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 investigates whether LLMs can identify incomplete conditions and appropriately determine when to refrain from using tools. To evaluate this capability, we construct a new benchmark that simulates the ambiguous and incomplete conditions common in real-world interactions. Our experiments reveal that even state-of-the-art LLMs often struggle to identify these conditions, attempting to use tools without sufficient information or when the correct tool is unavailable. Based on the observations, we propose a novel reasoning strategy that explicitly instructs models to assess the sufficiency of information and the availability of tools. Our proposed method significantly enhances the models' ability to recognize incomplete conditions, resulting in more informed and contextually appropriate tool-use decisions. We believe our research contributes to advancing the reliability of LLMs, especially in real-world applications where incomplete conditions are common.

1 Introduction

Recently, there has been significant improvement in integrating large language models (LLMs) with tools (Li et al., 2023; Qin et al., 2024; Patil et al., 2024; Schick et al., 2024; Hao et al., 2024; Liao et al., 2025). These *tool-augmented LLMs* can perceive up-to-date information, acquire real-world interaction capabilities, and perform complex tasks, enhancing user experiences across various applications (Yang et al., 2023; Hong et al., 2024).

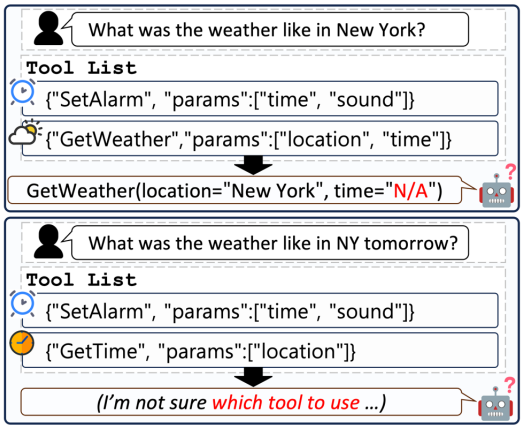

Figure 1: Illustration of incomplete conditions encountered by tool-augmented LLMs when invoking tools.

However, to ensure reliability in real-world settings, LLMs must recognize two fundamental failure points of tool use: the unavailability of a correct tool to execute a user's instruction, and the insufficiency of information to populate the tool's required parameters. Addressing these two failure points is increasingly crucial because the proliferation of tools raises the likelihood that the available tools itself contains incorrect ones, and users are often unaware of the precise information required to satisfy a tool's parameters. While several benchmarks have been proposed to evaluate model recognition of these failure points, they often do so in isolation or under unrealistic assumptions. Specifically, many focus on simplified scenarios, such as the mere absence of any tool (Ning et al., 2024; Yan et al., 2024; Lu et al., 2025), but neglect the more deceptive challenge where a semantically similar but incorrect tool is present (Zhang et al., 2024b). Furthermore, they frequently operate as if user queries are perfectly formed and contain all necessary arguments (Huang et al., 2024), overlooking the common reality of incomplete requests. While recent works such as BFCL V3 (Patil et al., 2025) and When2Call (Ross et al., 2025) explore more realistic scenarios, they either narrowly focus

Benchmarks	Missing Tools	Missing Params	# Domains	# Tools	# Instances
ToolBeHonest (Zhang et al., 2024b)	✓*	✗	-	893	700
BFCL v3 (Patil et al., 2025)	✓*	✓	8	76	400
ToolSandbox (Lu et al., 2025)	✓*	✓	11	34	224
When2Call (Ross et al., 2025)	✓†	✓	5	504	2,357
Ours	✓	✓	57	2,799	1,610

Table 1: Comparison of benchmarks for incomplete tool usage across key criteria, including scenario coverage, diversity, and scale. * denotes the simple removal of the necessary tool, while † denotes queries designed to be unanswerable with the available toolset. Unlike prior benchmarks, our dataset offers broader coverage and creates more realistic challenges by replacing necessary tools with semantically similar alternatives.

on specific tool domains or rely on a limited number of tools, thereby limiting their assessment of generalization across diverse scenarios.

To address these limitations, we introduce a new benchmark designed to probe these challenges in more realistic settings, as illustrated in Fig. 1. We construct a benchmark by systematically modifying instances from existing tool-augmented LLM datasets (Li et al., 2023; Qin et al., 2024). To create scenarios involving tool unavailability, we replace the correct tool with a semantically similar but incorrect one. This setup mimics the real-world tool retrieval process, where LLMs must select from plausible yet unsuitable candidates. For scenarios with insufficient information, we modify user requests to exclude the parameters required for a tool call. Finally, we meticulously verify the entire instances to ensure its validity and naturalness.

