Can Tool-augmented Large Language Models be Aware of Incomplete Conditions?

Anonymous ACL submission

Abstract

Recent advancements in integrating large language models (LLMs) with tools have allowed the models to interact with real-world environments. However, these *tool-augmented LLMs* often encounter incomplete scenarios when users provide partial information or the necessary tools are unavailable. Recognizing and managing such scenarios is crucial for LLMs to ensure their reliability, but this exploration remains understudied. This study examines whether LLMs can identify incomplete conditions and appropriately determine when to refrain from using tools. To this end, we address a dataset by manipulating instances from two datasets by removing necessary tools or essential information for tool invocation. Our experiments show that LLMs often struggle to identify the absence of information required to utilize specific tools and recognize the absence of appropriate tools. We further analyze model behaviors in different environments and compare their performance against humans. Our research can contribute to advancing reliable LLMs by addressing common scenarios during interactions between humans and LLMs.[1]

1 Introduction

Recently, there has been significant improvement in integrating large language models (LLMs) with tools (Li et al., 2023; Qin et al., 2023; Patil et al., 2023; Schick et al., 2024; Hao et al., 2024). These *tool-augmented LLMs* can perceive up-to-date information, acquire real-world interaction capabilities, and perform complex tasks (Wang et al., 2024), enhancing user experiences across various applications (Yang et al., 2023; Hong et al., 2023).

Despite their capabilities, tool-augmented LLMs often face scenarios where users lack comprehensive knowledge about available tools or the necessary tools are absent. When LLM agents interact with users, they frequently encounter situations with missing tools or incomplete information needed for tool usage. While significant advances have been made in tool-augmented LLMs, existing research has primarily focused on complete scenarios (Huang et al., 2023; Zhang et al., 2024b).

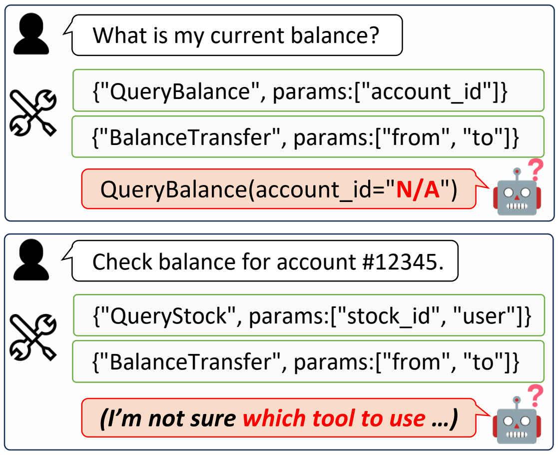

Figure 1: An illustration of incomplete conditions for tool invocation by tool-augmented LLMs.

We investigate whether tool-augmented LLMs can recognize incomplete conditions where no appropriate tool is available or the provided information is insufficient for tool utilization, as depicted in Fig.1. We construct a dataset by manipulating instances from existing tool-augmented LLM datasets (Qin et al., 2023; Li et al., 2023), with human verification ensuring valid instances.

Our experimental results show varying performance patterns across different types of incomplete scenarios: while larger models generally demonstrate strong performance in recognizing tool unavailability, the performance gaps become more pronounced when handling insufficient information for tool usage, particularly in real-world applications. To better understand these capabilities, we conduct comprehensive analyses, including human evaluation and Chain-of-Thought (CoT) prompting (Wei et al., 2022), few-shot learning

[1] Our code and dataset will be publicly available.

approaches, and API call feedback effects. Our findings show that while larger models approach or exceed human-level performance, the effectiveness of enhancement strategies varies across model sizes and tasks.

Our contributions are: (1) We construct a dataset simulating incomplete conditions by manipulating tool-use datasets. (2) We evaluate LLMs' ability to recognize impossible tool invocations, revealing their struggles with identifying missing information. (3) We provide comprehensive analyses of model behavior in incomplete scenarios.

2 Related Work

Recent research has explored the capability enhancement of LLMs with external tools (Wang et al., 2024; Gu et al., 2024; Cai et al., 2024; Fan et al., 2024), ranging from basic retrieval systems (Chen et al., 2017) and arithmetic operations (Schick et al., 2024) to complex programming languages (Gou et al., 2024; Zhang et al., 2024a) and APIs (Xu et al., 2023; Yuan et al., 2024; Guo et al., 2024). Various benchmarks have been developed to evaluate LLMs' tool usage capabilities, including tool selection timing (Huang et al., 2023), robustness to noisy descriptions (Ye et al., 2024b), error handling (Sun et al., 2024), and safety considerations (Ye et al., 2024a).

Although some studies have examined the awareness of LLMs when the necessary tools are not provided, the existing work has significant limitations. Previous studies focused on limited tool scenarios (Ning et al., 2024; Yan et al., 2024), assumed perfect tool utilization plan is given (Huang et al., 2024), or overlooked situations with similar but incorrect tools (Zhang et al., 2024b). Our work addresses these gaps by exploring more realistic scenarios, including situations where incorrect but similar tools are provided or where users fail to supply the necessary information for tool invocation. Although research in question answering has explored handling irrelevant knowledge and ambiguous requests (Feng et al., 2020; Min et al., 2020; Kamath et al., 2020; Cole et al., 2023; Jeong et al., 2024), LLMs' ability to recognize impractical tool usage remains underexplored.

3 How to Evaluate the Awareness of Tool-augmented LLMs

To simulate incomplete scenarios where necessary tools are unavailable or users provide partial information, we manipulate instances from a test set of two benchmarks: APIBank (Li et al., 2023) and ToolBench (Qin et al., 2023). These datasets are designed to evaluate how effectively LLMs can respond to user requests using APIs.[2] In their original dataset, each instance has an available API that can address the user's request, with sufficient information provided to invoke the API. This section describes our data source (§3.1), manipulation strategy (§3.2), and human verification process (§3.3) for creating reliable incomplete scenarios.

3.1 Data Source

We construct a new dataset by leveraging two existing datasets: APIBank (Li et al., 2023) and ToolBench (Qin et al., 2023). APIBank consists of 73 APIs and 314 manually annotated multi-turn conversations. We select 450 instances from the test split, excluding those that require a tool-retrieval module. ToolBench is based on 16,000 real-world APIs across 49 categories from RapidAPI Hub.[3] We utilize 764 test instances as filtered by Guo et al. (2024). These instances are used as the original data for manipulation.

3.2 Simulating Incomplete Scenarios with Instance Manipulation

We simulate two incomplete scenarios in which LLMs cannot properly invoke tools by manipulating the original dataset instances: (1) replacing relevant APIs with irrelevant ones, and (2) partially removing user utterances.

Replacement of Relevant API We use a dense retriever to replace the appropriate APIs with similar but irrelevant ones. This manipulation ensures that the desired tool is unavailable, simulating a scenario where LLMs must decide not to use any of the provided tools. Specifically, the relevant APIs in the original instance are replaced with other APIs from separate API pools. To find semantically similar APIs, we use the sentence encoder proposed by Gao et al. (2021).[4] We concatenate the name and description of each API, convert this text into a fixed-size vector, and then select one of the most similar APIs by calculating cosine similarity between the relevant API and all available APIs in the dataset. Examples of successful and failed instances of this manipulation are in Fig. 4 to Fig. 7.

[2] We refer to Tool and API interchangeably.

[3] `https://rapidapi.com/hub`

[4] `princeton-nlp/sup-simcse-roberta-large`

	API Replacement		Utterance Removal	
	APIBank	ToolBench	APIBank	ToolBench
# Instances	423	477	304	406
Avg. Turns	6.18	1	5.18	1
Avg. APIs	2.13	5.13	2.05	5.51
Avg. Uttr. Words	17.66	52.38	18.87	51.15
Avg. API Length	434.39	713.35	443.38	747.53

Table 1: **Dataset Statistics**. The *API Replacement* and *Utterance Removal* denote *Relevant API Replacement* and *Partial Removal of User Utterance*, respectively. *Avg. API Length* denotes the string length of each API.

