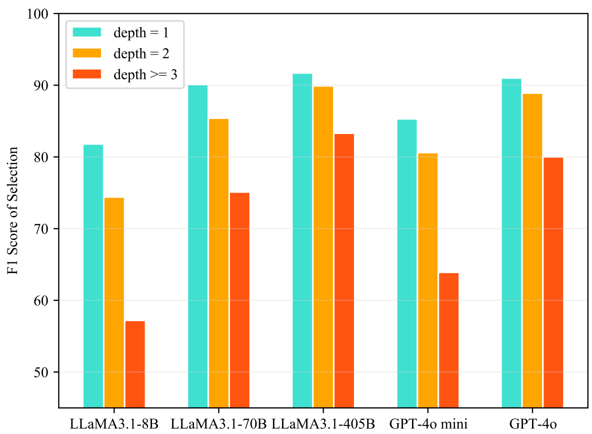

Figure 4: The relation between nesting depth and Selection F1 among LLMs.

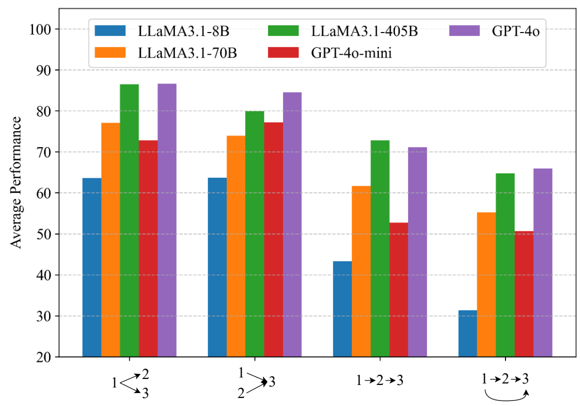

Figure 5: The averaged performance of different nesting structures. The arrow between two numbers indicates the nesting shape. For example, the first structure type (1→2,1→3) denotes that the 1st tool call's response contributes the input parameters for both the 2nd and the 3rd tool calls.

potential points of failure, making it more challenging for the LLM to maintain coherence and accuracy throughout the entire call chain. Secondly, deeper nesting requires the model to retain and process more contextual information over longer sequences, which can lead to issues with memory and attention span. This is particularly relevant for LLMs that have fixed-length context windows, where the ability to handle long-range dependencies diminishes as the sequence length increases.

5.3 Nesting Structure Analysis

As shown in Figure 5, we observe a similar phenomenon with the nesting depth experiment, where shallow tool calls result in higher average performance. However, the averaged performance would be higher if there is a leading tool call (1→2,1→3) that is followed by two separate calls (1→3,2→3). Besides the long nesting call structure (1→2→3), the additional nesting connection (1 → 3) at the most right column obtain the lowest performance among all the structures, which shows the adverse effect of structure complexity.

5.4 Robustness Analysis

Although existing benchmarks have comprehensively evaluated the tool calling ability of LLMs, there is a lack of robustness analysis for tool calling based on these benchmarks. We intend to analyze the robustness of tool calls from both tool and query perspectives on NESTOOLS .

From the query perspective, we notice that some conjunctions that do not affect the meaning often appear in the query of tool calls, such as "Then, After that, Please", etc. We guess whether the model works well because of the existence of a shortcut. Considering that the rule-based screening cannot completely cover the connectives in the query, we intend to remove these additional words and restore an original direct query. Considering that the rule-based screening cannot completely cover the connectives in the query, we intend to ask LLMs to remove these additional words. The prompt can be found in Appendix B.3. To ensure that this operation does not affect the core information of

(a) Error proportions of non-nested parameters in Qwen2-7B

(b) Error proportions of non-nested parameters in LLaMA3.1-8B

(c) Error proportions of nested parameters in Qwen2-7B

(d) Error proportions of nested parameters in LLaMA3.1-8B

Figure 6: Parameter error analysis for Qwen2-7B and LLaMA3.1-8B

the query, we check some samples and don't find biases caused by this process.

From the tool perspective, we would like to explore the impact of tool integrity on the experimental results when the tool is missing a small part of the information, in order to keep consistency with the original evaluation process, and to ensure that the core information of the tool is complete, we only mask out the type of the return value of the tool to explore the impact of tool completeness on the experimental results.