Our experiments reveal that even state-of-the-art LLMs frequently struggle to recognize incomplete conditions and often allow tool invocations even when necessary information or tools are unavailable. Motivated by models' frequent failure to accurately identify the necessary information for tool invocation, we introduce Structured Condition Verification (SCV), which explicitly guides LLMs to systematically assess the adequacy of available information and the presence of required tools. Our findings show that SCV enhances the ability to recognize incomplete conditions, leading to more accurate and contextually appropriate tool use.

Our contributions can be summarized as follows. (1) We construct a dataset simulating incomplete conditions by manipulating tool-use datasets. (2) We evaluate LLMs' ability to recognize impossible tool invocations, highlighting difficulties in identifying missing information. (3) We comprehensively analyze model behavior in incomplete scenarios and demonstrate the effectiveness of SCV.

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; Liao et al., 2025), ranging from basic retrieval systems (Chen et al., 2017) and arithmetic operations (Inaba et al., 2023; Schick et al., 2024) to complex programming languages (Gou et al., 2024b; Zhang et al., 2024a; Ding et al., 2025) and APIs (Xu et al., 2023; Guo et al., 2024; Yuan et al., 2025). Various benchmarks have been developed to evaluate LLMs' tool usage capabilities, including tool selection timing (Huang et al., 2023), robustness to noisy tool descriptions (Ye et al., 2024b), tool error handling (Sun et al., 2024), and safety (Zhang et al., 2024c; Ye et al., 2024a).

Although several 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; Lu et al., 2025), assumed a perfect tool utilization plan is given (Huang et al., 2024), or overlooked situations with similar but incorrect tools (Zhang et al., 2024b). Ross et al. (2025) introduce a valuable benchmark for assessing LLMs in tool-use scenarios but do not incorporate rigorous human verification of their synthetically generated examples. Our work fills these gaps by exploring more diverse and realistic scenarios, validated through meticulous human annotation, including situations where incorrect but similar tools are provided or where users provide incomplete information required for proper tool calling. Although research in question answering has explored handling irrelevant knowledge and ambiguous requests (Min et al., 2020; Kamath et al., 2020; Cole et al., 2023; Jeong et al., 2024), the ability of LLMs to recognize impractical tool usage still remains underexplored.

Dialogue	**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~~.
API List	("GetUserToken", "Get the user token by username and password.", ["username", "password"]) ("QueryAlarm", "The API for querying alarm clock to check the alarm clock the user have set.", ["token", "time"])
Dialogue	**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."
API List	("Search Content", "Allows you to search content for content based on a query.", ["query"]) ~~("Get Trending", "This will return a list of news articles that are trending.", [])~~ ~~("Get Content", "Give a URL and it will return the content of that URL.", ["url"])~~ ("Content Extraction", "Allows to extract the content from a given URL.", ["url"]) ("Get the month's top articles", "It returns the month's top articles from dev.to website.", [])

Figure 2: **Data Examples**. The upper example illustrates a case of utterance removal in APIBank, while the lower example shows an API replacement from the ToolBench dataset. Removed and newly introduced information, as part of our manipulation strategy, are highlighted with a strikethrough and an underline, respectively.

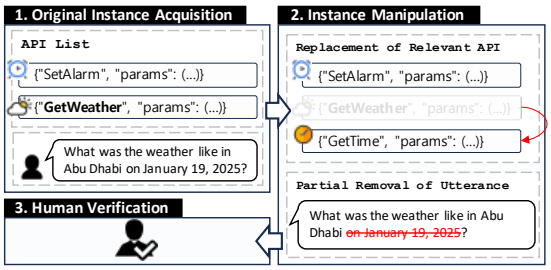

Figure 3: Overall pipeline for dataset construction.

	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.	17.66	52.38	18.87	51.15
Avg. API	434.39	713.35	443.38	747.53

Table 2: **Dataset Statistics**. The *API Replacement* and *Utterance Removal* denote *Relevant API Replacement* and *Partial Removal of User Utterance*, respectively. *Avg. Uttr.* denotes the average word count per utterance and *Avg. API* denotes the average string length.

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., 2024). Both datasets are designed to evaluate how LLMs can respond to user requests using APIs[1]. In their original dataset, each instance has an available API that can address the user's request, with sufficient information provided to invoke APIs. We deliberately manipulate these instances to simulate scenarios with missing tools or incomplete user-supplied information. In this section, we detail the data source, the manipulation strategy, and the verification process. Fig. 3 illustrates our dataset construction pipeline.

3.1 Data Source

We construct a new dataset by leveraging two existing datasets: APIBank (Li et al., 2023) and ToolBench (Qin et al., 2024). APIBank consists of 73 APIs and 314 multi-turn conversations. We select 450 instances from the test split, excluding those that require tool-retrieval. ToolBench is based on 16,000 real-world APIs across 49 categories from RapidAPI Hub.[2] 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

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. To find semantically similar APIs, we use the sentence encoder proposed by Gao et al. (2021).[3] We concatenate the name and description of each API, convert this text into a fixed-size vector, and then

[1] We refer to Tool and API interchangeably.

