ToolHop: A Query-Driven Benchmark for Evaluating Large Language Models in Multi-Hop Tool Use

Junjie Ye[1,2], **Zhengyin Du**[2], **Xuesong Yao**[2], **Weijian Lin**[2], **Yufei Xu**[2], **Zehui Chen**[2], **Zaiyuan Wang**[2], **Sining Zhu**[2], **Zhiheng Xi**[1], **Siyu Yuan**[1], **Tao Gui**[1,3*], **Qi Zhang**[1,3*], **Xuanjing Huang**[1,3*], **Jiecao Chen**[2]

[1] Fudan University [2] ByteDance

[3] Shanghai Collaborative Innovation Center of Intelligent Visual Computing

jjye23@m.fudan.edu.cn, {qz, tgui}@fudan.edu.cn

arXiv:2501.02506v4 [cs.CL] 20 May 2025

Abstract

Effective evaluation of multi-hop tool use is critical for analyzing the understanding, reasoning, and function-calling capabilities of large language models (LLMs). However, progress has been hindered by a lack of reliable evaluation datasets. To address this, we present *ToolHop*, a dataset comprising 995 user queries and 3,912 associated tools, specifically designed for rigorous evaluation of multi-hop tool use. ToolHop ensures diverse queries, meaningful interdependencies, locally executable tools, detailed feedback, and verifiable answers through a novel query-driven data construction approach that includes tool creation, document refinement, and code generation. We evaluate 14 LLMs across five model families (i.e., LLaMA3.1, Qwen2.5, Gemini1.5, Claude3.5, and GPT), uncovering significant challenges in handling multi-hop tool-use scenarios. The leading model, GPT-4o, achieves an accuracy of 49.04%, underscoring substantial room for improvement. Further analysis reveals variations in tool-use strategies for various families, offering actionable insights to guide the development of more effective approaches. Code and data can be found in https://huggingface.co/datasets/bytedance-research/ToolHop.

1 Introduction

The task of multi-hop tool use presents a significant challenge for large language models (LLMs) (OpenAI, 2023; Touvron et al., 2023; Bai et al., 2023). As illustrated in Figure 1, it requires LLMs to incrementally decompose a complex multi-hop query into atomic subqueries, invoke the appropriate tools, and iteratively retrieve results from the tool feedback until the final answer is reached. This process demands advanced capabilities such as comprehension, reasoning, and function-calling (Qin et al., 2023; Qu et al., 2024), making the evaluation of multi-hop tool use essential for assessing these skills. Furthermore, such evaluations are pivotal for advancing LLMs toward generalized intelligence (Xi et al., 2023).

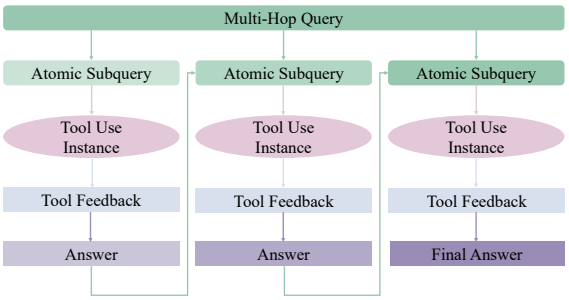

Figure 1: An illustration of multi-hop tool use. The process entails decomposing a complex multi-hop query into multiple atomic sub-queries, sequentially invoking the appropriate tools, retrieving results from the tool feedback, and iterating until the final answer is derived. This demonstrates the integration of comprehension, reasoning, and function-calling capabilities.

Existing studies have made progress in evaluating tool use of LLMs. Some focus on evaluating single-step tool use in simulation environments, requiring manual calibration of correct tool-call results (Chen et al., 2024; Ye et al., 2024a,b). Others examine the process of tool use, leveraging advanced models like GPT-4 to go beyond single-step evaluations and providing some valuable insights (Qin et al., 2024; Huang et al., 2024b; Ye et al., 2025).

However, these works still fall short of offering a reliable evaluation of multi-hop tool use. Specifically, a key limitation of prior work lies in their reliance on tool-driven data construction methods, where a collection of tools is gathered and queries are simulated for them (Tang et al., 2023; Wu et al., 2024; Liu et al., 2024). This approach fails to ensure that the collected tools are interdependent or that the queries involve genuine

*Corresponding authors.

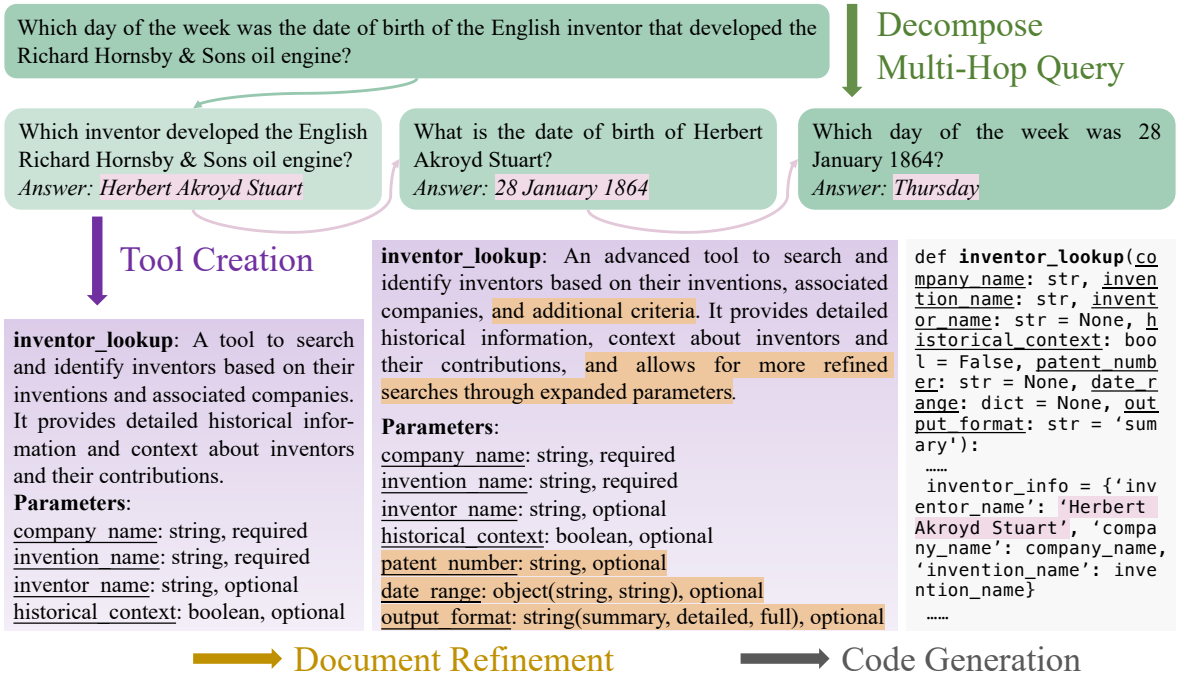

Figure 2: An illustration of our proposed query-driven data construction scheme, comprising three key processes: tool creation, document refinement, and code generation. This approach incrementally develops detailed tool document and code implementation for each atomic subquery within a multi-hop query.

multi-hop reasoning. Furthermore, the absence of verifiable answers forces these studies to depend on process analysis using models, introducing model bias and evaluation errors (Guo et al., 2023; Eloundou et al., 2024).