As shown in Table 3, In the case of masking off the type of the tool return value, LLaMA3.1-8B shows a slight decrease in all metrics. However, In the case of removing the connectives in the query, the performance drops more significantly, especially the Order metric, which proves that the conjunction does help LLMs understand the order of the call chain. An interesting observation is that after applying additional factors, the R-value metrics generally increase, indicating that LLM tends to call more tools to cover the gold standard answers. In summary, it can be inferred that the robustness of current LLMs in the field of tool calling is still insufficient and worth exploring.

5.5 Error Analysis

We analyze the errors from LLMs during the evaluation process, unfolding both the tool and parameter perspectives.

5.5.1 Tool Error

We summarize the errors in the tool perspective and divide them into five categories: tool selection, tool omission, tool redundancy, hallucinations, and generation format. Since these types of errors are usually mixed, it is hard to provide quantitative analysis. To this end, we provide each error category with a case study in Appendix B.4.

Here are our overall findings: *(1)* **Tool selection**: We find that similar tools may confuse the model, leading to errors in tool selection. *(2)* **Tool Omission**: Although the golden tools corresponding to each query are included in the evaluation prompt, the model may still miss some key tools. *(3)* **Tool Redundancy**: LLMs sometimes call redundant tools that are not task-related. *(4)* **Hallucinations**: When faced with some specific tasks, the model may hallucinate and construct tools on its own that do not exist in the tool pool. *(5)* **Generation Format**: During the evaluation process, some LLMs may generate results in the wrong format, which cannot be parsed for evaluations. The results regarding the accuracy of the format, as presented in Appendix B.1, reveal that smaller models are more inclined to generate content in incorrect formats.

5.5.2 Parameter error

For errors in the parameter section, we analyze the error types from two perspectives: *(1)***Non-nested parameter** errors include parameter omissions (Omission), redundant parameters (Redundancy), the parameter value is correct but the type is incorrect (Type), the parameter value is wrongly extracted (Extraction), and other errors (Others). *(2)***Nested parameter** errors include non-nested parameter mistaken for nested (Hallucination), nested parameter not involved (Omission), nested parameter recognized as non-nested parameter (Identification), nested parameter identified but take an incorrect value (Confusion).

As shown in Figure 6, we count the proportion of different error categories of Qwen2-7B and LLaMA3.1-8B. For non-nested parameters, LLaMA3.1-8B tends to mispredict parameter types, while Qwen2-7B makes fewer mistakes. For nested parameters, the most common mistake made by Qwen2-7B is recognizing nested parameters as non-nested parameters. The errors made by LLaMA3.1-8B are relatively average, and they are more likely to fill in the wrong nested parameter values after identifying them. This shows that LLaMA3.1-8B is more capable of identifying nested tool calling than Qwen2-7B to some extent.

6 Conclusion

In this paper, we introduce NesTools , a large-scale dataset of high-quality for evaluating the LLMs' nested tool learning abilities. We propose a novel automatic method to generate cases involving nested tool calls. Through a multi-step process of manual review and refinement, we ensure the quality of NesTools . Statistical analysis reveals that our dataset spans a wide range of domains. Furthermore, experiments conducted on 22 LLMs demonstrate the significance of this task. Our findings indicate that LLMs still struggle with effectively handling nested tool calls. We hope that NesTools , combined with our detailed experiments, can provide valuable insights and inspire future in-depth exploration of practical applications in tool agents.

Limitations

This study mainly builds complex nested tool calling instances, providing a new perspective on nested tool calling. Although promising, in reality, there may be situations where the necessary preceding tools are missing, and the model needs to use its own capabilities to perform additional processing on the return value. Another limitation is that the tools involved in this study cannot be executed. Although these tools are not specific implementations, they can be used as simulation representations to accurately describe the functions of the tools, which can be used to evaluate the capabilities of LLMs in tool learning and provide guidance for the future development of actual tools. Later, we will try to design real execution interfaces for these tools to better simulate nested tool calls in real situations.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 62376177, 62261160648) and Provincial Key Laboratory for Computer Information Processing Technology, Soochow University. This work is also supported by Collaborative Innovation Center of Novel Software Technology and Industrialization, Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions. We would also like to thank the anonymous reviewers for their insightful and valuable comments.

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*, abs/2303.08774.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *ArXiv preprint*, abs/2309.16609.