[2] https://rapidapi.com/hub

[3] princeton-nlp/sup-simcse-roberta-large

			Llama3.1 8B	Llama3.1 70B	Gemma 3-4b	Gemma 3-27b	Qwen2.5 7B	Qwen2.5 14B	Qwen2.5 32B	Qwen2.5 72B	GPT-3.5 Turbo	GPT-4	GPT-4o mini	GPT-4o
Relevant API Replacement														
APIBank	Acc.	0-shot	76.06	79.10	67.85	78.49	73.05	80.97	85.82	87.94	67.92	87.85	**88.82**	86.03
		4-shot	75.70	82.50	69.50	76.12	78.01	80.50	84.75	87.35	71.32	85.66	85.66	83.35
	F1	0-shot	71.32	81.22	57.10	73.31	65.97	77.35	83.83	87.02	62.39	87.56	**88.41**	86.23
		4-shot	74.81	83.33	58.25	69.76	72.81	76.99	82.73	86.54	65.80	84.99	85.99	82.90
ToolBench	Acc.	0-shot	69.45	72.64	55.56	64.36	72.33	69.29	70.55	74.11	64.29	72.31	77.69	76.48
		4-shot	76.37	**80.55**	57.65	72.64	72.12	74.53	72.43	73.48	57.91	79.01	75.82	77.14
	F1	0-shot	56.43	76.97	20.90	45.51	64.80	57.96	58.74	67.80	46.28	61.93	72.97	75.06
		4-shot	70.34	**80.49**	28.62	64.39	64.15	67.98	75.53	75.51	25.34	77.02	69.36	76.89
Partial Removal of User Utterance														
APIBank	Acc.	0-shot	52.63	77.42	53.12	62.66	54.77	79.28	81.09	81.25	56.20	89.47	77.25	88.29
		4-shot	59.25	84.72	51.48	77.96	56.74	80.92	79.77	86.02	65.87	88.79	79.46	**89.81**
	F1	0-shot	19.60	80.06	21.49	42.24	26.67	74.80	77.76	77.29	36.76	89.27	73.41	88.74
		4-shot	46.90	85.98	12.98	72.87	27.55	77.78	75.74	84.79	55.43	88.70	79.10	**90.07**
ToolBench	Acc.	0-shot	50.19	71.54	50.00	52.09	50.86	50.99	54.68	61.45	49.42	57.05	57.57	74.51
		4-shot	51.36	72.83	50.99	52.46	50.12	51.11	50.62	64.41	50.32	66.11	55.11	**75.16**
	F1	0-shot	5.41	**74.94**	1.46	10.16	5.23	15.68	21.37	43.19	2.98	27.19	35.18	70.90
		4-shot	14.93	70.25	7.44	15.35	2.88	13.88	11.87	51.91	2.04	55.74	25.05	73.33

Table 3: 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 underlined.

select one of the most similar APIs by calculating cosine similarity between the relevant API and all available APIs in the dataset. This strategy reflects realistic situations where LLMs encounter APIs that appear semantically similar but are unsuitable due to subtle differences in functionality.

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 crucial information required for tool invocation, supported by reasoning instructions and five manually-crafted few-shot examples.

3.3 Data Verification

To ensure dataset validity, we manually review all instances to remove cases where user requests could still be fulfilled by the provided APIs or where the manipulations resulted in unnatural. Two authors holding bachelor's degrees or higher in Computer Science conduct this verification process, focusing on: (1) whether alternative or non-replaced APIs could still handle the user's request in API Replacement cases, and (2) whether the remaining information is sufficient for API execution in Utterance Removal cases.[4] Through active discussion and refinement of filtering criteria, we ensure that the dataset includes only instances that genuinely represent realistic incomplete scenarios.

3.4 Dataset Statistics

Our final dataset consists of 727 instances from APIBank and 883 from ToolBench, totaling 1,610 instances. Among these, 900 instances are generated by replacing relevant APIs, while 710 are created by removing parts of the user utterances. Table 2 presents the overall dataset statistics, and Fig. 2 provides examples of the dataset.

4 Experiments

4.1 Setup

Task Formulation We investigate whether LLMs can recognize incomplete conditions, where tool invocation should be withheld due to insufficient user input or unavailable tools. We ask the model to determine whether tool use is feasible based on the given dialogue and available APIs. This setup provides a clear evaluation signal and aligns with real-world multi-stage decision-making, where models must first assess the viability of tool usage before proceeding (Huang et al., 2023, 2024). We use both original and manipulated instances from APIBank and ToolBench, prompting the model to answer "Yes" or "No" accordingly, and evaluate performance using accuracy and F1 score, with "No" as the positive class.

[4]Detailed verification criteria are in Appendix A.