To address these challenges, we introduce *ToolHop*, a novel dataset specifically designed to evaluate LLMs' multi-hop tool use capabilities. ToolHop comprises 995 multi-hop queries and 3,912 locally executable tools, constructed using a query-driven data construction scheme. This methodology involves tool creation, document refinement, and code generation, which can expand a single multi-hop query into a comprehensive multi-hop tool use test case. An analysis of ToolHop demonstrates its effectiveness in accommodating diverse queries, ensuring meaningful interdependencies, supporting locally executable tools, and delivering detailed feedback alongside verifiable answers. This design rigorously evaluates LLMs' multi-hop tool use capabilities.

We evaluate ToolHop on 14 LLMs from five different families (i.e., LLaMA3.1 (Team, 2024a), Qwen2.5 (Team, 2024b), Gemini1.5 (Reid et al., 2024), Claude3.5 (Bai et al., 2022), and GPT (OpenAI, 2023)). Our results reveal that while tools significantly improve model performance, even the top-performing model, GPT-4, achieves only 49.04% accuracy in multi-hop tool use, highlighting considerable room for improvement. Further studies reveal that different model families exhibit distinct patterns in tool use, leading to varied outcomes. For instance, the Qwen2.5 (Team, 2024b) family of models tends to emphasize parallel calls, which results in hallucinations, while the GPT family leverages tool feedback to improve their performance in tool usage. These insights provide valuable guidance for developing more effective methods.

Our contributions are as follows:

- We introduce ToolHop, a test set of 995 multi-hop queries with 3,912 locally executable tools, designed to assess LLMs' ability to use tools in multi-hop scenarios. It ensures diverse queries, meaningful interdependencies, locally executable tools, detailed feedback, and verifiable answers.

- We propose a novel query-driven data construction process that can expand queries into multi-hop tool use data via tool creation, document refinement, and code generation.

- We provide a comprehensive evaluation of 14 LLMs, identifying significant limitations in current tool-use capabilities and offering

insights for future improvements.

2 ToolHop

In this section, we introduce ToolHop in detail. Specifically, we first provide a formal definition of multi-hop tool use (Section 2.1), then explain our proposed query-driven data construction scheme (Section 2.2), and finally analyze the quality of the ToolHop dataset (Section 2.3).

2.1 Task Formulation

Given a multi-hop query q, which can be decomposed into subqueries $q_1, q_2, ..., q_l$; and a collection of tools $\mathbb{T} = (t_1, t_2, \ldots, t_l)$, where each tool t_i is defined by a document doc_i and a code implementation fun_i, the description document doc_i includes the tool name n_i, a function description d_i, and the corresponding parameters $p_i = (p_i^1, p_i^2, \ldots, p_i^k)$.Each parameter p_i^j is characterized by its name np_i^j, a description dp_i^j, its type tp_i^j, and whether it is required rp_i^j. The goal of multi-hop tool use is for the model $\mathcal{M}$ to utilize the information in $\mathbb{T}$ to sequentially invoke the appropriate tools $t_1, t_2, ..., t_l$, where each tool t_i is used to solve subquery q_i and produce an intermediate answer a_i. The output a_i then serves as one of the inputs to the next tool t_{i+1}, enforcing the need for the model to correctly understand complex dependencies between tools and accurately decompose the original query. The final answer a is obtained after executing the full sequence of tool calls. A visual illustration of this process is provided in Figure 1.

2.2 Query-Driven Data Construction

As illustrated in Figure 2, we propose a novel query-driven data construction scheme that departs from traditional tool-driven approaches. This scheme comprises three key stages that involves tool creation, document refinement, and code generation. Given a multi-hop user query q, the scheme extends q to produce a sequence of corresponding tool documents $\text{doc}_{i..l}$ and their associated code implementations $\text{fun}_{i..l}$.

Tool Creation The query-driven data construction begins with the multi-hop user query q, which serves as the foundation for building dynamic tools. The tool creation process accepts q and generates a preliminary set of tool documents $\text{doc}'_{1..l}$. These documents are designed to be both relevant to q and interdependent.

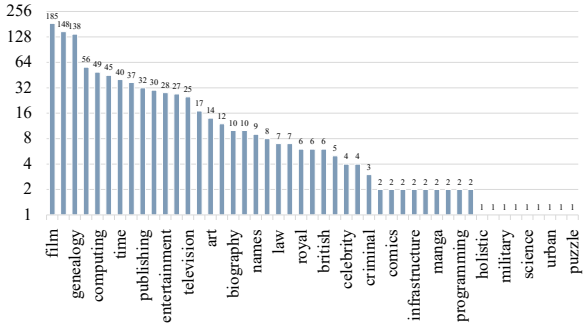

Figure 3: Distribution of user queries across 47 domains in the ToolHop dataset.

# Tools	Three	Four	Five	Six	Seven
# Data	428	353	136	10	68

Table 1: Distribution of the number of tools required to solve each query in ToolHop.

To achieve this, q is decomposed into a sequence of atomic subqueries $q_1, q_2, \ldots, q_l$, where each subquery q_i depends on resolving the preceding ones (i.e., q_{i-1}). For each q_i, a preliminary document doc'_i is created. . These documents not only capture the input-output logic of q_i, but are also structured to generalize to similar queries. By maintaining backward and forward dependencies between documents, this approach ensures both modularity and cohesion, simplifying the tool creation process.

Document Refinement The initial tool documents doc'_i, derived directly from atomic queries, are typically rudimentary due to the limited information in q_i. The document refinement process transforms doc'_i into a more comprehensive document doc_i, designed to better support the evaluation of models in complex multi-hop scenarios.

This process involves two key aspects. On the one hand, the tool's functionality is expanded by introducing features such as result filtering and customizable formats, all while maintaining compatibility with the original functionality. On the other hand, the number of parameters is increased, and their types are optimized. For instance, parameters initially represented as simple strings are replaced with structured types such as arrays or objects, enabling the tools to handle more complex inputs. These refinements ensure that the resulting tool documents are robust, versatile, and capable of addressing intricate cases.

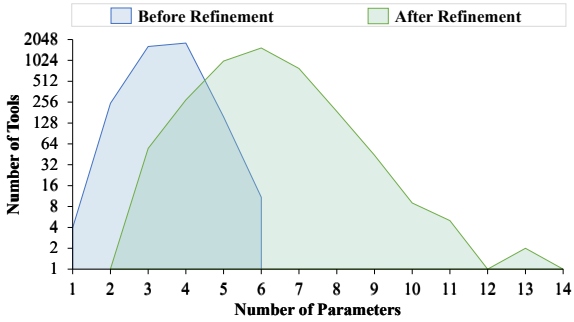

Figure 4: Distribution of the number of tool parameters before and after document refinement.

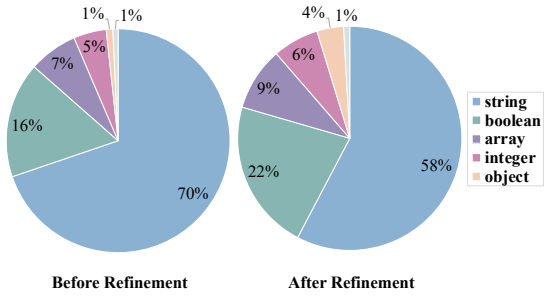

Figure 5: Distribution of tool parameter types before and after document refinement.