Kinjal Basu, Ibrahim Abdelaziz, Subhajit Chaudhury, Soham Dan, Maxwell Crouse, Asim Munawar, Vernon Austel, Sadhana Kumaravel, Vinod Muthusamy, Pavan Kapanipathi, and Luis Lastras. 2024. API-BLEND: A comprehensive corpora for training and benchmarking API LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12859–12870, Bangkok, Thailand. Association for Computational Linguistics.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9510–9529, Bangkok, Thailand. Association for Computational Linguistics.

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*, abs/2407.21783.

Shijue Huang, Wanjun Zhong, Jianqiao Lu, Qi Zhu, Jiahui Gao, Weiwen Liu, Yutai Hou, Xingshan Zeng, Yasheng Wang, Lifeng Shang, Xin Jiang, Ruifeng Xu, and Qun Liu. 2024. Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4363–4400, Bangkok, Thailand. Association for Computational Linguistics.

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*, abs/2310.03128.

Tatsuro Inaba, Hirokazu Kiyomaru, Fei Cheng, and Sadao Kurohashi. 2023. MultiTool-CoT: GPT-3 can use multiple external tools with chain of thought prompting. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1522–1532, Toronto, Canada. Association for Computational Linguistics.

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

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*, abs/2401.04088.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. 2023. Art: Automatic multi-step reasoning and tool-use for large language models. *ArXiv preprint*, abs/2303.09014.

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

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

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

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Tool learning with large language models: A survey. *ArXiv preprint*, abs/2405.17935.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. 2023. Taskbench: Benchmarking large language models for task automation. *ArXiv preprint*, abs/2311.18760.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *ArXiv preprint*, abs/2306.06624.

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

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: Self-instruct tool learning dataset for agent tuning and detailed benchmark. *ArXiv preprint*, abs/2405.08355.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *ArXiv preprint*, abs/2407.10671.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. 2024. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *ArXiv preprint*, abs/2401.00741.

Junjie Yu, Xing Wang, and Wenliang Chen. 2024. Reliable data generation and selection for low-resource relation extraction. In *AAAI Conference on Artificial Intelligence*.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. 2024. Llama-moe: Building mixture-of-experts from llama with continual pre-training. *ArXiv preprint*, abs/2406.16554.

A Dataset Construction details

A.1 Tool/Instance Generation

The corresponding prompt is presented in Table 4. An example for the generation process can be found in Table 5.

A.2 Query Normalization

The corresponding prompt is presented in Table 6.

A.3 Query Refinement

The corresponding prompt is presented in Table 7.

A.4 Other details

Based on predefined review criteria, the final agreement is computed by averaging the concordance between query and call chain verification. N_T means that the data is considered error-free by all annotators, and data with N_F is considered to have errors by all annotators. $N_{overall}$ represents the total number of data. The final agreement is defined as:

$$p = \frac{N_T + N_F}{N_{overall}}$$

B Evaluation details

B.1 Other Evaluation Metrics

In addition to the four metrics of **Selection**, **Order**, **Parameter**, and **Nested Param**, we also consider the following metrics:

Format measures the accuracy of LLM's output format.

Tree measures the pass rate of the entire call tree of LLMs, requiring absolute correctness in the four previous metrics of Selection, Order, Parameter, and Nested Param to be considered a pass.

We provide the format accuracy and Tree pass rate of all evaluated LLMs in Table 8.

B.2 Evaluation Prompt

The corresponding prompt is presented in Table 9.

B.3 Prompt of removing conjunctions

The corresponding prompt is presented in Table 10.

B.4 Error Examples

B.4.1 Tool Selection

The error examples for Tool Selection are shown in Table 11.

B.4.2 Tool Omission

The error examples for Tool Omission are shown in Table 12.

B.4.3 Tool Redundancy

The error examples for Tool Redundancy are shown in Table 13.

B.4.4 Hallucinations

The error examples for Hallucinations are shown in Table 14.

B.4.5 Generation Format

The error examples for Generation Format are shown in Table 15.