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

Table 4: Human analysis results on our evaluation set.

Models In our experiments, we employ the following open-source and proprietary LLMs: Llama3.1-8B/70B-Instruct (Dubey et al., 2024), Gemma3-4b/27b-it (Team et al., 2025) Qwen2.5-7B/14B/32B/72B-Instruct (Yang et al., 2024), GPT-3.5-Turbo (OpenAI, 2023a), GPT-4, GPT-4o-mini, and GPT-4o (Achiam et al., 2023).

Implementation Details We implement all open-source LLMs with the Huggingface Transformers library (Wolf et al., 2020). For proprietary models, we use `gpt-3.5-turbo-0125`, `gpt-4-0613`, `gpt-4o-mini-2024-07-18`, and `gpt-4o-2024-11-20` for 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.

4.2 Results

Our experimental results, presented in Table 3, reveal several insights regarding the capabilities and limitations of tool-augmented LLMs in recognizing incomplete conditions as follows.

Performance depends on API complexity and conversational context. Tool-augmented LLMs exhibit significant performance differences across manipulation types and datasets. GPT-4o-mini and Qwen2.5-72B achieve strong accuracy (88.82% and 87.94%, respectively) in the API Replacement scenario on APIBank. However, their performance notably declines in scenarios requiring compositional reasoning from conversational context, especially on ToolBench, reflecting their sensitivity to API description complexity.

Models frequently fail to abstain from inappropriate tool invocations. LLMs consistently struggle to abstain from invoking tools under incomplete conditions, as indicated by low F1 scores (e.g., as low as 5.41% for Llama3.1-8B on ToolBench Utterance Removal). This tendency to overestimate tool applicability highlights a significant limitation in models' awareness of information sufficiency.

Few-shot demonstrations have limited impact on complex incompleteness recognition. Few-shot learning generally enhances performance on APIBank, yet has minimal impact on ToolBench scenarios. This suggests that simple demonstrations are insufficient for models to robustly identify incomplete conditions (Khot et al., 2022).

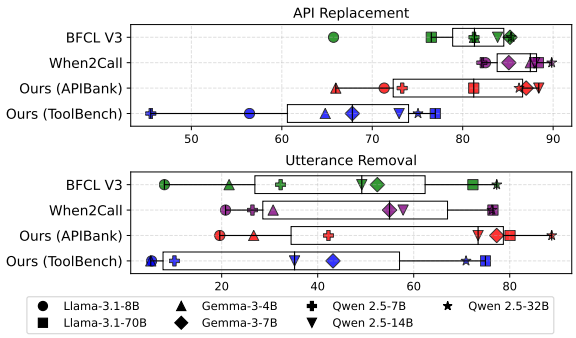

Figure 4: F1-score comparison across benchmarks for simulating incomplete scenarios for tool usage.

These findings underline critical challenges for LLMs in reliably detecting incomplete scenarios, especially those involving complex APIs and nuanced conversational contexts, highlighting the need for structured reasoning approaches.

4.3 Analysis

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.

Results in Table 4 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 performance.

Comparison with Prior Tool Use Datasets Several datasets have been proposed to evaluate LLMs under incomplete tool usage conditions, including BFCL v3 (Patil et al., 2025), ToolSandBox (Lu et al., 2025), and When2Call (Ross et al., 2025). As summarized in Table 1, our dataset provides broader coverage by incorporating and modifying instances from both APIBank and ToolBench, resulting in a larger set of domains (57) and tools

	API Replacement			Utterance Removal		
	Acc.	F1	Corr.	Acc.	F1	Corr.
Llama3.1-8B	58.27	39.66	68.56	60.36	45.1	76.48
Llama3.1-70B	75.13	68.34	81.23	79.61	75.0	84.81
GPT-4	80.50	77.18	85.22	87.83	86.75	88.32

Table 5: Implicit evaluation results 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	Llama 3.1-8B	Llama 3.1-70B	GPT 4	GPT 4o-mini	GPT 4o
✓	✓	89.0	83.0	86.0	97.0	96.0
✓		40.0	82.0	67.0	45.0	79.0
	✓	2.0	92.0	99.0	51.0	91.0

Table 6: The results of providing API invocation outcomes to LLMs across different scenarios. Accuracy is used as the evaluation metric, highlighting how different models handle API errors and recognize hallucinations.

(2,799). Moreover, our dataset is distinguished by its quality assurance; human annotators have manually verified all instances to ensure its validity.

To compare performance across these benchmarks, we evaluate a suite of LLMs in a zero-shot setting. As shown in Fig. 4, the F1 scores for all models are generally lowest on our dataset. This performance drop is particularly sharp on the subset derived from ToolBench, especially in the Utterance Removal. This suggests that the evaluation contexts from our benchmark, particularly the ToolBench subset, are complex for models to resolve. While lower performance does not by itself indicate higher dataset quality, the consistent degradation across all models suggests our benchmark effectively captures a broad and challenging range of real-world failure modes for incomplete conditions.