Code Generation Once refined tool documents doc_i are complete, the code generation process produces corresponding locally executable functions fun_i. These functions allow external invocation of tools, enabling seamless multi-turn interactions between the model and tools.

Code generation systematically maps document information to code. For instance, the tool name in doc_i is converted into the function name, while parameter specifications are used to define the function signature. To ensure the correctness of fun_i, the atomic query q_i and its answer a_i are included as inputs, requiring the function to return a_i when executed with q_i. Additionally, a robust exception-handling mechanism is implemented, enabling tools to provide informative error messages for invalid inputs while maintaining normal operation. Moreover, the generated code is verified to ensure it functions as intended.

Dataset Construction To effectively implement our approach, we draw on queries from the MoreHopQA dataset (Schnitzler et al., 2024), which consists of multi-hop questions that can be decomposed into at least three atomic queries with answers. Using this foundation, we generate 995 user queries and 3,912 corresponding locally executable tools, which collectively form the ToolHop dataset.[1]

2.3 Dataset Analysis

To ensure that the ToolHop dataset rigorously evaluates the multi-hop tool-use capabilities of LLMs, we conduct a comprehensive analysis across five critical dimensions. This analysis validates ToolHop's ability to represent diverse and challenging multi-hop tool-use scenarios.[2]

Diverse Queries Real-world user needs vary widely, requiring an effective LLM to flexibly utilize tools to address queries spanning multiple domains. To evaluate such capabilities, a suitable dataset must encompass queries from a broad range of topics. ToolHop is explicitly designed to prioritize diversity in its multi-hop queries, reflecting real-world scenarios.

To verify this diversity, we use GPT-4o to categorize all queries in ToolHop into distinct domains. Similar categories are merged to ensure clarity and independence. As shown in Figure 3, the categorization reveals that ToolHop spans 47 unique domains, including topics such as movies and television, academic subjects, and family relationships. This broad coverage ensures that ToolHop effectively evaluates LLM performance across diverse query types, enhancing its representativeness and practical applicability for real-world tool-use scenarios.

Meaningful Interdependencies Previous evaluation for tool use (Song et al., 2023; Yang et al., 2023; Ye et al., 2024b; Han et al., 2024) typically assemble tools from disparate sources and then generate user queries for them. However, these approaches fail to account for interdependencies between tools, often producing queries that inadequately represent multi-hop reasoning. To address this limitation, ToolHop employs a novel query-driven framework. It begins by formulating multi-hop queries and subsequently constructs the required tools based on each atomic query. This approach inherently preserves the multi-hop structure of queries and enforces meaningful

[1] Examples of generated documents and code implementations are provided in Appendix G and Appendix H.

[2] A comparison between ToolHop and existing benchmarks related to tool use can be found in Appendix A.

Refinement	Zero	One	Two	Three	Four
Before	2	2433	1250	202	25
After	2	2490	1198	200	22

Table 2: Distribution of the number of required parameters before and after document refinement.

Refinement	string	boolean	array	integer	object	number
Before	4758	2	404	333	24	114
After	4473	2	755	241	44	102

Table 3: Distribution of required tool parameter types before and after refinement.

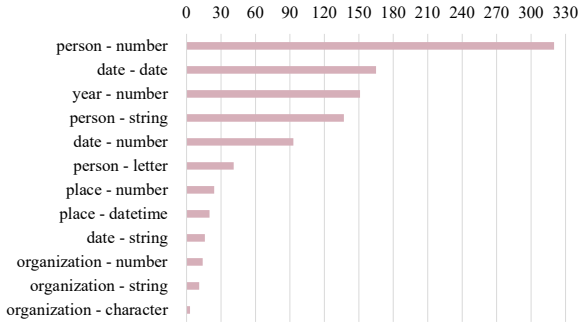

Figure 6: Distribution of answer types for the second atomic subquery and final answers in ToolHop.

interdependencies between tools.

To validate the effectiveness of this approach, we analyze the distribution of tools associated with each query in ToolHop. As shown in Table 1, the number of tools required per query ranges from three to seven, which corresponds to the minimum number of reasoning hops required to resolve the queries, emphasizing the importance of multi-hop reasoning. This distribution underscores the complexity of queries handled by ToolHop and its capability to support scalable multi-hop tool use.

Locally Executable Tools Tools are a core component of the tool use task. ToolHop includes 3,912 locally deployable and directly executable tools, enabling zero-cost invocation and seamless interaction by LLMs. To better align the constructed tools with the diverse requirements of real-world applications, we enhance their complexity through a document refinement process.

Figure 4 shows that the average number of parameters per tool increased from 3.49 to 5.91 after refinement. This reflects an intentional shift toward more expressive tools, which better capture the complexity of real-world tasks. Concurrently, Figure 5 illustrates a 12% reduction in simple string parameters, replaced by more structured types such as arrays, booleans, and objects, which enable richer and more precise tool interactions. Table 2 and Table 3 further demonstrate that the refinement process preserves the number and types of required parameters while increasing the diversity of optional parameters.

Detailed Feedback Effective multi-turn interaction between LLMs and tools requires not only correct outputs for valid inputs but also meaningful error messages for invalid ones. Our approach incorporates two key strategies to address this need.

On the one hand, we include atomic queries and their corresponding answers as part of the input during code generation, ensuring tools reliably produce correct outputs for solvable problems. On the other hand, we integrate robust exception-handling mechanisms into the generated code. Since the tools are locally executable, we can validate LLM-generated call instances using a compiler, providing detailed error reports and feedback to guide subsequent interactions.

Verifiable Answers A key limitation of earlier tool-driven datasets is the absence of predetermined answers, which makes validation difficult. ToolHop overcomes this issue by predefining both queries and answers, enabling straightforward comparison with model outputs.

To ensure verifiability, we analyze the answer types for the second atomic subquery and the final query, which is presented in Figure 6. The result demonstrates that ToolHop supports diverse and flexible answer types while standardizing final answers into objective entities. This design simplifies validation, enhances robustness, and enables consistent performance evaluation.

3 Experimental Setup

We use ToolHop to evaluate the representative families of LLMs for multi-hop tool use. In this section, we introduce the families of LLMs evaluated (Section 3.1) and describe the implementation details of our experiments (Section 3.2).

3.1 Models

We use ToolHop to evaluate 14 LLMs from five families, including **LLaMA3.1-Instruct-8B** and **LLaMA3.1-Instruct-70B** from the LLaMA3.1 family, **Qwen2.5-Instruct-7B**, **Qwen2.5-Instruct-14B**, **Qwen2.5-Instruct-32B**, and **Qwen2.5-Instruct-72B** from the Qwen2.5 fmaily,

Source	Family	Version	Answer Correctness (↑)			Invocation Error (↓)	
			Direct	Mandatory	Free	Query	Instance
		Avg.	*19.83*	*32.12*	*32.84*	*18.72*	*8.68*
Open-Source	LLaMA3.1	Instruct-8B	13.17	12.76	13.47	41.61	21.10
		Instruct-70B	18.79	19.10	12.76	35.08	14.24
	Qwen2.5	Instruct-7B	11.46	9.85	16.18	28.84	7.09
		Instruct-14B	17.39	26.38	26.13	15.78	6.82
		Instruct-32B	20.00	25.03	22.61	12.46	3.46
		Instruct-72B	17.89	45.43	38.29	13.27	4.93
Closed-Source	Gemini1.5	flash-002	18.59	29.35	32.76	13.59	6.69
		pro-002	18.89	31.16	33.07	14.57	6.61
	Claude3.5	Haiku	36.08	38.09	44.72	23.48	15.81
		Sonnet	27.14	39.90	45.23	19.60	15.83
	GPT	3.5-Turbo	17.09	35.38	36.58	11.76	6.03
		4o-mini	19.40	40.20	43.42	11.66	3.58
		4-Turbo	18.59	47.94	46.83	10.95	4.97
		4o	23.12	49.04	47.74	9.45	4.31

Table 4: Performance of various LLMs on ToolHop, including answer correctness and invocation error. 'Direct,' 'Mandatory,' and 'Free' denote the direct answer, mandatory tool use, and free choice scenarios, respectively. 'Query' and 'Instance' refer to the percentage of queries and tool invocation instances with errors, respectively. '*Avg.*' represents the average across all LLMs. Values above the average are highlighted in teal, and those below are highlighted in maroon, with darker shades indicating larger deviations.

