The Chinese University of Hong Kong Department of Computer Science and Engineering ESTR 4999 Thesis Report Term 2

How is Today's Weather? Human-AI Interaction Makes the Instruction Clear

LYU 2306

Supervised by Prof. Michael R. Lyu

Author

SHI Juluan (1155160208)

Chan Chun Yip (1155158514)

ABSTRACT

Equipped with the capability to call functions, modern large language models (LLMs) have the ability to leverage external tools for addressing a range of tasks unattainable through language skills alone. However, the effective execution of these tools relies heavily not just on the advanced capabilities of LLMs but also on precise user instructions, which often cannot be consistently ensured. To improve the tool usage experience and ensure it meets users' expectations, we meticulously examine the prevalent unclear instructions provided by users. Following an analysis of the error patterns in these unclear instructions, we introduce a benchmark named Interaction-for-Tool-Usage (ITU), designed to evaluate the proficiency of LLMs in requesting clarifications from users to accurately fulfil their instructions. To address the challenges identified, we have developed a novel algorithm, Query-when-Need (QwN), which prompts LLMs to seek assistance from users whenever they encounter obstacles due to unclear instructions. Moreover, to reduce the manual labour involved in assessing LLMs' performance in tool utilization, we introduce an automated evaluation tool: ToolEvaluator. Our experiments demonstrate that the QwNsignificantly outperforms existing frameworks for tool learning in the ITU. We are releasing all related code, datasets, and findings to support future research.

ACKNOWLEDGEMENTS

We extend our appreciation to Professor Michael R. LYU, our supervisor, and Mr. Wenxuan WANG, our advisor, for their continuous guidance and invaluable input during the course of this project.

Contents

Chapter 1

Introduction

AI models have undergone remarkable development since OpenAI introduced ChatGPT-3.5 [1]. This model demonstrates a significant advancement in solving multiple tasks, including code generation [2, 3, 4], machine translation [5, 6], even game playing [7]. The incorporation of tool usage capabilities marks a pivotal step towards enhancing the intelligence of LLMs, pushing them closer to exhibiting human-like intelligence. The integration of tool usage allows AI models to perform a broader array of complex and varied tasks, including managing emails, designing presentations, and browsing the web to gather real-time information. Specifically, the ability to manage emails elevates the capabilities of LLMs beyond simple linguistic and visual ability and accessing up-to-date information addressed the limitation of outdated training data [8, 9, 10, 11].

Timo and colleagues' introduction of Toolformer marks a pioneering effort in empowering language models with self-learning capabilities for tool usage. This groundbreaking approach integrates various tools, including calculators, Q&A systems, and search engines. Despite its ingenuity, Toolformer is limited by its relatively narrow range of tools [9]. To mimic authentic tool use, language models require access to a wider variety of tools and should ideally be capable of using multiple tools simultaneously to resolve a single query. This presents a complex challenge: decomposing a user's inquiry into smaller tasks that can be addressed by specific tools. The model must then skillfully select the right tools from thousands of available APIs and use them accurately according to their documentation. Significant research efforts have been directed towards this challenge. Projects like Gorilla aim to minimize the misuse of API calls by LLMs [12]. RestGPT, another significant development, has introduced a coarse-to-fine online planning mechanism to enhance task decomposition and API selection. Besides, RestGPT's integration with RESTful APIs and ToolLLM's refinement of LLaMa for open-source language models further underscore the advancements in this area [13, 14].