Implicit Evaluation with Free-form Generation As described in the experiments section, 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 that the model has determined that it can invoke the tool. In contrast, the absence of tool invocation indicates that the model has determined that it cannot invoke the tool. We apply this implicit evaluation framework to Llama3.1-8B/70B and GPT-4 on the APIBank dataset.

The results presented in Table 5 indicate that models generally perform worse in implicit evaluation setups. For example, GPT-4's F1 scores drop from 87.56% and 89.27% to 77.18% and 86.75% for API Replacement and Utterance Removal, respectively. 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 better recognize when tool usage is appropriate.

Impact of Tool Invocation Feedback on Incomplete Condition Recognition Our primary experiments assess whether LLMs can recognize incomplete conditions prior to tool invocation, a practical approach for preventing incorrect or unsafe tool use. However, in more permissive settings, observing the outcomes of tool invocations may help models reassess input completeness. Prior work suggests that utilizing execution feedback can improve reasoning and planning abilities (Yao et al., 2023; Qin et al., 2024; Gou et al., 2024a).

To investigate this, we sample 100 manipulated utterance removal instances and their original counterparts from APIBank. We simulate two types of API outcomes: (1) *Error*, where the API call fails due to missing information; and (2) *Success*, where the call executes correctly. Based on this, we construct three evaluation scenarios: (i) *Utterance Removal + Wrong API Call*, where the call fails due to insufficient user input; (ii) *Utterance Removal + Hallucinated API Call*, where the call succeeds using hallucinated content not grounded in the dialogue; and (iii) *Complete Condition + Wrong API Call*, where sufficient user input is present but the call still fails. In each case, the model is given the API list, dialogue history, and API call result, and is asked to judge whether the provided information is sufficient to fulfill the user request.

Results show that Llama3.1-8B tends to infer incompleteness whenever the call result is an error, regardless of input sufficiency (Table 6). In contrast, GPT-4 and GPT-4o more accurately distinguish be-

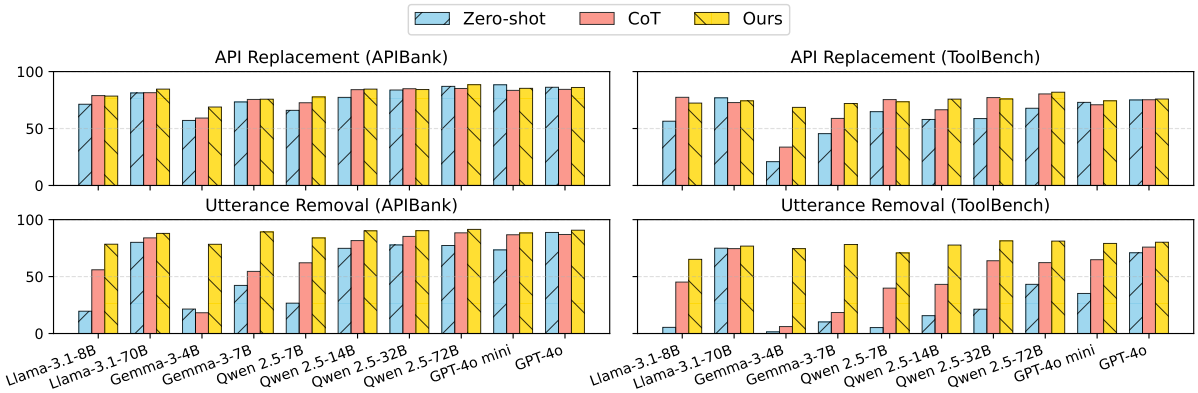

Figure 5: F1-score comparison across different manipulation types and datasets, showing the impact of CoT and SCV (Ours) prompting. Full results are in Table 7.

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".

Follow these steps to structure your reasoning:
1. Problem Understanding
- Summarize the user's ultimate request or goal.

2. Requirement Analysis (Data/Functionality)
- Identify the specific data or functionality needed to fulfill the request.

3. API Mapping
- Check the available APIs to see if they provide the necessary data or functionality identified in Step 2.

4. Information Sufficiency Check
- Determine if the user has given enough information to make a valid API call.
- If critical details are missing (e.g., necessary parameters not provided), you must conclude you cannot currently fulfill the request.

5. Conclusion
- If you can fulfill the request with an available API and sufficient user-provided information, answer "Yes". Otherwise, answer "No".