Gemini1.5-flash-002 and **Gemini1.5-pro-002** from the Gemini1.5 family, textbfClaude3.5-Haiku and **Claude3.5-Sonnet** from the Claude3.5 family, and **GPT-3.5-Turbo**, **GPT-4o-mini**, **GPT-4-Turbo**, and **GPT-4o** from the GPT family.[3]

3.2 Implementation Details

In the data construction stage, we use GPT-4o to assist with processing.[4] For evaluation, open-source LLMs are tested using their chat templates with greedy decoding, while closed-source LLMs are evaluated via their APIs with a temperature setting of 0. To ensure consistency across evaluations, all tools are implemented through the models' function call interfaces.

4 Main Results

In this section, we present the key evaluation dimensions (Section 4.1) and observations (Section 4.2).

4.1 Evaluation Dimensions

Evaluating the capabilities of LLMs requires a comprehensive approach that assesses both their ability to provide correct answers and their effectiveness in invoking external tools. We analyze these dimensions through answer correctness and invocation error.

Answer Correctness For the accuracy of LLM responses, our query-driven data construction scheme enables direct comparison with predefined standard answers. We consider three evaluation scenarios: the **direct answer** scenario, where LLMs solve queries independently without external tools; the **mandatory tool use** scenario, where models are required to use provided tools extensively to maximize their tool-use capabilities; and the **free choice** scenario, where external tools are available but optional, allowing LLMs to balance independent problem-solving with tool use.

Invocation Error In the mandatory tool use scenario, we assess errors made when invoking tools, leveraging detailed feedback for each tool to identify errors. We focus on three types: tool hallucination, where models invoke tools not included in the provided toolset; parameter hallucination, where unprovided parameters are used for a given tool; and parameter missing,

[3]More details can be found in Appendix D.
[4]Prompts are detailed in Appendix B.

where required parameters for a tool are omitted. Errors are quantified from the percentage of queries containing incorrect calls relative to **total queries**, and the percentage of incorrect tool invocations relative to **all tool use instances**.

4.2 Evaluation Observations

From the results presented in Table 4, we can make several notable observations.[5]

While LLMs have significantly enhanced their ability to solve complex multi-hop queries with the use of tools, their multi-hop tool use capabilities still leave considerable room for improvement. Comparing the direct answer scenario (i.e., Direct) versus the mandatory tool use scenario (i.e., Mandatory), we observe that the use of tools increases LLMs' answer correctness by an average of 12.29%. Notably, the GPT family of models improves its accuracy by an average of 23.59% through tool use, underscoring how effective tool-use capabilities enhance their performance in solving complex multi-hop problems. Despite these improvements, the overall accuracy in the mandatory tool use scenario remains limited. Even the best-performing model, GPT-4o, achieves only 49.04% answer correctness in this scenario. Furthermore, 9.45% of queries exhibit hallucinations. The performance of LLaMA3.1-Instruct-8B reveals further challenges, with over 40% of queries containing invocation errors, underscoring the need for better documentation understanding.

The performance of different LLM families indicates that most are optimized for tool use, but they exhibit distinct characteristics when solving multi-hop queries. In both the mandatory tool use scenario and the free choice scenario (i.e., Free), LLMs generally opt to use tools, with answer correctness in these two conditions differing by only 0.62%. This indicates that most LLMs are specifically optimized for tool use. However, different LLM families show varying strengths in their tool use. For instance, Qwen2.5-Instruct-72B improves its answer correctness by 27.54% through tool use, while the Claude3.5 family excels in the direct answer scenario without tool reliance. The underlying reasons for these differences are explored in depth in Section 5.

Examining the performance of different versions within each LLM family, larger models generally demonstrate better tool use to meet user needs, aligning with the scaling law (Kaplan et al., 2020; Chung et al., 2022)**.** Both open-source and closed-source LLMs show an increase in answer correctness and a decrease in invocation error in the mandatory tool use scenario as model size grows. Notably, the correlation between invocation errors and answer correctness is stronger at the query level than at the instance level, suggesting that invocation errors in specific queries significantly impair problem-solving. Interestingly, this pattern enables the inference of relative model sizes within families. For instance, based on performance patterns, GPT-4o is likely a larger and more advanced version compared to other models in the GPT family.

5 Further Studies

From the results in Section 4.2, we observe significant variation in the performance across different families of LLMs. To further investigate these differences, we analyze each family in detail and present the following key observations.[6]

The LLaMA3.1 and Gemini1.5 families perform poorly in multi-hop tool use scenarios compared to other LLMs from the same source, primarily due to their incomplete support for tool use capabilities. In the case of LLaMA3.1, the inability to output both natural language text and tool call instances simultaneously restricts its capacity to perform chain-of-thought (CoT) (Wei et al., 2022) reasoning during tool use, hampering its understanding and analysis of user intent. On the other hand, the Gemini1.5 family of models lack support for union-type parameters, which prevents them from handling tool lists that include complex parameter structures. This limitation significantly reduces their effectiveness in such scenarios.

The enhancement of the Qwen2.5 family with parallel tool calls introduces a trade-off between efficiency and accuracy. Compared to the LLaMA3.1 family, the Qwen2.5 family has improved its ability to utilize tools, particularly with the addition of parallel invocation, which is intended to increase the problem-solving efficiency. However, in multi-hop tool use scenarios, forcing parallel invocation without first processing the results of previous tool calls leads to hallucinations in parameter value assignments, resulting in incorrect answers. For instance, in the mandatory tool

[5]Model performance under various inference frameworks can be found in Appendix E.

[6]Examples illustrating these observations can be found in Appendix F.

use scenario, the percentage of queries involving parallel tool calls is 70.1% for Qwen2.5-Instruct-14B and even higher at 75.08% for Qwen2.5-Instruct-32B, contributing to their relatively poor performance. In contrast, Qwen2.5-Instruct-72B reduces the percentage of parallel calls to just 3.82%, significantly improving its performance.