Partial Removal of User Utterance We partially remove user utterances in a conversation to mimic scenarios where (1) the request is unclear, making it impossible to use APIs, or (2) the request is clear but lacks necessary information for API invocation. This manipulation removes essential information from user utterances, rendering appropriate API invocation infeasible. We automate this process using a proprietary LLM (i.e., GPT-4 (Achiam et al., 2023)) to generate naturally corrupted dialogues. The model is instructed to identify and remove critical information required for tool invocation, and we enhance the quality of this process with reasoning prompts and five manually designed few-shot samples. Sample instances are in Fig. 8 to Fig. 11.

3.3 Data Verification

To ensure the validity of our dataset, we manually reviewed all instances to remove cases where user requests could still be handled by the provided APIs or where the manipulations resulted in unnatural conversations. Two authors holding bachelor's degree or higher in Computer Science conducted this verification process, focusing on two key aspects: (1) whether alternative or non-replaced APIs could still fulfill the user's request in API Replacement cases, and (2) whether the remaining information was sufficient for API execution in Utterance Removal cases.[5] Through active discussion and refinement of filtering criteria, we ensured that the final dataset includes only instances that genuinely represent incomplete scenarios.

3.4 Dataset Statistics

Our final dataset includes 727 instances from APIBank and 883 from ToolBench, totaling 1,610 instances. Of these, 900 instances were generated by replacing relevant APIs, and 710 were created by removing parts of the user utterances. Table 1 presents the overall dataset statistics.

[5]Detailed dataset verification criteria are in Appendix A.2.

4 Experimental Setup

Task Formulation We set the task where LLMs recognize incomplete conditions as a binary classification problem. We ask the LLM to determine whether APIs can be invoked to fulfill user requests based on conversations and available APIs. To assess whether LLMs can recognize both complete and incomplete conditions, we utilize the instances of two datasets(APIBank and ToolBench) and corresponding manipulated samples. LLMs should answer "Yes" or "No" for the original and manipulated instances, respectively. We measure performance using accuracy and F1 score, with "No" designated as the positive class.

Models Following open-source and proprietary LLMs are used for our experiments: Phi-3-small-8k-instruct (Abdin et al., 2024), Mistral-Instruct-v0.2 (7B) (Jiang et al., 2023), Llama-3.1-8B/80B-Instruct (Dubey et al., 2024), Qwen2-7B-Instruct (Yang et al., 2024), Claude-3-Haiku (Anthropic, 2024), GPT-3.5-Turbo (OpenAI, 2023a), and GPT-4 (Achiam et al., 2023), GPT-4o-mini (Achiam et al., 2023), and GPT-4o (Achiam et al., 2023).[6]

5 Results and Analysis

We present our key observations as follows. Further analyses, such as the impact of tool invocation feedback, are provided in Appendix B.

Model performance varies significantly across manipulation types and datasets. Experimental results in Table 2 show distinct performance patterns. In API Replacement, GPT-4 variants and Llama 3.1 70B achieve strong results (88.82% and 79.10% accuracy on APIBank). The performance gap widens in the Utterance Removal type, especially in ToolBench, where smaller models show significant degradation. Notably, extremely low F1 scores in ToolBench stem from models' predictions concentrating on "Yes" responses, indicating their tendency to overestimate tool applicability and suggesting that Utterance Removal requires more advanced language understanding.

Few-shot learning shows dataset-dependent effectiveness. The impact of few-shot learning varies across datasets. While models generally perform better with few-shot examples on APIBank, improvements are less pronounced in ToolBench. We attribute this to the complexity of real-world tools,

[6]More implementation details are in Appendix A.1.

			Phi-3 small	Mistral 7B	Llama3.1 8B	Llama3.1 70B	Qwen2 7B	Claude3 haiku	GPT-3.5 Turbo	GPT-4	GPT-4o mini	GPT-4o
Relevant API Replacement												
APIBank	Acc.	0-shot	61.60	69.38	76.06	79.10	68.77	68.89	67.92	<u>87.85</u>	**88.82**	86.03
		4-shot	61.97	75.09	75.70	82.50	71.81	65.74	71.32	85.66	85.66	83.35
	F1	0-shot	36.80	62.16	71.32	81.22	58.62	63.53	62.39	<u>87.56</u>	**88.41**	86.23
		4-shot	38.51	69.81	74.81	83.33	63.41	47.58	65.80	84.99	85.99	82.90
ToolBench	Acc.	0-shot	54.84	60.11	69.45	72.64	67.36	68.57	64.29	72.31	77.69	76.48
		4-shot	55.49	60.88	76.37	**80.55**	68.35	58.57	57.91	<u>79.01</u>	75.82	77.14
	F1	0-shot	14.55	34.36	56.43	76.97	52.48	59.03	46.28	61.93	72.97	75.06
		4-shot	17.52	35.04	70.34	**80.49**	60.44	27.36	25.34	<u>77.02</u>	69.36	76.89
Partial Removal of User Utterance												
APIBank	Acc.	0-shot	51.95	52.63	52.63	77.42	57.39	62.65	56.20	<u>89.47</u>	77.25	88.29
		4-shot	62.14	52.63	59.25	84.72	57.22	53.14	65.87	88.79	79.46	**89.81**
	F1	0-shot	4.07	23.14	19.60	80.06	29.69	52.59	36.76	89.27	73.41	<u>88.74</u>
		4-shot	38.23	20.51	46.90	85.98	29.21	9.80	55.43	88.70	79.10	**90.07**
ToolBench	Acc.	0-shot	49.94	50.19	50.19	71.54	51.10	53.17	49.42	57.05	57.57	<u>74.51</u>
		4-shot	50.06	50.45	51.36	72.83	54.98	49.94	50.32	66.11	55.11	**75.16**
	F1	0-shot	0.51	5.41	5.41	**74.94**	10.85	27.89	2.98	27.19	35.18	70.90
		4-shot	1.03	4.49	14.93	70.25	33.59	1.02	2.04	55.74	25.05	<u>73.33</u>

Table 2: Performance evaluation results of LLM by manipulation type. The accuracy (Acc.) and F1 score (F1) are used for evaluation metrics. Both the zero-shot and four-shot performance are presented. The highest and the second-highest scores in each metric are highlighted in **bold** and <u>underlined</u>.

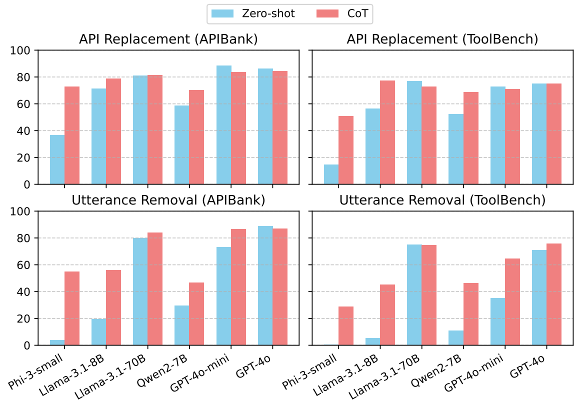

Figure 2: F1-score comparison across different manipulation types and datasets, showing the impact of CoT prompting. Full results are in Table 11.

	API		Utterance		Overall	
	Acc.	F1	Acc.	F1	Acc.	F1
Human	81.11	78.48	81.11	81.32	81.11	80.00

Table 3: **Human analysis results**. API and Utterance denote API Replacement and Utterance Removal.

which require significant reasoning skills and involve diverse scenarios (Khot et al., 2022).

Chain-of-Thought prompting benefits smaller models significantly. As shown in Fig. 2, CoT prompting enhances model performance, particularly for weak models. In APIBank's Utterance Removal task, Phi-3 small's F1 score increases from 4.07% to 38.23% with CoT, suggesting that explicit reasoning steps facilitate a better understanding of API-instruction relationships.