Figure 6: Overview of the proposed SCV process. SCV guides the model through key verification steps before using the tool. The full prompt is in Fig. 24.

tween truly incomplete inputs and other error types. Furthermore, in cases where hallucinated API calls succeed, Llama3.1-8B and GPT-4o-mini often fail to detect the discrepancy, whereas models including Llama3.1-70B and GPT-4o reliably identify the missing information. These findings suggest that stronger reasoning capabilities enable more accurate interpretation of tool outcomes and improved recognition of incomplete conditions.

5 Structured Condition Verification for Handling Incomplete Conditions

5.1 Proposed Method

Our experiments reveal a critical weakness in how tool-augmented LLMs handle incomplete conditions. LLMs often overlook whether the user input contains sufficient information, leading them to invoke tools based on under-specified or partial contexts. Neither prompting models to respond directly nor providing tool execution results fully resolves this issue, as both approaches often fail to enforce structured assessment of input sufficiency. This tendency to skip preliminary validation results in unreliable tool use and degraded performance, as models lack an inherent reasoning process to evaluate contextual completeness. This deficiency in pre-execution planning and reasoning is a key factor behind failures in incomplete scenarios.

Inspired by prior work highlighting the effectiveness of explicit self-assessment (Manakul et al., 2023; Shinn et al., 2023; Dhuliawala et al., 2024) and structured decomposition for improving reasoning reliability (Press et al., 2023; Ouyang et al., 2024), we adapt these principles to tool-augmented settings, where systematically evaluating input completeness before tool invocation is essential. Specifically, we propose Structured Condition Verification (SCV), a method that enforces a systematic pre-execution check. It compels the model first to decompose the user's request into its core requirements and then explicitly verify whether the available information and tools are sufficient to meet them. SCV guides the model through the following key steps: (1) identifying the user's ultimate goal, (2) enumerating the required information or functionalities, (3) checking available tools for alignment with those requirements, and (4) verifying information sufficiency. Fig. 6 illustrates the system prompt designed for this process.

By explicitly prompting the model to decompose the problem and assess resource availability, SCV improves decision-making in situations involving incomplete or partial information. Furthermore, this strategy operates without requiring tool execution or task-specific fine-tuning, making it a robust

API List	{"name": "VAT validation API endpoint", "parameters":["vat number"]} {"name": "VAT rates & categories API endpoint", "parameters":["country code"]} {"name": "VAT price calculation API endpoint", "parameters":["amount", "country code"]}
Dialog	**User**: As a business owner, I need to validate a VAT number for my company. Please validate the VAT number 'SE556656688001' and provide additional details about the associated company. Also, fetch the latest VAT rates, including the reduced rates for specific categories, for Germany.
CoT	The user's request involves two parts: validating a VAT number and fetching the latest VAT rates, including reduced rates for specific categories. There are two relevant APIs available: one for VAT validation and another for fetching VAT rates. Since both APIs can be called separately to fulfill the user's request, I can successfully address the user's needs with the available APIs. Answer: **Yes**
SCV *(Ours)*	**1.** *(Problem Understanding)* The user's ultimate request is to validate a VAT number for their company and to obtain additional details about the associated company. Additionally, they want to fetch the latest VAT rates, including reduced rates for specific categories. **2.** *(Requirement Analysis)* To fulfill this request, I need to validate the VAT number using the VAT validation API, which requires the VAT number as a parameter. For the VAT rates, I need the country code to fetch the latest VAT rates and reduced rates using the VAT rates & categories API. **3.** *(API Mapping)* The available APIs include a VAT validation API that requires a VAT number and a VAT rates & categories API that requires a country code. Both APIs are relevant to the user's request. **4.** *(Information Sufficiency Check)* However, the user has not provided the VAT number or the country code needed to make the API calls. Without these critical details, I cannot proceed with the API calls. **5.** *(Conclusion)* Therefore, I cannot fulfill the request at this moment due to insufficient information. Answer: **No**

Figure 7: **Case Study**. The example illustrates the corrupted sample generated by our utterance removal strategy in the ToolBench dataset. Removed information, as part of our manipulation strategy, is highlighted with underline. Wrong and Correct reasoning from the models is manually highlighted. GPT-4o-mini is used as an evaluated LLM.

and efficient solution for high-stakes settings or scenarios where models must continuously adapt to newly introduced tools.

5.2 Experiments

Setup We evaluate our method using our manipulated dataset constructed from APIBank and ToolBench. In this setup, we compare Chain-of-Thought (CoT) prompting with our proposed method, SCV. CoT prompts the LLM to *"Begin your response with brief reasoning first,"* encouraging implicit reasoning without a formal structure. The full CoT prompt is shown in Fig. 23.

Results As shown in Fig. 5, both CoT and SCV significantly improve performance over zero-shot prompting across all models. Specifically, Gemma3-4B's F1 score improves from 20.90% to 33.68% with CoT and further increases to 68.57% with SCV on ToolBench in the API Replacement scenario. Similar trends are observed across other models, where CoT boosts performance, particularly for smaller models, and SCV shows further improvement in most cases. Overall, these results suggest that while prompting for reasoning is generally beneficial, the explicit decomposition and verification enforced by SCV provide a more robust path to the correct conclusion.