The optimization of CoT reasoning in the Claude family of models gives them a distinct advantage in the direct answer scenario. Even without explicit CoT prompts, the Claude3.5 family of models independently adopt a step-by-step CoT approach to decompose user queries and generate answers. This method significantly improves their accuracy compared to other LLMs in such scenarios. For instance, in the direct answer scenario, Claude3.5-Haiku applies CoT reasoning to 64.92% of queries, while Claude3.5-Sonnet does so for 8.5%. Additionally, the Claude3.5 family of models do not fully rely on the answers returned by tools. This allows them to produce correct responses using their own internal knowledge when tool invocations lead to errors. Despite a relatively high tool invocation error rate, this ability explains why overall answer correctness remains high.

The GPT family of models demonstrates some ability to correct tool call behavior after an error occurs, but this heavily depends on the level of detail in the feedback provided. Leveraging our query-driven data construction process, we offer detailed feedback when a tool call fails. We calculate the percentage of queries with call errors in the mandatory tool use scenario where the GPT family of models ultimately provide the correct answer. We compare this to the percentage of correct answers when only minimal feedback is given, such as a simple hint indicating the call failed (e.g., 'Failed!'). As shown in Table 5, the GPT family of models exhibit a significant improvement in performance when detailed feedback is provided, successfully correcting their behavior to arrive at the correct answer. However, when only basic error hints are provided, the correctness of their final answers drops by 20.66%. This highlights not only the importance of detailed feedback but also the challenges in further enhancing the models' correction capabilities.

Based on these observations, we propose the following recommendations to enhance the model's tool use capabilities in the future: 1) Develop a robust and adaptable tool-use model that can support a wide range of complex tools; 2) Optimize the model's parallelism and other capabilities while prioritizing improvements in its understanding of user intent to avoid potential negative effects; and 3) Investigate effective strategies for leveraging rich tool feedback to enhance the model's error correction abilities.

Version	w/ Feedback	w/o Feedback	$\Delta_{C\to I}$	$\Delta_{I\to C}$
3.5-Turbo	36.75	21.37	20.51	5.13
4o-mini	38.53	11.93	29.36	2.75
4-Turbo	29.31	12.07	17.24	0.00
4o	47.87	24.47	25.53	2.13

Table 5: Answer correctness of the GPT family of models in queries containing invocation error. 'w/ Feedback' and 'w/o Feedback' represent cases where detailed feedback or only simple error reporting is provided, respectively. '$\Delta_{C\to I}$' denotes the proportion of correct answers that become incorrect, while '$\Delta_{I\to C}$' represents the proportion of incorrect answers that become correct, when transitioning from detailed feedback to simple error reporting.

6 Related Works

LLMs in Tool Use The use of tools is a prominent hallmark of biological intelligence (Shumaker et al., 2011). Equipping LLMs with the ability to use tools is therefore a key milestone in advancing their capabilities toward artificial general intelligence (Ye et al., 2023; Xi et al., 2023). Tools broadly encompass APIs, online services, application software, and other models that can be represented in formats accessible to LLMs (Qin et al., 2023). A critical factor in enhancing tool-use performance is constructing extensive datasets that detail tool use (Tang et al., 2023; Liu et al., 2024). This involves generating diverse user queries and their corresponding tool sets. Existing approaches often employ a tool-driven methodology, collecting tools from various sources and using models to simulate user queries (Zhuang et al., 2023; Yu et al., 2024). However, these methods lack diversity, fail to ensure dependency consistency, and cannot reliably verify data correctness. In this paper, we propose a query-driven data construction approach. This method extends the range of locally executable tools through multi-hop queries, improving dataset quality and better supporting the development of LLM tool-use capabilities.

Evaluation of Tool Use Effectively evaluating the tool-use capabilities of LLMs is crucial for

identifying their strengths and weaknesses. Existing methods, such as manual verification (Tang et al., 2023) or checking for the presence of a final answer (Qin et al., 2024), fall short in providing objective and reliable measures of performance. Multi-dimensional approaches (Ye et al., 2025, 2024a) attempt to evaluate the process and outcomes of tool use but risk introducing model bias and inconsistencies. In this paper, we focus on evaluating LLMs in multi-hop tool use scenarios. Our query-driven data construction scheme predefines verifiable answers, ensuring accurate assessments and providing a robust framework for evaluation.

7 Conclusion

In this paper, we introduce ToolHop, a novel dataset designed to evaluate LLMs in multi-hop tool use. ToolHop employs a query-driven data construction framework, encompassing tool creation, document refinement, and code generation. This approach overcomes the limitations of previous methods, ensuring diverse queries, meaningful interdependencies, locally executable tools, detailed feedback, and verifiable answers. Using ToolHop, we benchmark 14 LLMs across five families, providing a comprehensive evaluation of their tool-use capabilities. Further studies illuminate the distinct characteristics of different LLM families, offering actionable insights to enhance their performance. By setting a robust standard for multi-hop tool use evaluation, ToolHop lays the groundwork for advancing LLMs' ability to perform complex tool-based reasoning tasks.

Limitations

While our dataset effectively evaluates the performance of LLMs in multi-hop tool use, one limitation of this work is the lack of an immediate strategy for enhancing these capabilities. Nonetheless, the scalability of our data construction scheme represents a significant advantage, as it can be readily adapted to create training datasets aimed at addressing this challenge. We hypothesize that targeted training using such datasets could markedly improve the ability of LLMs to perform multi-hop tool use tasks. Additionally, we provide a detailed analysis of current tool-use characteristics in LLMs, offering valuable insights that can serve as a foundation for future research and advancements in this area.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by National Natural Science Foundation of China (No. 62476061,62206057,62076069), Shanghai Rising-Star Program (23QA1400200), Natural Science Foundation of Shanghai (23ZR1403500), Program of Shanghai Academic Research Leader under grant 22XD1401100.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *CoRR*, abs/2309.16609.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosiute, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemí Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022. Constitutional AI: harmlessness from AI feedback. *CoRR*, abs/2212.08073.

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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9510–9529. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun

Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. Anytool: Self-reflective, hierarchical agents for large-scale API calls. *CoRR*, abs/2402.04253.

Tyna Eloundou, Alex Beutel, David G. Robinson, Keren Gu-Lemberg, Anna-Luisa Brakman, Pamela Mishkin, Meghan Shah, Johannes Heidecke, Lilian Weng, and Adam Tauman Kalai. 2024. First-person fairness in chatbots. *CoRR*, abs/2410.19803.

Zishan Guo, Renren Jin, Chuang Liu, Yufei Huang, Dan Shi, Supryadi, Linhao Yu, Yan Liu, Jiaxuan Li, Bojian Xiong, and Deyi Xiong. 2023. Evaluating large language models: A comprehensive survey. *CoRR*, abs/2310.19736.

Han Han, Tong Zhu, Xiang Zhang, Mengsong Wu, Hao Xiong, and Wenliang Chen. 2024. Nestools: A dataset for evaluating nested tool learning abilities of large language models. *CoRR*, abs/2410.11805.

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. 2024a. Planning, creation, usage: Benchmarking llms for comprehensive tool utilization in real-world complex scenarios. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 4363–4400. 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, and Lichao Sun. 2024b. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. *Neural Comput.*, 3(1):79–87.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *CoRR*, abs/2001.08361.

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, EMNLP 2023, Singapore, December 6-10, 2023*, pages 3102–3116. Association for Computational Linguistics.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. 2024. Toolace: Winning the points of LLM function calling. *CoRR*, abs/2409.00920.