Human performance establishes a reliable benchmark. We conducted a human evaluation with nine participants analyzing 20 balanced instances each from 180 selected APIBank samples. Results in Table 3 show consistent human performance (81.11% accuracy) across manipulation types, with F1 scores from 78.48% to 81.32%. This provides a meaningful reference point for model evaluation, with some larger models demonstrating comparable or superior performance in handling incomplete situations. More details on human evaluations are in Appendix A.3.

6 Conclusion

In this work, we investigate the capability of tool-augmented LLMs to recognize incomplete conditions. Specifically, we examine scenarios where users only provide partial information or the required tools are inaccessible. Our experimental results, based on carefully manipulated datasets, reveal varying performance patterns across different incomplete scenarios and model scales. While larger models show promising results, the effectiveness of enhancement strategies like few-shot learning and Chain-of-Thought prompting varies across different manipulation types and datasets. These findings highlight the importance of considering diverse incomplete scenarios in developing more reliable and safer tool-augmented LLMs for real-world applications.

Limitations

While we explore the ability of tool-augmented LLMs to abstain from tool usage in incomplete conditions, our study has several limitations. First, the data annotation process primarily relies on model-based annotations, although human verification is conducted. This approach might differ from actual cases where humans provide incomplete information regarding tool usage. Additionally, our focus is solely on API-based tools, which, although significant, do not encompass the full spectrum of tools such as plugins, robotic systems, and other interactive systems. Despite these limitations, our research underscores the importance of developing reliable tool-augmented LLMs, highlighting the need for further advancements in this area.

References

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

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. *Preprint*, arXiv:2305.17126.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Jeremy Cole, Michael Zhang, Daniel Gillick, Julian Eisenschlos, Bhuwan Dhingra, and Jacob Eisenstein. 2023. Selectively answering ambiguous questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 530–543, Singapore. 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 arXiv:2407.21783*.

Zhiting Fan, Ruizhe Chen, Ruiling Xu, and Zuozhu Liu. 2024. BiasAlert: A plug-and-play tool for social bias detection in LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14778–14790, Miami, Florida, USA. Association for Computational Linguistics.

Yulan Feng, Shikib Mehri, Maxine Eskenazi, and Tiancheng Zhao. 2020. "none of the above": Measure uncertainty in dialog response retrieval. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2013–2020, Online. Association for Computational Linguistics.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. *Preprint*, arXiv:2309.17452.

Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivasa, Hugo Latapie, and Yu Su. 2024. Middleware for LLMs: Tools are instrumental for language agents in complex environments. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7646–7663, Miami, Florida, USA. Association for Computational Linguistics.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2024. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

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 and virtual meeting. 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: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations*.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Adaptive-rag: Learning to adapt retrieval-augmented large language models through question complexity. *arXiv preprint arXiv:2403.14403*.

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 arXiv:2310.06825*.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5684–5696, Online. Association for Computational Linguistics.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

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.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2020. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5783–5797, Online. Association for Computational Linguistics.

Kangyun Ning, Yisong Su, Xueqiang Lv, Yuanzhe Zhang, Jian Liu, Kang Liu, and Jinan Xu. 2024. Wtu-eval: A whether-or-not tool usage evaluation benchmark for large language models. *Preprint*, arXiv:2407.12823.

OpenAI. 2023a. Chatgpt: A large language model developed by openai. `https://www.openai.com/chatgpt`. Accessed: 2024-06-16.

OpenAI. 2023b. Prompt engineering. `https://platform.openai.com/docs/guides/prompt-engineering`. Accessed: 2024-06-16.

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

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

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

Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. 2024. Tools fail: Detecting silent errors in faulty tools. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14272–14289, Miami, Florida, USA. Association for Computational Linguistics.

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

Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. 2024. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

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

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 arXiv:2407.10671*.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

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*.

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)*, pages 2181–2211, Bangkok, Thailand. Association for Computational Linguistics.

Junjie Ye, Yilong Wu, Songyang Gao, 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. *arXiv preprint arXiv:2401.08326*.

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

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024a. CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13643–13658, Bangkok, Thailand. Association for Computational Linguistics.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, Tetsuya Sakai, Tian Feng, and Hayato Yamana. 2024b. ToolBeHonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11388–11422, Miami, Florida, USA. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

A Experimental Details

A.1 Implementation Details

All open-source LLMs are implemented with the Transformers library (Wolf et al., 2020). For proprietary models, we use `claude-3-haiku-20240307`, `gpt-3.5-turbo-0125`, `gpt-4-0613`, `gpt-4o-mini-2024-07-18`, and `gpt-4o-2024-11-20` for Claude-3-Haiku, GPT-3.5-Turbo, GPT-4, GPT-4o-mini, and GPT-4o, respectively. The temperature is set to 0 across all models. We use two original and two manipulated instances to evaluate with few-shot examples. In experiments with instances from ToolBench, we select models with a context length exceeding 8192 tokens to handle the frequently encountered lengthy API descriptions.

A.2 Detailed Dataset Verification Process

Our preliminary analysis revealed that automatic generation sometimes failed to create truly incomplete scenarios, making manual verification crucial for maintaining dataset quality. To resolve this, we conducted the dataset verification process involved rigorous criteria for both manipulation types. For *Relevant API Replacement*, we removed instances where:

- The user's request could be fulfilled using alternative APIs
- The request could be handled using non-replaced APIs, particularly in ToolBench instances

For *Partial Removal of User Utterance*, we filtered out cases where:

- Only non-essential information was removed, allowing APIs to still fulfill the request
- One of multiple requests was removed, but remaining requests could be resolved through existing API invocations
- The conversation context became unnatural or grammatically incorrect
- The result was identical to the original instance
- The last utterance no longer pertained to the user, making API execution unnatural

Two authors independently reviewed the instances, discussing and resolving any disagreements to ensure consistent application of these criteria. This process was iteratively refined until we achieved a high level of agreement on the identification of truly incomplete scenarios.

A.3 Human Analysis

We conducted a human analysis on the same tasks to establish a performance baseline for LLM evaluation. Specifically, we selected 180 instances from the APIBank dataset, ensuring class balance. Nine participants with relevant academic backgrounds were recruited to solve "Yes/No" questions for 20 instances each, following the same evaluation protocol applied to the LLMs. Each participant was compensated with $10 for their participation.

B Further Analysis

We perform analyses to understand how LLMs perceive and respond in incomplete conditions.

B.1 Implicit Evaluation Results with Free-form Generation

As described in Section 4, our experimental setup involves a binary classification task in which models are asked to determine whether the current conditions are incomplete for tool invocation by answering "Yes" or "No." This approach directly assesses the models' ability to evaluate whether the necessary conditions for tool usage are met. We also consider an alternative method that implicitly gauges the models' awareness to complement our explicit evaluation. In this setup, the models engage in a simulated dialogue with a user, and their next actions are monitored. If the model generates a response that includes invoking a tool, it is considered the model has determined that it can invoke the tool. In contrast, the absence of tool invocation is considered that the model has determined that it cannot invoke the tool. We apply this implicit evaluation framework to GPT-3.5-Turbo and GPT-4 on the APIBank dataset.

The results presented in Table 4 indicate that models generally perform worse in implicit evaluation setups. For example, GPT-4's F1 scores drop from 87.30 and 89.27 to 77.18 and 86.75 for API Replacement and Utterance Removal, respectively. GPT-3.5-Turbo's performance on Utterance Removal is the exception to this trend, where accuracy and F1 scores improve from 55.94 and 36.63 to 59.70 and 49.90. Based on these results, we hypothesize that the binary classification setup, which explicitly prompts the model to assess whether the current tool list or user information is sufficient for tool invocation, enables the model to recognize better when tool usage is appropriate.