Tool/Instance Generation

Please generate several realistic functions with definitions in the given specific domain and generate complete function calling statements in ‘main()‘, shaped like "data0,data1 = func0(*args)" , "data2,data3,data4 = func1(*args)",etc. Do not add any additional python code in ‘main()‘. The most important thing is that the former function’s return value should be latter ones’ input argument, in other words, data1, data2, etc. should work as the arguments of subsequent function calls.In this scenario, the subsequent function will use a return value of a preceding function as its parameter, you need to ensure that the inner meaning of this parameter in both functions should be consistent, with identical parameter type, while slight difference in parameter names and descriptions are acceptable.
All required parameters must be filled in, by directly writing the actual values in order. Optional parameters can be filled in or not filled in, but remember to add "parameter_name=" before the value of the optional parameter if you are sure to fill in this optional parameter.
You need to write the task in natural language as a comment under the function calling statements in ‘main()‘ to describe the entire calling process. Please ensure that it includes all the function calling values.
Don’t use python’s intrinsic functions in your generation, such as lists and dictionaries (index access, slicing, adding, deleting, etc.).
The generated API only needs to be related to the given domain and does not need to be completely subordinate to the given domain.

Here are some examples.
{example}

It’s your turn to generate.
Domain: {}
Answer:

Table 4: Prompt of Tool/Instance Generation.

An example for Tool/Instance Generation

````
Domain: Food and Beverage/Food product development
```python
def get_top_selling_wine(location: str, date: str):
    """Retrieve the top-selling wine in a specific location on a specific date.

    required parameters:
        location (str): the geographical location (country, region, city)
        date (str): the date of the sales in 'YYYY-MM-DD' format

    Returns:
        wine_name (str): the name of the top-selling wine
        sales_value (float): the sales value of the wine in dollars
    """
    ...
def get_product_origin(product: str):
    """Retrieve the origin of a specific product.

    required parameters:
        product (str): the name or code of the product

    Returns:
        origin (str): the geographical origin of the product
    """
    ...
def get_export_quantity(location: str, date: str, product: str):
    """Retrieve the export quantity of a specific product
     from a specific location on a specific date.

    required parameters:
        location (str): the geographical location (country, region, city)
        date (str): the date of the export in 'YYYY-MM-DD' format
        product (str): the name or code of the product

    Returns:
        quantity (int): the export quantity of the product
    """
    ...
def main():
    data0, data1 = get_top_selling_wine("California", "2023-10-08")
    data2 = get_product_origin(data0)
    data3 = get_export_quantity(data2, "2023-10-08", data0)
    # I would like to know which wine sold the best in California on October 8th,
     2023. Could you provide me with the origin of this wine and inform me about
      the export data of this wine from the local area on that day?
if __name__ == "__main__":
    main()
```
````

Table 5: An example for Tool/Instance generation process.

Query Normalization

Below are some APIs, a task, and API call result for this task. You need to check if any values of the API call result that are not present in the task, incorporate those values into the original task without mentioning the specific APIs, rewrite the original task to ensure that it includes all the values,which is very important.If the value is in the format of complex data type (such as a dictionary, list, tuple, etc.), you need to incorporate each item from the value into the task in natural language, rather than directly adding the value in the raw format to the task.
The instructions of the rewritten task do not need to be completely consistent with the information in the given API such as the name and despcription, and some uniqueness of the real-life scenario should be retained.
You only need to generate the rewritten task without adding any additional explanations.

Here is an example.
{example}

It's your turn to generate.

APIs:
{gold apis}
Task:
{task}
API call result:
{call chain with gold apis}
rewritten task:

Table 6: Prompt of query normalization.

Query Refinement

You are an excellent task rewriter. Next, I will give you a task query, and ask you to rewrite it in high quality. Please follow the tips below.
Tips:
1. Please make sure that the actual name of the API(e.g., get_article_headline) involved in the call does not appear in the rewritten task.
2. The API call result represents the golden tool calling process corresponding to the query. Parameter value in the "API_call_" format represents a return value from a previous tool. Refer to the provided APIs information to ensure that the rewritten task is consistent with the golden tool calling process.
3. Please make sure that all regular parameters in the gold tool calling process, which are not in the "API_call_" format, can be extracted or inferred from the rewritten task.
4. Without changing the core meaning and logic of the original task, and while adhering to the call chain, appropriately modify the form of expression, make the task more coherent and aligned with real-world scenarios.

Here's an example:
{example}

It's your turn to generate.

APIs:
{gold apis}
Task:
{original task}
API call result:
{call chain with gold apis}
rewritten task:

Table 7: Prompt of query refinement.