Weijie Lv, Xuan Xia, and Sheng-Jun Huang. 2024. Codeact: Code adaptive compute-efficient tuning framework for code llms. *CoRR*, abs/2408.02193.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

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, 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. 2023. Tool learning with foundation models. *CoRR*, abs/2304.08354.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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. *CoRR*, abs/2405.17935.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, Jean-Baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew M. Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, James Molloy, Jilin Chen, Michael Isard, Paul Barham, Tom Hennigan, Ross McIlroy, Melvin Johnson, Johan Schalkwyk, Eli Collins, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Clemens Meyer, Gregory Thornton, Zhen Yang, Henryk Michalewski, Zaheer Abbas, Nathan Schucher, Ankesh Anand, Richard Ives, James Keeling, Karel Lenc, Salem

Haykal, Siamak Shakeri, Pranav Shyam, Aakanksha Chowdhery, Roman Ring, Stephen Spencer, Eren Sezener, and et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR*, abs/2403.05530.

Julian Schnitzler, Xanh Ho, Jiahao Huang, Florian Boudin, Saku Sugawara, and Akiko Aizawa. 2024. Morehopqa: More than multi-hop reasoning. *CoRR*, abs/2406.13397.

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. *CoRR*, abs/2311.18760.

Robert W Shumaker, Kristina R Walkup, and Benjamin B Beck. 2011. *Animal tool behavior: the use and manufacture of tools by animals*. JHU Press.

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. *CoRR*, abs/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. *CoRR*, abs/2306.05301.

Meta Team. 2024a. Introducing llama 3.1: Our most capable models to date.

Qwen Team. 2024b. Qwen2.5: A party of foundation models!

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

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. *CoRR*, abs/2405.08355.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. Gpt4tools: Teaching large language model to use tools via self-instruction. 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*.

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, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, Jie Zhou, Siming Chen, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. A comprehensive capability analysis of GPT-3 and GPT-3.5 series models. *CoRR*, abs/2303.10420.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Tao Ji, Qi Zhang, Tao Gui, and Xuanjing Huang. 2025. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 156–187. Association for Computational Linguistics.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024a. Toolsword: Unveiling safety issues of large language models in tool learning across three stages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 2181–2211. Association for Computational Linguistics.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024b. Rotbench: A multi-level benchmark for evaluating the robustness of large language models in tool learning. In *Proceedings of the 2024 Conference on Empirical*

Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024, pages 313–333. Association for Computational Linguistics.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2024. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *CoRR*, abs/2410.07745.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for LLM question answering with external tools. *CoRR*, abs/2306.13304.

A Comparison of Various Benchmarks

Benchmarks	# Tools	# Instances	Multi-Tool?	Meaningful Interdependencies?	Locally Executable Tools?	Detailed Feedback?	Verifiable Answers?
API-Bank (Li et al., 2023)	73	314	✓	×	✓	✓	✓
ToolAlpaca (Tang et al., 2023)	426	3938	×	×	×	×	×
RestBench (Song et al., 2023)	94	157	✓	×	×	×	✓
ToolBench (Qin et al., 2024)	16464	126484	✓	×	✓	✓	×
MetaTool (Huang et al., 2024b)	199	21127	✓	×	×	×	✓
TaskBench (Shen et al., 2023)	103	23271	✓	×	✓	✓	×
T-Eval (Chen et al., 2024)	15	533	✓	×	✓	✓	✓
ToolEyes (Ye et al., 2025)	568	382	✓	×	✓	✓	×
UltraTool (Huang et al., 2024a)	2032	5824	✓	×	×	×	×
Seal-Tools (Wu et al., 2024)	4076	14076	✓	×	×	×	✓
AnyToolBench (Du et al., 2024)	-	400	✓	×	✓	✓	×
BFCL v3 (multi-turn) (Patil et al., 2023)	-	1000	✓	✓	×	×	✓
ToolHop	3912	995	✓	✓	✓	✓	✓

Table 6: A comparison between ToolHop and existing benchmarks related to tool use.

To illustrate the necessity and advantages of ToolHop, Table 6 presents a comparison between ToolHop and existing benchmarks related to tool use. Most existing benchmarks (e.g., API-Bank) lack meaningful interdependencies between tools. This limitation prevents them from evaluating a model's ability to perform multi-hop tool use, where the output of one tool serves as the input to another. The only benchmark that includes meaningful interdependencies, BFCL v3 (multi-turn), lacks locally executable tools and detailed feedback. Moreover, it only supports models that generate all tool calls at once, rather than allowing multi-turn interactions with the environment. This makes it unsuitable for evaluating dynamic tool use, such as reasoning updates based on feedback.

In contrast, ToolHop is the only benchmark that incorporates all five key features: diverse queries, meaningful interdependencies, locally executable tools, detailed feedback, and verifiable answers. These features make it a comprehensive and realistic dataset for evaluating model performance in multi-hop tool use scenarios.

Additionally, most existing datasets become obsolete within 1–2 months as models quickly surpass their challenges. However, even the most advanced models continue to struggle with ToolHop. To maintain its effectiveness over time, we actively expand the dataset with more difficult tasks, establishing it as a long-term and instructive benchmark for evaluating tool-use capabilities.

B Prompt for Data Construction

Our proposed query-driven data construction scheme involves tool creation, document refinement, and code generation. The prompts used for each process are provided in Table 7, Table 8, and Table 9, respectively.

Identify the appropriate tool to solve the given problem and provide an analysis of the tool design. The output should be in JSON format, following the specified structure.

Steps

1. **Analyze the Problem**: Understand the question and determine the type of information required to answer it.
2. **Tool Design**: Design a tool that can solve the problem, considering the complexity and additional functionalities it might need.
3. **Parameter Specification**: Define the parameters for the tool, ensuring they are comprehensive and flexible for various use cases.
4. **Output Construction**: Format the output in JSON, including both the analysis and the tool schema.

Notes

- Ensure the tool is versatile enough to handle similar queries for different sports figures.
- Consider edge cases.

Output Format

The output should be a JSON object with the following structure **without any other contents**:
- "analysis": A detailed analysis of the ideas behind the tool design.
- "tool": A JSON schema characterizing the tool, including its name, description, and parameters.

Example

{Example}

Question: {Question}

Output:

Table 7: The prompt for tool creation, where '{Example}' and '{Question}' represent the example and subquery, respectively.

Refine the design of a tool by enhancing its description and increasing the complexity of parameters (e.g., numbers and types) while maintaining compatibility with the original functionality.

Steps

1. **Analyze the Current Tool**: Examine the existing tool's description and parameters to understand its functionality and limitations.
2. **Identify Areas for Refinement**: Determine which aspects of the tool can be improved or expanded to better meet real-world requirements.
3. **Refine the Description**: Enhance the tool's description to clearly articulate its refined functionality.
4. **Add and Refine Parameters**: Introduce new parameters or refine existing ones to increase complexity and utility, ensuring they align with the original functionality.
5. **Ensure Compatibility**: Verify that the refined version remains compatible with the original tool's purpose and structure.

Output Format

The output should be in JSON format with the following structure **without any other contents**:
{
"analysis": "Analysis of ideas about refining the tool.",
"refined_version": the version after refinement, should be follow JSON SCHEMA format as the original tool
}

Notes

- Ensure that any new parameters added are relevant and enhance the tool's functionality.
- Maintain backward compatibility with the original tool's design and purpose.

Tool:
{Tool}

Table 8: The prompt for document refinement, where '{Tool}' represents the preliminary document.