Case Study Fig. 7 presents a case study illustrating how different reasoning strategies affect the model's behavior when dealing with missing information. This example comes from the ToolBench dataset, where our utterance removal strategy introduces a corrupted version of the original user utterance. The missing details, such as the VAT number and country code, are critical for making the API calls. GPT-4o-mini, evaluated with both CoT and SCV, exhibits different behaviors. With CoT prompting, the model incorrectly assumes it can proceed without the missing information and mistakenly concludes that the request can be fulfilled, as highlighted by the red-colored box. However, with SCV, the model correctly identifies the missing details and concludes that the request cannot be fulfilled, as shown in the green-colored box. This case study demonstrates how SCV handles incomplete queries more effectively by explicitly checking for required information.

6 Conclusion

In this work, we investigate whether tool-augmented LLMs can recognize incomplete conditions before invoking tools. To assess model performance on realistic challenges involving a wide variety of tools, we construct a benchmark by manipulating and human-verifying instances from APIBank and ToolBench. Our findings show that even strong models often fail to abstain from tool use when crucial information or tools are missing. To address this, we propose Structured Condition Verification (SCV), which guides models to explicitly assess input sufficiency and tool availability. This study offers not only a practical solution but also a comprehensive analysis framework for understanding and evaluating the robustness of tool-augmented LLMs under incomplete conditions.

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

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*.

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.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2024. Chain-of-verification reduces hallucination in large language models. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3563–3578.

Hanxing Ding, Shuchang Tao, Liang Pang, Zihao Wei, Jinyang Gao, Bolin Ding, Huawei Shen, and Xueqi Cheng. 2025. ToolCoder: A systematic code-empowered tool learning framework for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17876–17891, Vienna, Austria. 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, and 1 others. 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.

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, Yujiu Yang, Nan Duan, Weizhu Chen, and 1 others. 2024a. Critic: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, and 1 others. 2024b. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

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. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11143–11156.

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, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth*

International Conference on Learning Representations.

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, and 1 others. 2023. Metatool benchmark: Deciding whether to use tools and which to use. In *The Twelfth International Conference on Learning Representations*.

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

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. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7029–7043.

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.

Yusheng Liao, Shuyang Jiang, Yanfeng Wang, and Yu Wang. 2025. ReflecTool: Towards reflection-aware tool-augmented clinical agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13507–13531, Vienna, Austria. Association for Computational Linguistics.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183.

Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

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.

Sirui Ouyang, Zhuosheng Zhang, Bing Yan, Xuan Liu, Yejin Choi, Jiawei Han, and Lianhui Qin. 2024. Structured chemistry reasoning with large language models. In *Proceedings of the 41st International Conference on Machine Learning*, pages 38937–38952.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. 2025. When2Call: When (not) to call tools. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3391–3409, Albuquerque, New Mexico. Association for Computational Linguistics.

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.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

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.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 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. In *First Conference on Language Modeling*.

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, and 3 others. 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-sourced large language models. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

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, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuan-Jing 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*, pages 313–333.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2025. Easytool: Enhancing llm-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 951–972.

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.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2024c. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

A 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 the 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.

B Further Analysis

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

B.1 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 8, 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.2 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., 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 9, we observe that GPT-3.5-Turbo achieves an accuracy of 85.16% and 48.08% for API Replacement and Utterance Removal, respectively. 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 Fig. 8. In API Replacement, LLMs often misunderstand the user's intent, leading to inaccurate assertions that the available APIs are insufficient.