Model	Format	Tree
Proprietary Models		
GPT-4o	**100.0**	25.9
GPT-4o-mini	94.4	18.3
GPT-3.5	99.4	13.5
Claude-3.5	**100.0**	**26.8**
Open-Weight Models		
LLaMA3.1-8B	98.0	6.9
LLaMA3.1-70B	99.1	9.7
LLaMA3.1-405B	97.5	14.7
Mistral-7B v0.2	68.8	2.9
Mixtral-8x7B	91.3	7.4
Mixtral-8x22B	**99.5**	17.5
Qwen2-0.5B	1.2	0.0
Qwen2-1.8B	66.5	0.0
Qwen2-7B	94.2	4.6
Qwen2-57B	97.5	9.2
Qwen2-72B	98.6	**20.4**
Qwen1.5-0.5B	3.2	0.0
Qwen1.5-1.8B	12.6	0.0
Qwen1.5-7B	59.4	2.4
Qwen1.5-14B	85.4	7.9
Qwen1.5-32B	85.6	10.7
Qwen1.5-72B	86.1	7.8
Qwen1.5-110B	88.8	13.0

Table 8: Format accuracy and Tree pass rate of NESTOOLS . **Bold** represents the best score among all models.

Prompt of Evaluation

You have access to a list of APIs and the task description.You need to follow the given task description and determine which API to call in sequence according to the order required by the task description. API can be retrieved from the APIs list. Finally, you only need to return the API call result without any other content.
The final result should be in the format of [{"api_name":__,"api_id":__,"parameters":{"arg0":"value0", "arg1":"value1",...},"responses":{"arg0":"API_call_0", ... ,"argn":"API_call_n"}},{"api_name":__, "api_id":__,"parameters":{"arg0":"value0","arg1":"value1",...},"responses":{"arg0":"API_call_{n+1}", ...}}, ...].
You don't need to know the actual return value of the API call, just assign each return value as a string "API_call_{number}" in "responses", such as "API_call_0","API_call_1","API_call_2" and so on. The "number" in "API_call_{number}" should increase by one from 0 globally.
Please first determine which APIs to call in sequence based on the task, and then determine the parameter values of each API depending on the specific details of the task. If you decide to call the API, you need to fill in all this API's required parameters which can be found in this API's "required" list. If you think the task does not include the actual value of a necessary parameter in API's "required" list, you can assign the necessary parameter a value of "UNK". The remaining parameters are optional parameters, determine whether to fill them in according to the task. If you think the parameter value to be filled in is the return value of a previous API call, set it as "API_call_x", then the parameter value can be filled in with "API_call_x".

Now it is your turn to generate the API call result based on the APIs and task description below. Remember that you only need to generate the API call result, not any additional explanations.
APIs:
{tools}
Task description:
{task description}
API call result:

Table 9: Prompt of evaluation.

Remove Conjunctions

Now there is a task description. I hope you can remove potential conjunctions like "Please", "then", "subsequently", "After that" etc., while maintaining the core meaning. Ensure that the text remains in natural language without punctuation errors and adjust the capitalization accordingly.

Here is an example:
{example}

Now its your turn:
Task description: {}
Result:

Table 10: Prompt of removing conjunctions.

Model	Query	Response
Qwen2-7B	I currently require a brand new red medium-sized dress. Please assist in designing the layout and selecting a luxurious silk fabric for this dress. Finally, arrange for Taylor Smith to bring the entire design to life and maintain oversight of the progress.	[{"api_name": "design_garment", "parameters": {"garment_type": "dress", "color": "red", "size": "medium"}, "responses": {}}, {"api_name": "fabric_selection", "parameters": {"design": "A red medium-sized dress", "fabric_type": "luxurious silk"}, "responses": {}}, {"api_name": "choose_manufacturer", "parameters": {"design_sketch": "A red medium-sized dress made of luxurious silk", "material_list": ["luxurious silk"], "location": "Taylor Smith's workshop"}, "responses": {}}]
LLaMA3.1-8B	Please share the title of a piece in the politics division that incorporates the word which is elections. Following this, extract information concerning the author who penned the aforementioned piece. Lastly, suggest 5 articles that bear similarity about elections.	[{"api_name": "request_article", "parameters": {"article_title": "piece in the politics division that incorporates the word which is elections", "author": "UNK", "access_link": "UNK"}, "responses": {"access_granted": "API_call_0", "download_link": "API_call_1"}}, {"api_name": "get_author_info", "parameters": {"author": "API_call_1"}, "responses": {"birthplace": "API_call_2", "articles_written": "API_call_3"}}, {"api_name": "recommend_similar_articles", "parameters": {"keyword": "elections", "limit": "5"}, "responses": {"similar_articles": "API_call_4"}}]

Table 11: Error examples for tool selection.