Create a function implementation based on a provided tool document, question, and answer. The function should strictly adhere to the tool's specifications, including the function name, parameter names, and types. Ensure the function is fully realized and capable of returning different feedback based on the input parameters.

Steps

1. **Understand the Tool Document**: Review the tool document to identify the function name, parameter names, and types.
2. **Analyze the Question and Answer**: Determine how the function should be used to answer the question.
3. **Implement the Function**:
- Use the tool name as the function name.
- Define parameters exactly as specified in the tool document.
- Implement the function logic to produce the correct answer for the given question.
- Simulate additional return values as specified in the tool document.
4. **Error Handling**: Develop a robust error handling mechanism to return valid error messages for incorrect inputs or other issues.

Notes

- Ensure parameter types and names match exactly with the tool document.
- Simulate additional return values as needed based on the tool's documentation.
- Implement comprehensive error handling to cover potential issues.

Output format

Output the result in JSON format with the following structure **without any other contents**:
{ "analysis": "Detailed analysis of how the function was designed, including reasoning for parameter choices and exception handling.",
"function": "The specific function design, including code and comments explaining each part."
}

Tool Document:
{document}

Question: {question}

Answer: {answer}

Table 9: The prompt for code generation, where '{document}', '{question}' and '{answer}' represent the refined document, the subquery and the corresponding answer, respectively.

C Prompt for Domain Classification

We conduct a domain analysis of the queries in ToolHop using GPT-4o, with the corresponding prompts provided in Table 10.

Identify the domain of the given sentence by analyzing its content and context. The domain should be a single, specific category that best describes the subject matter of the sentence.

Steps

1. **Analyze the Sentence**: Break down the sentence to understand its components and context.
2. **Identify Key Elements**: Look for specific terms or phrases that indicate the subject matter, such as names, dates, or specific topics.
3. **Determine the Domain**: Based on the analysis, select the most appropriate domain that encapsulates the main focus of the sentence.

Output Format

The output should be in JSON format with the following structure **without any other contents**:
{
"analysis": "Analysis of the given sentence.",
"domain": The domain of the sentence, as short as possible
}

Notes

- Ensure the domain is specific and directly related to the main subject of the sentence.
- Consider the broader context if the sentence includes specific names or events.

Sentence: {sentence}

Table 10: The prompt for domain classification, where '{sentence}' represents the multi-hop query.

D Details for Models

We evaluate 14 LLMs from five families, spanning both open- and closed-source models, to provide a comprehensive analysis of their performance in multi-hop tool use.

- **LLaMA3.1 Family.** The LLaMA3.1 family, developed by Meta, includes open-source LLMs with model sizes of 8B, 70B, and 405B, and context lengths up to 128K. These models are optimized for tasks such as long text summarization, multilingual dialogue, and code generation. Due to computational constraints, this study evaluates **LLaMA3.1-Instruct-8B** and **LLaMA3.1-Instruct-70B**.

- **Qwen2.5 Family.** The Qwen2.5 family, developed by Alibaba, consists of open-source LLMs pre-trained on 18 trillion tokens. These models are designed to excel in mathematics, programming, and knowledge representation, with versions ranging from 0.5B to 72B. Our evaluation focuses on **Qwen2.5-Instruct-7B**, **Qwen2.5-Instruct-14B**, **Qwen2.5-Instruct-32B**, and **Qwen2.5-Instruct-72B**.

- **Gemini1.5 Family.** The Gemini1.5 family, developed by DeepMind, utilizes a mixture-of-experts (Jacobs et al., 1991) architecture for advanced reasoning across large datasets. This family includes flash and pro versions. For this paper, we analyze **Gemini1.5-flash-002** and **Gemini1.5-pro-002**.

- **Claude3.5 Family.** The Claude3.5 family, developed by Anthropic, includes closed-source Haiku and Sonnet versions, which are known for advancements in instruction-following and nuanced reasoning. This evaluation considers **Claude3.5-Haiku** and **Claude3.5-Sonnet**.

- **GPT Family.** The GPT family, developed by OpenAI, comprises closed-source LLMs designed for text generation, multimodal understanding, and tool use. In this paper, we evaluate **GPT-3.5-Turbo**, **GPT-4o-mini**, **GPT-4-Turbo**, and **GPT-4o**.

E Performance under Various Inference Frameworks

Source	Family	Version	FC	ReAct	CodeAct
Tool-Use	ToolLLaMA-2	7B-v2	-	16.08	-
	TL-CodeLLaMA-2	7B	-	-	17.59
Open-Source	LLaMA3.1	Instruct-8B	13.47	38.59	25.63
		Instruct-70B	12.76	57.79	44.19
	Qwen2.5	Instruct-7B	16.18	40.50	22.91
		Instruct-14B	26.13	46.13	36.78
		Instruct-32B	22.61	48.04	45.73
		Instruct-72B	38.29	48.44	46.63
Closed-Source	Gemini1.5	flash-002	32.76	4.82	5.13
		pro-002	33.07	22.61	7.24
	Claude3.5	Haiku	44.72	34.07	48.34
		Sonnet	45.23	44.52	49.15
	GPT	3.5-Turbo	36.58	28.14	25.63
		4o-mini	43.42	47.14	40.80
		4-Turbo	46.83	53.67	42.61
		4o	47.74	54.17	44.92

Table 11: Performance of various LLMs on ToolHop across different inference frameworks. 'FC' refers to answer correctness in the Free scenario using built-in function-calling templates.

In addition to using the built-in chat template, we also compare the performance of the models under other commonly used tool-use frameworks. Table 11 shows how different models perform with the ReAct (Yao et al., 2023) and CodeAct (Lv et al., 2024) inference frameworks, and compares these results to performance in the Free scenario. The results show that while model performance varies across frameworks, none of them effectively solves the challenges presented by ToolHop in any setting, which further supports our conclusions.

We also observe the following:

- Open-source models benefit more from the ReAct framework, as it enforces a step-by-step reasoning process before tool invocation, which improves planning ability. In contrast, models using FC template often invoke tools directly without explicit reasoning, suggesting that open-source models still have room to improve their built-in function-calling capabilities.

- Closed-source models are generally well-optimized for FC, and their performance remains more stable across different frameworks compared to open-source models. However, different models still show preferences for specific frameworks. For instance, the GPT series performs best with ReAct, while the Claude 3.5 series prefers CodeAct, likely due to differences in training data and methodology.

F Case Study

In Section 4.2 and Section 5, we analyze the performance of different LLMs across various scenarios. In this section, we present relevant examples from Figure 7 to Figure 10.

User
What is the result of substracting 3 from the reverse of the year of birth of the man who conducted the experiment Operation light house with Wernher von Braun?

Qwen2.5-Instruct-32B
{'arguments': '{"project_name": "Operation Light House", "figure_name": "Wernher von Braun", "association_type": "direct", "output_format": "json"}', 'name': 'historical_association_finder'}
{'arguments': '{"name": "Wernher von Braun", "information_type": "birth_year", "output_format": "integer"}', 'name': 'biographical_info_retriever'}
{'arguments': '{"numbers": [1912], "output_format": "integer"}', 'name': 'reverse_number_advanced'}
{'arguments': '{"numbers": [2191, 3], "operation": "subtract", "precision": 0}', 'name': 'advanced_arithmetic_calculator'}
{'arguments': '{"numbers": [2188], "operation": "average", "precision": 0}', 'name': 'advanced_arithmetic_calculator'}

Tool
No associations found for Operation Light House.