	API Replacement			**Utterance Removal**		
	Acc.	F1	Corr.	Acc.	F1	Corr.
GPT-3.5-T	59.69	52.17	58.75	59.70	49.90	62.66
GPT-4	80.50	77.18	85.22	87.83	86.75	88.32

Table 4: Zero-shot implicit evaluation results of GPT-3.5-Turbo and GPT-4 on the APIBank dataset. The Correlation (Corr.) metric measures the instance-wise agreement between the model's binary classification and free-form generation predictions.

Utterance Removal	**Errors in API Call**	**GPT-3.5 Turbo**	**GPT-4**	**GPT-4o mini**	**GPT-4o**
✓	✓	96.0	86.0	97.0	96.0
✓		7.0	67.0	45.0	79.0
	✓	13.0	99.0	51.0	91.0

Table 5: The results of providing API invocation outcomes to LLMs across different scenarios. Accuracy is used as the evaluation metric, highlighting the differences in how different models handle erroneous API call results and their ability to recognize hallucinations and incomplete conditions.

B.2 Impact of Tool Invocation Feedback on Incomplete Condition Recognition

Our experiment primarily focuses on assessing whether LLMs can recognize incomplete conditions before invoking a tool. This approach is practical as it helps mitigate potential safety or security risks associated with incorrect tool invocation. However, in more relaxed environments where some degree of incorrect tool usage is acceptable, allowing the model to observe the outcomes of tool invocation and reassess the completeness of the conditions could potentially enhance performance. Recent studies in tool-agent frameworks also suggest that observing the outcomes of previous actions and adjusting plans accordingly can improve task execution (Yao et al., 2023; Qin et al., 2023). Building on these insights, we explore whether providing LLMs with the results of tool invocation can enhance their ability to recognize incomplete conditions.

To this end, we first randomly sample 100 manipulated instances in utterance removal, along with their original counterparts, from our main experiment on the APIBank dataset. For each instance,

	7B$_{\text{BASE}}$	7B	13B$_{\text{BASE}}$	13B	70B$_{\text{BASE}}$	70B
API Replacement						
Acc.	50.06	49.94	54.86	53.75	**60.52**	59.66
F1	0	0.94	**64.05**	14.55	56.68	36.19
Utterance Removal						
Acc.	50.00	50.49	53.01	51.29	**54.91**	52.15
F1	0	3.22	**62.25**	5.98	45.87	16.77

Table 6: Performance comparison between Llama-2 model family (Touvron et al., 2023) of different sizes and training approaches. Models not trained on instruction-tuning datasets are marked as BASE.

	API Replacement		**Utterance Removal**	
	Num.	Acc.	Num.	Acc.
Mistral-7B	149	87.25	19	47.37
GPT-3.5-T	182	85.16	52	48.08
GPT-4	360	97.50	273	99.63

Table 7: The results of explanation for the incomplete scenario. *Num.* represents the count of accurately identified incomplete instances, which corresponds to the number of instances evaluated in the explanation assessment. *Acc.* denotes accuracy of explanations judged by GPT-4, respectively.

	Mistral-7B	**Claude-3**	**GPT-3.5-T**	**GPT-4**
API Replacement				
Yes Ratio (%)	87.74/88.79	70.69/91.91	81.37/92.74	75.74/57.25
Utterance Removal				
Yes Ratio (%)	97.16/98.22	84.41/99.51	97.65/99.13	91.09/73.33

Table 8: Predictive distribution on main experiments of ToolBench. The indicators are the same as Table 10.

we consider two types of API call results: 1) Errors in API Calls: Errors caused by incomplete API calls, and 2) Successful API Calls: Cases where a complete API call is made and executed correctly. Based on these, we categorize the input given to the LLMs into three scenarios: 1) *Utterance Removal + Wrong API Call*: Instances where an API call fails due to missing information, 2) *Utterance Removal + Hallucinated API Call*: Instances where APIs are invoked with information not provided in the dialogue context, yet the API calls are successful, and 3) *Complete Condition + Wrong API Call*: Scenarios where the API can be correctly invoked, but the call fails, resulting in an error. In each scenario, the LLMs are provided with the API list, conversation history, and API call results, and they are asked to judge whether the information given was sufficient to address the user requests. In the first and second scenarios, the LLMs should recognize that an API invocation is not possible. In the third scenario, they should determine that the API calls can be successfully made.

When API call results are erroneous, GPT-3.5-Turbo often concludes that the API cannot be invoked, regardless of whether the necessary conditions for a successful API call are met, as demonstrated in Table 5. In contrast, GPT-4 and GPT-4o show a stronger ability to assess the feasibility of API calls accurately, even in complex scenarios. Notably, when the necessary information to invoke APIs is missing, but the API call is still successful, GPT-3.5-Turbo and GPT-4o-mini exhibit limited awareness of hallucinations. On the other hand, GPT-4 and GPT-4o is more adept at recognizing hallucinated API calls and correctly identifying incomplete conditions, even when the API call succeeds. These results may suggest that the reasoning ability of LLMs plays a critical role in interpreting API call results and making accurate predictions.

B.3 Impacts of Model Size and Instruction Tuning

We analyze the impact of model size and instruction-tuning on model performance across our dataset. Specifically, we compare Llama-2-7B, Llama-2-13B, and Llama-2-70B models against their instruction-tuned counterparts, Llama-2-7B-chat, Llama-2-13B-chat, and Llama-2-70B-chat. The evaluation results, presented in Table 6, reveal that both the 7B and 7B-chat models frequently predict "Yes" for most instances, resulting in very low F1 scores. Overall, models that were not trained on instruction-following data generally outperform their instruction-tuned counterparts, regardless of model size. Interestingly, while instruction-tuned models show performance improvements as the model size increases, the non-instruction-tuned models exhibit varying trends, with a decrease in F1 score as the model size scales from 13B to 70B. These findings suggest that training on instruction-following datasets may affect the models' ability to recognize incomplete conditions, indicating a need for further exploration in future work.

B.4 Can Tool-augmented LLMs Correctly Explain Incomplete Conditions?

We probe whether LLMs can accurately explain their decision-making process when they correctly identify incomplete conditions. To this end, we instruct the models to generate explanations for their decisions and assess whether these explanations correctly identify why tools cannot be used. We adopt the Judge LLM (i.e., GPT-4) (Zheng et al.,

Conversation	**User:** Can you check a reminder to call my mom on March 16th? **AI:** Sure, Can you confirm your username and password? **User:** My username is JohnDoe and my password is pass123.
API List	~~{"name":"GetUserToken", "description":"Get the user token", "parameters":["username", "password"]}~~ {"name":"RegisterUser", "description":"Register an account", "parameters":["name", "password", "email"]} {"name":"QueryReminder", "description":"Query a reminder", "parameters":["token", "content", "time"]}
Explanation	The available APIs do not include an API for setting reminders. The only relevant API is 'QueryReminder' which is used for querying reminders, but there is no API for setting reminders.
Conversation	**User:** Can you show me the historical events ~~on December 10th~~?
API List	{"name":"QueryHistoryToday", "description":"Query the history of the given date", "parameters":["date"]} {"name":"Calculator", "description":"Provide basic arithmetic operations", "parameters":["formula"]}
Explanation	The available APIs don't include an API for showing historical events. The only API provided is for the calculator to perform arithmetic operations.

Table 9: **Examples of Incorrect Explanation by GPT-3.5-Turbo**. The upper example illustrates a case of erroneous explanation in API Replacement, while the lower example shows one in Utterance Removal. Note that the model correctly identifies both instances as incomplete within a binary classification setup. Removed and newly included information, as part of our manipulation strategy, are highlighted with ~~strikethrough~~ and wavy underline, respectively. Wrong explanations from the models are manually highlighted by the authors. Additional examples can be found in Appendix C.