Model	Query	Response
LLaMA3.1-70B	Scan the ISBN "978-3-16-148410-0" of a book to extract comprehensive data and verify its presence in the library. Upon identifying the book situated, launch an augmented reality interaction correlating to the book.	[{"api_name": "scan_isbn", "parameters": {"isbn": "978-3-16-148410-0"}, "responses": {"book_details": "API_call_0", "availability": "API_call_1"}}, ~~{"api_name": "locate_book", "parameters": {...}, "responses": {...}},~~{"api_name": "engage_ar_experience", "parameters": {"availability": "API_call_1", "exact_location": "API_call_0"}, "responses": {"ar_message": "API_call_2", "ar_duration": "API_call_3"}}]
Qwen2-7B	I would like to know which company is responsible for producing the Smartphone X10, and I am also interested in understanding the comprehensive development value of the industrial zone it belongs to.	[{"api_name": "get_product_manufacturer", "parameters": {"product_name": "Smartphone X10"}, "responses": {"manufacturer_name": "API_call_0"}}, ~~{"api_name": "get_factory_location", "parameters": {...}, "responses": {...}},~~{"api_name": "evaluate_property", "parameters": {"location": "API_call_0", "type_of_land": "industrial", "area": 100.0}, "responses": {"land_value": "API_call_1", "development_potential": "API_call_2"}}]

Table 12: Error examples for tool omission.

Model	Query	Response
LLaMA3.1-8B	I'm planning a metro communication route that needs to span about 50 km, and we're targeting a bandwidth capacity of 100 Gbps. Could you please provide a modulation format that will work well over that distance. Also, there's the challenge of compensating for a line loss of around 3.5 dB, so I'll need to fine-tune the settings accordingly. Once got the right settings, Could you tell me how to set up a stable connection between A1 and B1?	[{"api_name": "choose_modulation_format", "parameters": {"route_type": "metro", "distance": 50, "bandwidth": 100}, "responses": {"modulation_scheme": "API_call_0", "system_settings": "API_call_1"}}, {"api_name": "adjust_amplification", "parameters": {"format": "API_call_0", "line_loss": 3.5}, "responses": {"optimized_gain": "API_call_2", "amplifier": "API_call_3", "signal_quality": "API_call_4"}}, {"api_name": "setup_receiver", "parameters": {"trans_config": "API_call_3", "link_budget": "API_call_2", "channel_type": "single-mode"}, "responses": {"receiver_status": "API_call_5", "signal_quality": "API_call_6"}}, {"api_name": "establish_connection", "parameters": {"amplifier_type": "API_call_3", "end_points": "A1-B1"}, "responses": {"connection_status": "API_call_7", "connection_details": "API_call_8"}}]
Qwen2-7B	I'm working on a project called "Alien Planet", which is a VFX scene I aim to create. It should last about 120 seconds and I want it to be in 4K resolution to ensure high quality. I'd like to include SFX featuring alien creatures and spacecraft in a mysterious atmosphere. Once everything is put together, I need the scene to be in MP4 format. Could you guide me with the whole process?	[{"api_name": "create_vfx_scene", "parameters": {"scene_name": "Alien Planet", "duration": 120, "resolution": "4K"}, "responses": {}}, {"api_name": "add_special_effects", "parameters": {"scene_path": "path_to_Alien_Planet_scene", "effects_list": ["alien_creatures", "spacecraft"], "mood": "mysterious"}, "responses": {}}, {"api_name": "render_scene", "parameters": {"scene_objects": ["alien_creatures", "spacecraft"], "quality": "high", "resolution": [3840, 2160]}, "responses": {}}, {"api_name": "export_final_scene", "parameters": {"scene_path": "path_to_rendered_Alien_Planet_scene", "format_type": "MOV"}, "responses": {}}]

Table 13: Error examples for tool redundancy.