Tool
No data found for Wernher von Braun.

Tool
2191

Tool
{"result": 2188}

Tool
At least two numbers are required.

Qwen2.5-Instruct-32B
<answer>2188</answer>

Figure 7: The Qwen2.5 family of LLMs emphasizes parallel tool calls in the mandatory tool use scenario, which can lead to hallucinations and incorrect answers.

User

How many hours are there between the winter time zone of the place of birth of the person who submitted the first version of the equal rights amendment to congress in 1923 and UTC?

Claude3.5-Haiku

Let's solve this step by step:

1. First version of Equal Rights Amendment submitted in 1923 by Alice Paul
2. Alice Paul was born in Mount Laurel, New Jersey
3. New Jersey is in the Eastern Time Zone
4. Eastern Time Zone is UTC-5 during standard time

<answer>5</answer>

Figure 8: The Claude 3.5 family of LLMs optimizes CoT reasoning in the direct answer scenario, enhancing their analytical and problem-solving capabilities."

User
What is the first letter of the first name of the father of the director of film Little Joe (Film) in lowercase?

GPT-4o
{'name': 'movie_director_lookup', 'arguments': '{"movie_title": "Little Joe"}'}

Tool
Jessica Hausner

GPT-4o
{'name': 'family_relationship_finder', 'arguments': '{"individual_name": "Jessica Hausner", "relationship_type": "father"}'}

Tool
an error occured when call family_relationship_finder: family_relationship_finder() missing 1 required positional argument: 'data_source'

GPT-4o
{'name': 'family_relationship_finder', 'arguments': '{"individual_name": "Jessica Hausner", "relationship_type": "father", "data_source": "default"}'}

Tool
Rudolf Hausner

……

GPT-4o
<answer>r</answer>

Figure 9: The GPT family of LLMs improves performance by refining calling behavior through the use of detailed tool feedback.

User
What is the first letter of the first name of the father of the director of film Little Joe (Film) in lowercase?

GPT-4o
{'name': 'movie_director_lookup', 'arguments': '{"movie_title": "Little Joe"}'}

Tool
Jessica Hausner

GPT-4o
{'name': 'family_relationship_finder', 'arguments': '{"individual_name": "Jessica Hausner", "relationship_type": "father"}'}

Tool
Error.

GPT-4o
<answer>error</answer>

Figure 10: The GPT fmaily of LLMs struggles to correct their calling behavior when provided with minimal feedback.

G Examples of Tool Documents

Our query-driven data construction scheme generates preliminary document prior to refinement. Below, we provide examples of documents before and after refinement. As shown, the refinement process enhances the tool's functionality, increases the number of parameters and introduces more diverse parameter types.

```
{
    "name": "album_release_date_finder",
    "description": "A tool designed to find the release date of music
    albums. It queries a database or API to retrieve accurate
    information about album release dates, accommodating variations in
    album titles and artist names.",
    "parameters": {
        "type": "object",
        "properties": {
            "album_name": {
                "type": "string",
                "description": "The name of the album for which the
                release date is being queried."
            },
            "artist_name": {
                "type": "string",
                "description": "The name of the artist or band
                associated with the album, to ensure accuracy in case
                of albums with similar names."
            },
            "output_format": {
                "type": "string",
                "enum": [
                    "date",
                    "text"
                ],
                "description": "The format of the output. Defaults to
                date (the release date in YYYY-MM-DD format)."
            }
        },
        "required": [
            "album_name"
        ]
    }
}
```

```
{
    "name": "album_release_date_finder",
    "description": "An advanced tool designed to find the release date
    of music albums. It queries a comprehensive database or API to
    retrieve accurate information about album release dates,
    accommodating variations in album titles, artist names, album
    versions, release regions, and languages. This tool ensures
    precision and flexibility in retrieving album release information.",
    "parameters": {
        "type": "object",
        "properties": {
```

```
            "album_name": {
                "type": "string",
                "description": "The name of the album for which the
                release date is being queried."
            },
            "artist_name": {
                "type": "string",
                "description": "The name of the artist or band
                associated with the album, to ensure accuracy in case
                of albums with similar names."
            },
            "album_version": {
                "type": "string",
                "description": "The specific version of the album
                (e.g., deluxe, remastered) to refine the search."
            },
            "release_region": {
                "type": "string",
                "description": "The geographical region where the album
                was released, which can affect the release date."
            },
            "language": {
                "type": "string",
                "description": "The language of the album, useful for
                albums released in multiple languages."
            },
            "output_format": {
                "type": "string",
                "enum": [
                    "date",
                    "text"
                ],
                "description": "The format of the output. Defaults to
                date (the release date in YYYY-MM-DD format)."
            }
        },
        "required": [
            "album_name"
        ]
    }
}
```

H Examples of Code Implementations

Our query-driven data construction scheme translates the refined tool document into code, enabling it to function as a locally executable tool. Below, we provide the code implementation of the refined document in Appendix G. The implementation fully realizes the defined functionality, provides valid feedback for correct parameter inputs, and incorporates a robust exception handling mechanism.

```
def album_release_date_finder(album_name: str, artist_name: str =
    '', album_version: str = '', release_region: str = '', language:
    str = '', output_format: str = 'date') -> str:
    """
    Finds the release date of a specified music album.

    Parameters:
    - album_name (str): The name of the album for which the release
    date is being queried.
    - artist_name (str): The name of the artist or band associated with
    the album.
    - album_version (str): The specific version of the album (e.g.,
    deluxe, remastered).
    - release_region (str): The geographical region where the album was
    released.
    - language (str): The language of the album.
    - output_format (str): The format of the output, either 'date' or
    'text'.

    Returns:
    - str: The release date of the album in the specified format.
    """
    # Simulated database/API response
    album_database = {
        'Boy': {
            'artist': 'U2',
            'release_date': '1980-10-20',
            'versions': {
                'standard': '1980-10-20',
                'deluxe': '2008-07-21'
            },
            'regions': {
                'US': '1980-10-20',
                'UK': '1980-10-20'
            },
            'languages': {
                'English': '1980-10-20'
            }
        }
    }

    # Error handling for required parameter
    if not album_name:
        return 'Error: The album_name parameter is required.'

    # Retrieve album information
```

```
    album_info = album_database.get(album_name)
    if not album_info:
        return 'Error: Album not found in the database.'

    # Check artist name if provided
    if artist_name and album_info['artist'] != artist_name:
        return 'Error: Artist name does not match the album record.'

    # Determine release date based on version, region, and language
    release_date = album_info['release_date']
    if album_version:
        release_date = album_info['versions'].get(album_version,
        release_date)
    if release_region:
        release_date = album_info['regions'].get(release_region,
        release_date)
    if language:
        release_date = album_info['languages'].get(language,
        release_date)

    # Format the output
    if output_format == 'text':
        return f'The album "{album_name}" by {album_info["artist"]} was
        released on {release_date}.'
    return release_date

# Example usage
print(album_release_date_finder(album_name='Boy', artist_name='U2',
output_format='date'))  # Output: '1980-10-20'
print(album_release_date_finder(album_name='Boy', artist_name='U2',
output_format='text'))  # Output: 'The album "Boy" by U2 was released
on 1980-10-20.'
```