	Llama-2-13B	Llama-2-70B	Vicuna-13B	Mistral-7B	Claude-3	GPT-3.5-T	GPT-4
API Replacement							
Yes Ratio (%)	24.35/0.0	57.8/62.0	78.13/54.85	68.56/67.27	64.62/84.16	64.18/65.84	51.65/53.78
Utterance Removal							
Yes Ratio (%)	24.34/0.0	65.62/62.71	88.82/65.13	87.04/89.28	69.57/96.88	78.29/72.20	50.49/49.34

Table 10: Predictive distribution on main experiments of APIBank. We measure *Yes Ratio*, which represents the proportion where the model predicts that it can invoke the APIs. We report the distribution for both zero-shot and four-shot in a format of (0-shot/4-shot).

2024) to evaluate the correctness of the explanations. We manually craft four-shot examples with a balanced class distribution to ensure a reliable evaluation.

In Table 7, we observe that GPT-3.5-Turbo achieves an accuracy of 85.16% for API Replacement, while Mistral-7B shows a performance with an accuracy of 87.25%. For Utterance Removal, the accuracy of GPT-3.5-Turbo and Mistral-7B are 48.08% and 47.37%, respectively, showing similar performance. These results indicate that it is more challenging for LLMs to provide accurate explanations when users offer insufficient context (Utterance Removal) compared to when the necessary tools are unavailable (API Replacement).

To further verify the reliability of Judge LLM in assessing explanation validity, we manually annotated 100 randomly sampled instances from GPT-3.5-Turbo's predictions, evenly divided between API Replacement and Utterance Removal. The agreement rate between human evaluators and Judge LLM is 82%. This high level of agreement suggests that Judge LLM's evaluations are closely aligned with those of human evaluators, establishing it as a credible and effective assessment tool.

Additionally, we examine instances where the LLMs generated incorrect reasoning, as shown in Table 9. In API Replacement, LLMs often misunderstand the user's intent, leading to inaccurate assertions that the available APIs are insufficient. Conversely, in Utterance Removal, the predominant errors stem from incorrect explanations asserting that no appropriate APIs are available, even when they are present.

B.5 Prediction Analysis of "Yes" Token under Incomplete Conditions

We analyze the models' predictions to understand how they respond to incomplete conditions, particularly focusing on their decisions to invoke tools. Specifically, we examine the likelihood with which LLMs indicate that they can call APIs based on their predictions during our main experiments. As shown in Table 8 and Table 10, most models tend to overestimate their ability to invoke tools, a tendency that becomes more pronounced when the

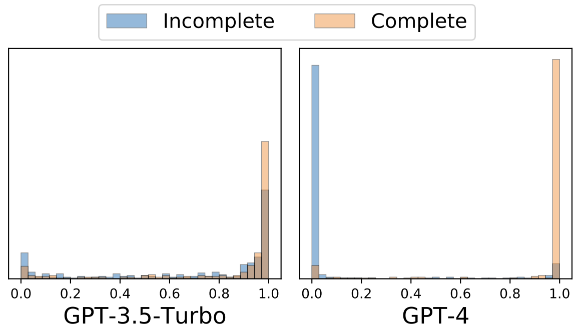

Figure 3: The probability distribution of "Yes" token for complete and incomplete instances. The utterance removal instances from APIBank are used for analysis.

models perform poorly.

To further illustrate these tendencies, we visualize the probability distributions of two LLMs when faced with complete and incomplete instances. The visualizations reflect the probability assigned to the token representing a decision to invoke a tool, as seen in Fig. 3. While GPT-4 successfully differentiates between complete and incomplete conditions, GPT-3.5-Turbo struggles to distinguish corrupted instances, highlighting a key difference in their decision-making processes.

C Samples of Incomplete Instances and Explanation

We present the manipulated instances with different strategies are presented from Fig. 4 to Fig. 11. Both accurately and inaccurately modified instances resulting from our dataset construction method are provided. We also present additional explanation examples generated by LLMs when they recognize incomplete conditions on manipulated samples. Examples of explanations in API Replacement are illustrated in Fig. 12 and Fig. 13. Examples of explanations in Utterance Removal are depicted in Fig. 14 and Fig. 15.

D Prompt Templates

The text prompt used in the dataset construction is presented in Fig. 16. The text prompt used in the zero-shot experiments (Table 2) is presented in Fig. 17. The text prompt used in CoT experiments is in Fig. 18 The text prompts used in the experiments of LLM explanation and LLM judgment are presented in Fig. 19 and Fig. 20 respectively. The text prompts used in Section B.2 are in Fig. 21 and Fig. 22. The text prompt used in Section B.1 is in Fig. 23. When implementing few-shot prompting, we follow the approach of setting up interactions between a human and an AI assistant to provide examples (OpenAI, 2023b).

			Phi-3 small	Llama3.1 8B	Llama3.1 70B	Qwen2 7B	GPT-4o mini	GPT-4o
API Replacement								
APIBank	Acc.	0-shot	61.60	76.06	79.10	68.77	88.82	86.03
		CoT	73.75	78.13	78.61	70.84	82.38	82.99
	F1	0-shot	36.80	71.32	81.22	58.62	88.41	86.23
		CoT	72.80	78.92	81.47	70.37	83.50	84.38
ToolBench	Acc.	0-shot	54.84	69.45	72.64	67.36	77.69	76.48
		CoT	61.32	75.16	64.95	66.92	64.51	70.33
	F1	0-shot	14.55	56.43	76.97	52.48	72.97	75.06
		CoT	50.97	77.40	72.80	68.61	70.87	75.27
Utterance Removal								
APIBank	Acc.	0-shot	51.95	52.63	77.42	57.39	77.25	88.29
		CoT	63.33	61.80	81.66	57.39	86.59	85.74
	F1	0-shot	4.07	19.60	80.06	29.69	73.41	88.74
		CoT	54.81	55.97	83.98	46.93	86.72	86.96
ToolBench	Acc.	0-shot	49.94	50.19	71.54	51.10	57.57	74.51
		CoT	51.88	52.26	68.31	53.43	58.99	72.19
	F1	0-shot	0.51	5.41	74.94	10.85	35.18	70.90
		CoT	29.01	45.17	74.56	46.27	64.82	75.87

Table 11: Performance evaluation results of LLM by manipulation type. The accuracy (Acc.) and F1 score (F1) are used for evaluation metrics. Both the zero-shot and Chain-of-Though (CoT) performance are presented.

[Conversation]

User: Can you tell me about the stock price of SQ on March 15th, 2022?

[Relevant API (Removed)]

```
{\"name\": \"QueryStock\", \"description\": \"This API queries the
stock price of a given stock code and date.\", \"input_parameters\":
{\"stock_code\": {\"type\": \"str\", \"description\": \"The stock
code of the given stock.\"}, \"date\": {\"type\": \"str\",
\"description\": \"The date of the stock price. Format: %Y-%m-%d\"}},
\"output_parameters\": {\"stock_price\": {\"type\": \"float\",
\"description\": \"The stock price of the given stock.\"}}}
```

[Replaced Irrelevant API]

```
{'name': 'QueryBalance', 'description': 'This API queries the
balance of a given user.', 'input_parameters': {'token': {'type':
'str', 'description': 'The token of the user.'}},
'output_parameters': {'balance': {'type': 'float', 'description':
'The balance of the user.'}}}
```

Figure 4: *API Replacement* Successful case from APIBank instances.

[Conversation]

User: I'm interested in reading the content of a specific URL. Can you fetch the content of the URL 'https://www.example.com' using the 'Get Content' API? Also, I would like to know the trending news articles using the 'Get Trending' API.