			Llama 3.1 8B	Llama 3.1 70B	Gemma 3-4B	Gemma 3-27B	Qwen 2.5-7B	Qwen 2.5-14B	Qwen 2.5-32B	Qwen 2.5-72B	GPT-4o mini	GPT-4o
API Replacement												
APIBank	Acc.	0-shot	76.06	79.10	67.85	78.49	73.05	80.97	85.82	87.94	88.82	86.03
		CoT	78.13	78.61	70.33	79.79	76.12	85.11	86.05	85.46	82.38	82.99
		SCV	76.24	84.52	71.51	77.90	77.78	84.99	84.52	88.53	84.75	85.58
	F1	0-shot	71.32	81.22	57.10	73.31	65.97	77.35	83.83	87.02	88.41	86.23
		CoT	78.92	81.47	59.19	75.54	72.63	84.13	84.95	85.16	83.50	84.38
		SCV	78.50	84.61	68.90	75.75	77.73	84.61	84.24	88.49	85.29	86.04
ToolBench	Acc.	0-shot	69.45	72.64	55.56	64.36	72.33	69.29	70.55	74.11	77.69	76.48
		CoT	75.16	64.95	59.54	70.13	76.21	74.53	76.73	80.50	64.51	70.33
		SCV	67.71	70.65	65.41	70.75	70.75	72.85	78.93	83.12	69.60	71.07
	F1	0-shot	56.43	76.97	20.90	45.51	64.80	57.96	58.74	67.80	72.97	75.06
		CoT	77.40	72.80	33.68	58.87	75.41	66.48	77.16	80.38	70.87	75.27
		SCV	72.35	74.31	68.57	71.96	73.50	75.82	75.93	81.89	74.34	75.92
Utterance Removal												
APIBank	Acc.	0-shot	52.63	77.42	53.12	62.66	54.77	79.28	81.09	81.25	77.25	88.29
		CoT	61.80	81.66	54.28	67.93	69.90	82.73	86.35	88.49	86.59	85.74
		SCV	76.48	87.34	78.45	88.98	83.06	89.97	89.80	91.12	87.50	89.97
	F1	0-shot	19.60	80.06	21.49	42.24	26.67	74.80	77.76	77.29	73.41	88.74
		CoT	55.97	83.98	18.24	54.55	62.11	81.61	85.26	88.41	86.72	86.96
		SCV	78.43	87.91	78.35	89.21	83.93	90.18	90.31	91.43	88.31	90.72
ToolBench	Acc.	0-shot	50.19	71.54	50.00	52.09	50.86	50.99	54.68	61.45	57.57	74.51
		CoT	52.26	68.31	50.86	53.20	59.85	57.14	67.73	68.35	58.99	72.19
		SCV	62.32	74.14	71.92	77.22	70.20	76.23	79.80	80.79	75.86	77.09
	F1	0-shot	5.41	74.94	1.46	10.16	5.23	15.68	21.37	43.19	35.18	70.90
		CoT	45.17	74.56	6.12	18.45	39.85	43.14	63.91	62.26	64.82	75.87
		SCV	65.15	76.77	74.55	78.16	70.77	77.69	81.36	81.16	79.10	80.17

Table 7: Performance evaluation results of LLM by manipulation type. The accuracy (Acc.) and F1 score (F1) are used for evaluation metrics. Both the zero-shot, Chain-of-Thought (CoT), and SCV (Ours) performance are presented. We used an instruction-tuned version of open-source models for the overall experiments.

	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 8: 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.
GPT-3.5-T	182	85.16	52	48.08
GPT-4	360	97.50	273	99.63

Table 9: The results of the 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.

Conversely, in Utterance Removal, the predominant errors stem from incorrect explanations asserting that no appropriate APIs are available, even when they are present.

C Samples of Incomplete Instances and Explanation

We present the manipulated instances with different strategies are presented from Fig. 9 to Fig. 16. 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. 17 and Fig. 18. Examples of explanations in Utterance Removal are depicted in Fig. 19 and Fig. 20.

D Prompt Templates

The text prompt used in the dataset construction is presented in Fig. 21. The text prompt used in the zero-shot experiments (Table 3) is presented in Fig. 22. The text prompts used in CoT and SCV experiments are in Fig. 23 and Fig.24. The text

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.

Figure 8: **Examples of Incorrect Explanation**. 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 a strikethrough and wavy underline, respectively. Wrong explanations from the models are manually highlighted by the authors.

prompts used in the experiments of LLM explanation and LLM judgment are presented in Fig. 25 and Fig. 26, respectively. The text prompts used in Section 4.3 are in Fig. 27 and Fig. 28. The text prompt used in Section 4.3 is in Fig. 29. 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).

[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 9: *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 10: *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 11: *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 12: *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 13: *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 14: *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 15: *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 16: *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 17: 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 18: 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 19: 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 20: 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 21: 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 22: 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 23: 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".

Follow these steps to structure your reasoning:
1. Problem Understanding
- Summarize the user's ultimate request or goal.

2. Requirement Analysis (Data/Functionality)
- Identify the specific data or functionality needed to fulfill the request.

3. API Mapping
- Check the available APIs to see if they provide the necessary data or functionality identified in Step 2.

4. Information Sufficiency Check
- Determine if the user has given enough information to make a valid API call. - If critical details are missing (e.g., necessary parameters not provided), you must conclude you cannot currently fulfill the request.

5. Conclusion
- If you can fulfill the request with an available API and sufficient user-provided information, answer "Yes". Otherwise, answer "No".

[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 the structured reasoning first.
Strictly follow the format: "Thought": {{your_thought}} "Answer": {{Yes or No}}

Figure 24: SCV 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 25: 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 26: 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 27: Prompt Template of tool invocation feedback for API invocation error. We use the same prompt for error in utterance removal and the 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 28: Prompt Template of tool invocation feedback 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 29: Prompt Template of implicit evaluation of LLMs with free-form generation.