Model	Query	Response
LLaMA3.1-8B	Please assist in designing an optimal prosthesis for a 35-year-old male, who has an unremarkable medical past and has suffered a fracture impacting his upper extremity. Afterward, evaluate the patient's steady condition and anticipate a complete recovery within a timeframe of 6 months. Finally, arrange an orthopedic surgical procedure for the patient.	[{"api_name": "design_prosthesis", "parameters": {"patient_info": "{"age": 35, "gender": "male", "medical_history": "unremarkable"}", "injury": "fracture", "affected_area": "upper extremity"}, "responses": {"prosthesis_details": "API_call_0", "estimated_cost": "API_call_1"}}, {"api_name": "patient_assessment", "parameters": {"patient_condition": "stable", "recovery_expectations": "complete recovery within 6 months"}, "responses": {"assessment_report": "API_call_2"}}, {"api_name": "schedule_orthopedic_surgery", "parameters": {"prosthesis": "API_call_0", "patient_info": "{äge:̈ 35, g̈ender:̈ m̈ale,̈ m̈edical_history:̈ ünremarkable}̈", "surgery_type": "orthopedic", "surgery_date": "2024-03-16"}, "responses": {"surgery_schedule": "API_call_3"}}]
Qwen2-7B	Assess the caliber of the Python code in the file found at "path/to/file.py". Subsequently, refine the code using the tool PyLint. Finally, deploy the optimized code as a web application to the production environment.	[{"api_name": "analyze_code_quality", "parameters": {"file_path": "path/to/file.py", "language": "Python"}, "responses": {"issues_found": ["issue1", "issue2"], "complexity_score": 4.5, "maintainability_score": 8.2}}, {"api_name": "optimize_code", "parameters": {"issues": ["issue1", "issue2"], "optimization_tool": "PyLint"}, "responses": {"optimized_code": "optimized_code_snippet"}}, {"api_name": "deploy_app", "parameters": {"app_id": "optimized_code_snippet", "server_type": "Production", "region": "Global"}, "responses": {"deployment_status": true, "deployed_url": "https://optimized-code.example.com"}}]

Table 14: Error examples for hallucinations.

Model	Query	Response
LLaMA3.1-405B	Please check the safety status of a car with registration number ABC123, last inspected on May 12, 2022. Issue a class C driving license to John Doe aged 30. Then book a vehicle inspection appointment for a truck with registration number DEF456 on June 20, 2022.	[{"api_name": "check_vehicle_safety", "parameters": {"vehicle_type": "car", "registration_number": "ABC123", "inspection_date": "May 12, 2022"}, "responses": {"safety_status": "API_call_0", "next_inspection_date": "API_call_1"}}, {"api_name": "issue_driving_license", "parameters": {"applicant_name": "John Doe", "age": 30, "license_type": "class C"}}, "responses": {"license_number": "API_call_2", "expiration_date": "API_call_3"}}, {"api_name": "book_vehicle_inspection", "parameters": {"registration_num": "DEF456", "vehicle": "truck", "inspection_date": "June 20, 2022"}, "responses": {"appointment_confirmation": "API_call_4"}}]
Qwen2-7B	Load the VR environment named "Fantasy Land" for user ID 12345. Obtain information about the user's avatar named "Wizard". Explore the VR world within the "Fantasy Land" environment, starting at coordinates (10, 5, 3) and interact with other users present.	[{"api_name": "load_vr_environment", "parameters": {"environment_name": "Fantasy Land", "user_id": 12345}, "responses": {"environment_type": "Fantasy", "user_preferences": "adventure", "avatar_skin_color": "blue"}}, {"api_name": "get_avatar_info", "parameters": {"user_ID": 12345, "avatar_name": "Wizard"}, "responses": {"avatar_height": 1.8, "avatar_accessories": "pointed hat, staff", "avatar_bio": "A powerful wizard who specializes in elemental magic."}, {"api_name": "explore_vr_world", "parameters": {"environment_type": "Fantasy", "user_coordinates": [10, 5, 3], "interact_with_others": true}, "responses": {"discovered_elements": ["ancient ruins", "enchanted forest", "dragon's lair"], "interactive_users": 5}}]

Table 15: Error examples for generation format.