[Relevant API (Removed)]

```
{\"category_name\": \"Tools\", \"tool_name\": \"URL Content Extractor\", \"api_name\":
\"Search Content\",  \"api_description\": \"Allows you to search content for content
based on a query.\", \"required_parameters\": [{\"name\": \"query\", \"type\": \"STRING\",
\"description\": \"\", \"default\": \"\"}], \"optional_parameters\": [], \"method\":
\"GET\"}\n{\"category_name\": \"Tools\", \"tool_name\": \"URL Content Extractor\",
\"api_name\": \"Get Trending\", \"api_description\": \"This will return a list of news
articles that are trending.\", \"required_parameters\": [], \"optional_parameters\": [],
\"method\": \"GET\"}
{\"category_name\": \"Tools\", \"tool_name\": \"URL Content Extractor\", \"api_name\":
\"Get Content\", \"api_description\": \"Give a url and it will spit back the content
of that url\", \"required_parameters\": [{\"name\": \"url\", \"type\": \"STRING\",
\"description\": \"\", \"default\": \"https://www.cnbc.com/2022/06/27/three-arrows-capital
-crypto-hedge-fund-defaults-on-voyager-loan.html\"}], \"optional_parameters\": [],
\"method\": \"GET\"}
```

[Replaced Irrelevant API]

```
{'category_name': 'News_Media', 'tool_name’: 'Article/Page Content Extractor for SEO’,
'api_name’: 'Content Extraction’, 'method': 'POST’, 'required_parameters': [{'name': 'url’,
'type': 'STRING’, 'description': '', 'default’: 'https://www.freelancer.com/articles/
starting-your business/questions-to-ask-about-your-business-idea’}], 'optional_parameters’:
 [], 'api_description': 'Allows to extract the content from a given URL’}
{'category_name’: 'News_Media', 'tool_name': 'dev-to-api', 'api_name': \"Get the month’s
top articles\", 'method': 'GET’, 'required_parameters’: [], 'optional_parameters': [],
'api_description': \"It returns the month's top articles from dev.to website.\"}
{'category_name': 'Tools', 'tool_name': 'URL Content Extractor', 'api_name’:
'Search Content', 'method': 'GET', 'required_parameters': [{'name': 'query', 'type’:
'STRING', 'description': '', 'default': ''}], 'optional_parameters': [], 'api_description’:
'Allows you to search content for content based on a query.'
```

Figure 5: *API Replacement* Successful case from ToolBench instances.

[Conversation]

User: Can you search the definition of the word \"plethora\" for me?
AI: Sure, searching for \"plethora\"...
API-Request: [Dictionary(keyword='plethora')]->[{'partOfSpeech': 'noun', 'definitions': [{'definition': '(usually followed by of) An excessive amount or number; an abundance.', 'synonyms': [], 'antonyms': [], 'example': 'The menu offers a plethora of cuisines from around the world.'}, {'definition': 'An excess of red blood cells or bodily humours.', 'synonyms': [], 'antonyms': []}], 'synonyms': ['glut', 'myriad', 'slew', 'superfluity', 'surfeit'], 'antonyms': []}]
AI: Here's what I found: \"a large or excessive amount of something.\”
Can I help you with anything else?\
User: Yes, can you give me the synonyms for \"plethora\"?

[Relevant API (Removed)]

```
{\"name\": \"Dictionary\", \"description\": \"This API searches the
dictionary for a given keyword.\", \"input_parameters\":
{\"keyword\": {\"type\": \"str\", \"description\": \"The keyword to
search.\"}}, \"output_parameters\": {\"results\": {\"type\":
\"list\", \"description\": \"The list of results. Format be like
[{\\\"partOfSpeech\\\": \\\"xxx\\\", \\\"definitions\\\":
[{\\\"definition\\\": \\\"xxx\\\", \\\"example\\\": \\\"xxx\\\",
\\\"synonyms\\\": [\\\"xxx\\\", \\\"xxx\\\"]}, ...]\"}}}
```

[Replaced Irrelevant API]

```
{'name': 'SearchEngine', 'description': 'This API searches for a
given keyword for search engine.', 'input_parameters': {'keyword':
{'type': 'str', 'description': 'The keyword to search.'}},
'output_parameters': {'results': {'type': 'list', 'description':
'The list of results.'}}}
```

Figure 6: *API Replacement* Failed case from APIBank instances.

[Conversation]
User: I'm creating a database and I need unique identifiers for each entry. Can you help me generate UUIDs?

[Relevant API (Removed)]


```
{\"category_name\": \"Tools\", \"tool_name\": \"Helper Function\",
\"api_name\": \"Random\", \"api_description\": \"Generate Random
String with custom length\\nBASIC: 20 \\nPRO: 30\\nULTRA: 70\\nMEGA:
150\", \"required_parameters\": [], \"optional_parameters\":
[{\"name\": \"length\", \"type\": \"NUMBER\", \"description\": \"\",
\"default\": \"20\"}], \"method\": \"GET\", \"template_response\":
{\"result\": \"str\"}}
{\"category_name\": \"Tools\", \"tool_name\": \"Helper Function\",
\"api_name\": \"Generate UUID\", \"api_description\": \"Generate
UUID v4 with dash or not.\\nmax\\nBASIC: 3\\nPRO: 20\\nULTRA:
50\\nMEGA: 150\", \"required_parameters\": [],
\"optional_parameters\": [{\"name\": \"timestamp_first\", \"type\":
\"BOOLEAN\", \"description\": \"\", \"default\": \"false\"},
{\"name\": \"remove_dash\", \"type\": \"BOOLEAN\", \"description\":
\"\", \"default\": \"true\"}, {\"name\": \"count\", \"type\":
\"STRING\", \"description\": \"\", \"default\": \"5\"}], \"method\":
\"GET\", \"template_response\": {\"result\": [\"list of str with
length 3\"]}}
```


[Replaced Irrelevant API]


```
{'category_name': 'Tools', 'tool_name': 'Random Strings',
'api_name': 'UUID', 'method': 'GET', 'required_parameters': [],
'optional_parameters': [], 'api_description': 'Generates a random
version 4 UUID.’}
{'category_name': 'Tools', 'tool_name': 'QR Code Generator',
'api_name': 'Generate Basic - Direct Image', 'method': 'GET',
'required_parameters': [{'name': 'data', 'type': 'STRING',
'description': '', 'default': '1234'}], 'optional_parameters':
{'name': 'size', 'type': 'NUMBER', 'description': '', 'default':
'500'}], 'api_description': \"Generates a QR code as a direct image
with limited settings. (NOTE: doesn't show correctly in RapidAPI)\"}
```


Figure 7: *API Replacement* Failed case from ToolBench instances.

[Conversation]
User: Hi, can you help me check my alarm for March 20th, 2023 at 6:30AM?
AI: Sure, to access your alarm information, I'll need to authenticate. Can you please provide me with your email, username, and password?
User: My email is janesmith@example.com, username is JaneSmith, and password is password.

[Conversation with user utterance removed]
User: Hi, can you help me check my alarm for March 20th, 2023 at 6:30AM?
AI: Sure, to access your alarm information, I'll need to authenticate. Can you please provide me with your email, username, and password?
User: My email is janesmith@example.com, username is JaneSmith.

[Relevant API]


```
{\"name\": \"GetUserToken\", \"description\": \"Get the user token
by username and password.\", \"input_parameters\": {\"username\":
{\"type\": \"str\", \"description\": \"The username of the user.\"},
\"password\": {\"type\": \"str\", \"description\": \"The password
of the user.\"}}, \"output_parameters\": {\"token\": {\"type\":
\"str\", \"description\": \"The token of the user.\"}}}
{\"name\": \"QueryAlarm\", \"description\": \"The API for querying
alarm clock, help user to check the alarm clock they have set.\",
\"input_parameters\": {\"token\": {\"type\": \"str\",
\"description\": \"User's token.\"}, \"time\": {\"type\": \"str\",
\"description\": \"The time for alarm. Format: %Y-%m-
%d %H:%M:%S\"}}, \"output_parameters\": {\"info\": {\"type\":
\"json\", \"description\": \"alarm info including username, time,
content and location\"}}}
```


Figure 8: *Utterance Removal* Successful case from APIBank instances.

[Conversation]
User: Can you help me analyze the number of pages indexed in Google for my company's website? The URL is 'www.example.com'.

[Conversation with user utterance removed]
User: Can you help me analyze the number of pages indexed in Google for my company's website?

[Relevant API]
{\"category_name\": \"Tools\", \"tool_name\": \"SEO - Count website pages in Google index\", \"api_name\": \"countPages\", \"api_description\": \"Count website pages in Google index\", \"required_parameters\": [{\"name\": \"url\", \"type\": \"STRING\", \"description\": \"\", \"default\": \"\"}], \"optional_parameters\": [], \"method\": \"GET\"}

Figure 9: *Utterance Removal* Successful case from ToolBench instances.

[Conversation]
User: Can you search for news articles about sustainable development goals?
AI: Sure, what specific keyword are you looking for?
User: the sustainable development.

[Conversation with user utterance removed]
User: Can you search for news articles about sustainable development goals?
AI: Sure, what specific keyword are you looking for?
User: Can you help me find information?

[Relevant API]
{\"name\": \"SearchEngine\", \"description\": \"This API searches for a given keyword for search engine.\", \"input_parameters\": {\"keyword\": {\"type\": \"str\", \"description\": \"The keyword to search.\"}}, \"output_parameters\": {\"results\": {\"type\": \"list\", \"description\": \"The list of results.\"}}}

Figure 10: *Utterance Removal* Failed case from APIBank instances.

[Conversation]
User: I am working on a project that requires unique IDs for each document. Can you assist me in generating GUIDs for the documents? I would need 50 GUIDs in total.

[Conversation with user utterance removed]
User: I am working on a project that requires unique IDs for each document. Can you assist me in generating GUIDs for the documents?

[Relevant API]
{\"category_name\": \"Tools\", \"tool_name\": \"GUID generator\", \"api_name\": \"GenerateGuid\", \"api_description\": \" \", \"required_parameters\": [], \"optional_parameters\": [], \"method\": \"GET\"} {\"category_name\": \"Tools\", \"tool_name\": \"GUID generator\", \"api_name\": \"BulkGenerateGuids\", \"api_description\": \" \", \"required_parameters\": [], \"optional_parameters\": [{\"name\": \"batch_size\", \"type\": \"NUMBER\", \"description\": \"The number of GUIDs to return. Must be between 1 and 10000. If the parameter is not provided, the default batch size is 20.\", \"default\": \"\"}], \"method\": \"GET\"}

Figure 11: *Utterance Removal* Failed case from ToolBench instances.

[Conversation]

User: Can you tell me today's date?

[API List]

```
{'name': 'AddAgenda', 'description': 'The API for adding a schedule
item includes parameters for token, content, time, and location.',
'input_parameters': {'token': {'type': 'str', 'description': "User's
token."}, 'content': {'type': 'str', 'description': 'The content of
the agenda.'}, 'time': {'type': 'str', 'description': 'The time for
agenda. Format: %Y-%m-%d %H:%M:%S'}, 'location': {'type': 'str',
'description': 'The location of the agenda.'}}, 'output_parameters':
{'status': {'type': 'str', 'description': 'success or failed'}}}
```

[LLM Explanation] *(ChatGPT)*

The available APIs do not include an API specifically for retrieving today's date. The AddAgenda API is for adding a schedule item and does not provide information on the current date.

Figure 12: Explanation Success Case in *API Replacement*. The model clearly states and explains that an API to know current date is not provided.

[Conversation]

User: Can you record my blood pressure and heart rate data?

AI: Sure, I can help you with that. Please provide me with your user ID, time, blood pressure, and heart rate data.

User: My user ID is 1234, time is 2022-01-01 10:00:00, blood pressure is 110/70, and heart rate is 90.

[API List]

```
{'name': 'QueryHealthData', 'description': 'This API queries the
recorded health data in database of a given user and time span.',
'input_parameters': {'user_id': {'type': 'str', 'description': 'The
user id of the given user. Cases are ignored.'}, 'start_time':
{'type': 'str', 'description': 'The start time of the time span.
Format: %Y-%m-%d %H:%M:%S'}, 'end_time': {'type': 'str',
'description': 'The end time of the time span. Format: %Y-%m-
%d %H:%M:%S'}}, 'output_parameters': {'health_data': {'type':
'list', 'description': 'The health data of the given user and time
span.'}}}
```

[LLM Explanation] *(ChatGPT)*

The available API "QueryHealthData" allows querying recorded health data based on user ID and a time span. However, the conversation between the user and AI only provided the user ID, time, blood pressure, and heart rate data for a specific timestamp (2022-01-01 10:00:00). The API requires a start and end time for the time span, which are not provided in the conversation. Therefore, the current information is not sufficient to generate API calls to record the blood pressure and heart rate data.

Figure 13: Example of failed explanation in *API Replacement*. The user wants to record health data, but only the *QueryHealthData* API, which is for searching health data, is provided instead of an API for recording health data. The model incorrectly assumes that it must use the *QueryHealthData* API and provides an erroneous explanation that it cannot call the API due to insufficient information.

[Conversation]

User: Can you help me find possible diseases related to a symptom I'm experiencing?
AI: Sure, I can search for possible diseases related to fatigue. Please provide the symptom you want to search for.
User: I prefer not to specify right now.

[API List]

{'name': 'SymptomSearch', 'description': 'This API searches for a given symptom.', 'input_parameters': {'symptom': {'type': 'str', 'description': 'The symptom to search.'}}, 'output_parameters': {'results': {'type': 'list', 'description': 'The list of results. Format be like [{"name":possible disease name, "description": disease details},...]'}}}
{'name': 'AppointmentRegistration', 'description': 'This API registers an appointment of hospital.', 'input_parameters': {'patient_name': {'type': 'str', 'description': 'The name of patient.'}, 'date': {'type': 'str', 'description': 'The date of appointment. Format be like %Y-%m-%d'}, 'doctor_name': {'type': 'str', 'description': 'The name of appointed doctor.'}}, 'output_parameters': {'appointment_id': {'type': 'str', 'description': 'The ID of appointment.'}}}

[LLM Explanation] *(ChatGPT)*

The available API "SymptomSearch" requires the user to provide the specific symptom to search for. Since the user preferred not to specify the symptom at the moment, we do not have enough information to make an API call to search for possible diseases related to the symptom.

Figure 14: Example of successful explanation in *Utterance Removal*. The model recognizes the need to call the *SymptomSearch* API to address the user's requirements. Additionally, it clearly acknowledges and explains that the necessary symptom information required to call the API has not been provided.

[Conversation]
User: Can you please check my account balance?
AI: Sure, I can help you with that. I just need your token. Do you have your token with you?
User: No, I don't have it.
AI: That's alright. I can get your token for you. Can you please confirm your username, password, and email address?
User: My username is user2 and email is user2@example.com.

[API List]

{'name': 'OpenBankAccount', 'description': 'This is an API for opening a bank account for a user, given the account, password and name.', 'input_parameters': {'account': {'type': 'str', 'description': 'The account for the user.'}, 'password': {'type': 'str', 'description': 'The password.'}, 'name': {'type': 'str', 'description': 'account holder name.'}}, 'output_parameters': {'status': {'type': 'str', 'description': 'success or failed'}}}
{'name': 'QueryStock', 'description': 'This API queries the stock price of a given stock code and date.', 'input_parameters': {'stock_code': {'type': 'str', 'description': 'The stock code of the given stock.'}, 'date': {'type': 'str', 'description': 'The date of the stock price. Format: %Y-%m-%d'}}, 'output_parameters': {'stock_price': {'type': 'float', 'description': 'The stock price of the given stock.'}}}
{'name': 'QueryBalance', 'description': 'This API queries the balance of a given user.', 'input_parameters': {'token': {'type': 'str', 'description': 'The token of the user.'}}, 'output_parameters': {'balance': {'type': 'float', 'description': 'The balance of the user.'}}}
{'name': 'GetUserToken', 'description': 'Get the user token by username and password.', 'input_parameters': {'username': {'type': 'str', 'description': 'The username of the user.'}, 'password': {'type': 'str', 'description': 'The password of the user.'}}, 'output_parameters': {'token': {'type': 'str', 'description': 'The token of the user.'}}}

[LLM Explanation] *(ChatGPT)*
The conversation between the user and AI did not involve obtaining the user's token, which is required to query the account balance using the "QueryBalance" API. The user provided their username and email address, but without the token, we cannot make an API call to check the account balance.

Figure 15: Example of failed explanation in *Utterance Removal*. The user's requirement is to check the account balance, which necessitates a token. However, the user has not provided the password required to obtain the token. Although an API exists to obtain the token, the model fails to recognize this and incorrectly states that the API call cannot be made due to the absence of the token.

[System Prompt]
You are an expert in data annotation with a specific task of processing user requests, corresponding API descriptions, and API calls.
Your goal is to remove specific information from the user requests to sound natural while ensuring the API call fails due to missing information.
Follow these steps to achieve this:

1. Analyze User Requests and API Information:
- Examine the relationship between the user's request, the API description, and the actual API call.
- Identify the parameters and details the API requires to function correctly.

2. Generate the Thought Process:
- Before making any modifications, articulate your thought process.
- Select information from the user utterances that can be removed while keeping the dialogue context intact.
- Ensure you only remove information relevant to the API descriptions and calls provided.

3. Modify the User Request:
- Adjust the user request so that it remains natural-sounding but lacks the necessary information for the API call.
- Do not alter or remove any parts of the dialogue other than the essential information identified.

Output the results in the following JSON format:

```
{
"thought": "YOUR_THOUGHT_PROCESS",
"answer": "MODIFIED_CONVERSATION"
}
```

[User Prompt]
API_Description:
{api_description}

Conversation:
{conversation}

Figure 16: Data Construction Prompt Template.

[System Prompt]
I will provide you with a user's query, and you should assess, based on your own capabilities, whether the current information is sufficient to generate API calls for addressing the user's query. The information includes (a) conversation between user and AI (you) and (b) the list of available APIs.
You can't use APIs other than the given APIs.
Based on the given query, if you think you can successfully address the user's request by making an API call, please respond with "Yes"; otherwise, respond with "No".

[User Prompt]
Here is the previous conversation between User and AI (you):
[Conversation Begins]
{conversation}
[Conversation Ends]

Here is the list of available APIs:
[List of APIs Begins]
{api_list}
[List of APIs Ends]

Can you address the user's request by making an API call now?
Please answer in "Yes" or "No".

Figure 17: Zero-shot Experiments Prompt Template.

[System Prompt]
I will provide you with a user's query, and you should assess, based on your own capabilities, whether the current information is sufficient to generate API calls for addressing the user's query. The information includes (a) converstaion between user and AI (you) and (b) the list of available APIs. You can't use APIs other than the given APIs.
Based on the given query, if you think you can successfully address the user's request by making an API call, please respond with "Yes"; otherwise, respond with "No". Begin your response with brief reasoning first.

[User Prompt]
Here is the previous conversation between User and AI (you):
[Conversation Begins]
{conversation}
[Converstaion Ends]

Here is the list of available APIs.
[List of APIs Begins]
{api_list}
[List of APIs Ends]

Can you address the user's request by making an API call now? Begin your response with brief reasoning first. Strictly follow the format: "Thought: {{your_thought}}
"Answer: {{Yes or No}}".

Figure 18: CoT Experiments Prompt Template.

[System Prompt]
I will provide you with a user's query, and you should assess, based on your own capabilities, whether the current information is sufficient to generate API calls for addressing the user's query. The information includes (a) conversation between user and AI (you) and (b) the list of available APIs.
You can't use APIs other than the given APIs.
Based on the given query, if you think you can successfully address the user's request by making an API call, please respond with "Yes"; otherwise, respond with "No".
Begin your response your decision ("Yes" or "No") and then output explanations behind your decision.
Follow this format: Decision: <Yes/No> Explanation: <your explanation>

[User Prompt]
Here is the previous conversation between User and AI (you):
[Conversation Begins]
{conversation}
[Conversation Ends]

Here is the list of available APIs:
[List of APIs Begins]
{api_list}
[List of APIs Ends]

Figure 19: Prompt template of explanation.

[System Prompt]
You are a judge evaluating the appropriateness of the AI Assistant's response.
The AI Assistant was tasked with determining whether it could create an API call to solve a user's query, based on the provided APIs and conversations.
There isn't enough information available to actually perform the API call.
Your role is to assess if the AI Assistant's reasoning for being unable to create the API call is valid.
If the necessary information is available in the APIs and conversations but the AI Assistant claims it isn't, or if the AI Assistant states that unnecessary information to call API is required, you should judge the response as incorrect.
Output "Correct" if the AI Assistant's reasoning is valid, and "Wrong" if it is not.

[User Prompt]
[Conversation Begins]
{conversation}
[Conversation Ends]

[List of APIs Begins]
{api_list}
[List of APIs Ends]

[AI Assistant Explanation Begins]
{explanation}
[AI Assistant Explanation Ends]

Figure 20: Prompt template of Judge LLM.

[System Prompt]
I will provide you with a user's query, and you should assess, based on your own capabilities, whether the current information is sufficient to generate API calls for addressing the user's query.
The information includes (a) conversation between user and AI (you), (b) the list of available APIs and (c) the previous API call made to address the user's request (if applicable).
You can't use APIs other than the given APIs.
Based on the given query, if you think you can successfully address the user's request by making an API call, please respond with "Yes"; otherwise, respond with "No".

[User Prompt]
Here is the previous conversation between User and AI (you):
[Conversation Begins]
{conversation}
[Conversation Ends]

Here is the list of available APIs:
[List of APIs Begins]
{api_list}
[List of APIs Ends]

This is not your first attempt, and the previous attempt has failed.
In your previous attempt, you determined that you could create an API call to address user's request, and you created an API Call in subsequent steps.
Before determining if you can now address the user's request with an API call, you must review and analyze the API calls you have previously made:
[Previous API Call Begins]
{api_call}
[Previous API Call Ends]

Can you address the user's request by making an API call now?
Please answer in "Yes" or "No".

Figure 21: Prompt Template of self-verification for API invocation error. We use the same prompt for error in utterance removal and complete scenario. Both receive information about an erroneous API call result and are asked whether the API call is currently feasible.

[System Prompt]
I will provide you with a user's query, and you should assess, based on your own capabilities, whether the current information is sufficient and appropriate to answer the user's query.
The information includes (a) conversation between user and AI (you), (b) the list of available APIs, and (c) the API call generated by AI(you) in the previous step to address the user's query.
No API other than the given API is available.
Based on the given information, if you think you can provide an appropriate answer to user's request, please response with "Yes"; otherwise, respond with "No".

[User Prompt]
Here is the previous conversation between User and AI (you):
[Conversation Begins]
{conversation}
[Conversation Ends]

Here is the list of available APIs:
[List of APIs Begins]
{api_list}
[List of APIs Ends]

Here is the API calls you have previously made
[Previous API Call Begins]
{api_call}
[Previous API Call Ends]

Can you provide an appropriate answer to the user now?
Please answer in "Yes" or "No".

Figure 22: Prompt Template of self-verification for hallucination.

[System Prompt]
You are an AI assistant tasked with addressing the user's query based on the provided conversation history (1 to t). Your goal is to respond to the user's query in step t+1.
You may use the given API descriptions to assist in your response.
If you choose to call an API, format the output as [ApiName(key1='value1', key2='value2', ...)], where ApiName is the actual API name and the keys and values are the corresponding parameters.
In this case, ensure that your output begins with a square bracket "[" and ends with a square bracket "]".

[User Prompt]
Here is the previous conversation between User and AI (you):
[Conversation Begins]
{conversation}
[Conversation Ends]

Here is the list of available APIs:
[List of APIs Begins]
{api_list}
[List of APIs Ends]

Figure 23: Prompt Template of implicit evaluation of LLMs with free-form generation.