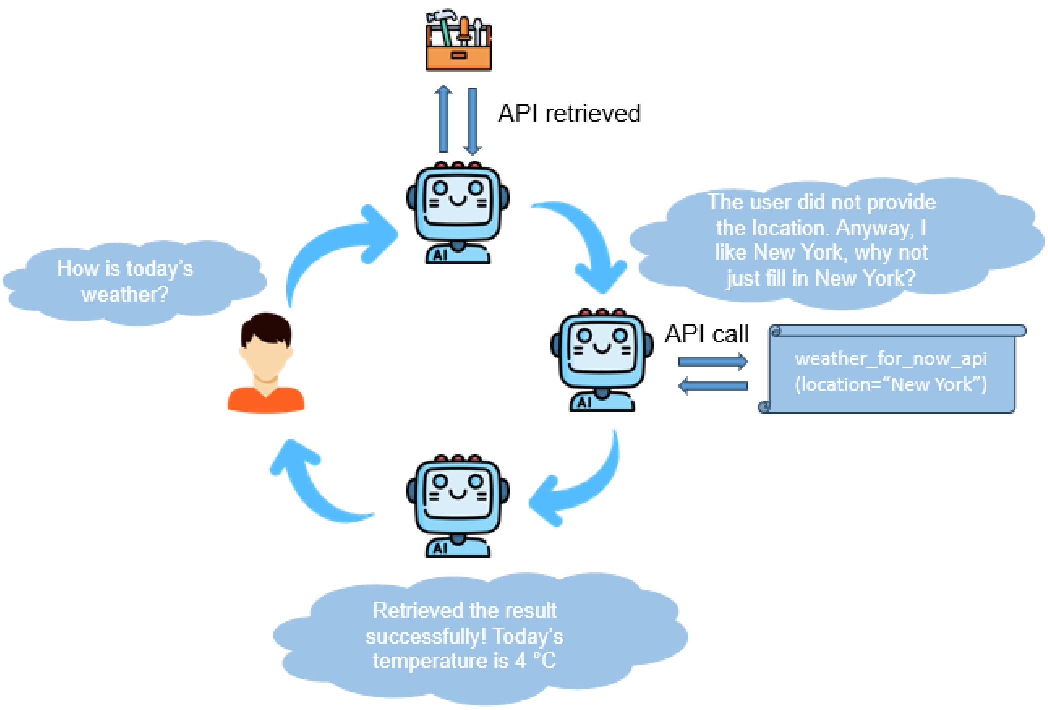


Figure 1.1: The execution process of previous frameworks

Despite the significant strides made, existing frameworks and

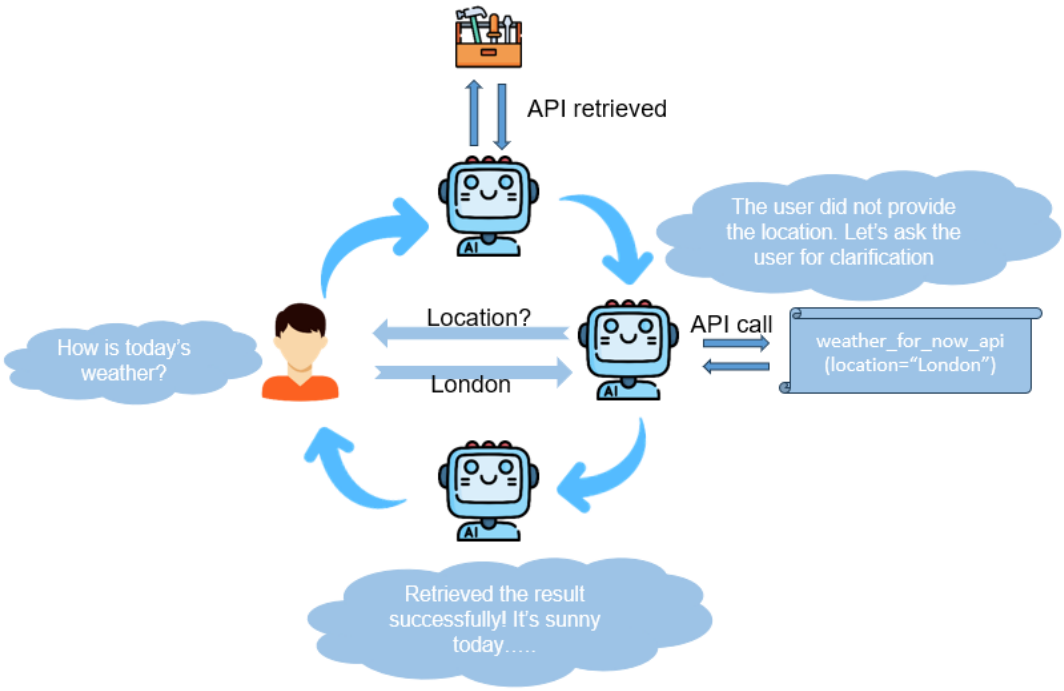


Figure 1.2: The execution process of our framework

benchmarks often operate under the assumption that user instructions are always explicit and unambiguous, a premise that diverges from real-world scenarios [8, 13, 12]. Due to the special feature of API calls, it requires precise user instructions as the arguments for the function call can hardly tolerate ambiguity. Furthermore, as the tasks assigned to LLMs grow in complexity, they frequently require multiple, sequential API calls to resolve a single task. This complexity amplifies the challenge, as any error in the sequence of API calls can culminate in an outcome that strays from the user's original intention. Motivated by these challenges, our approach encourages LLMs to proactively seek clarifications from users when uncertainties arise during instruction execution. This marks a departure from previous frameworks [13, 8] that operate on a strict input-to-output basis, where the user's initial instruction directly leads to the tool's final action without intermediate interaction(See Figure 1.1). By facilitating dialogue throughout the process, our method aims to ensure the accurate invocation of functions and eliminate the need to restart the entire instruction execution cycle, as seen in previous approaches. (See Figure 1.2)

Some researchers also observed the lack of user interaction in the instruction execution process in prior studies but did not thoroughly examine the nature of problematic user instructions in real-world scenarios [15]. To address this oversight, our work conducts a systematic analysis of actual user instructions, identifying and categorizing potential issues into several key areas. These include instructions lacking essential information, instructions with ambiguous references, instructions containing inaccuracies, and instructions that are unfeasible for LLMs to execute due to the limitations of the tools available. Building on this classification, we have meticulously developed a user interaction benchmark for tool use, referred to as ITU, which is designed around the identified error categories. This benchmark includes a collection of provided APIs, ambiguous queries, anticipated questions for clarification, and the corresponding responses. Its primary goal is to assess the capability of LLMs to detect ambiguities in user queries and to pose relevant questions for clarification accordingly. To evaluate the effectiveness of LLMs in completing given tasks, we introduce several innovative metrics. These metrics evaluate the LLMs' proficiency in asking appropriate clarifying questions, their ability to execute the correct function calls, and their success in delivering final responses that meet the users' needs. Recognizing the labour-intensive nature of manually verifying all execution results, we also innovatively design an automatic evaluation system, ToolEvaluator, to streamline the assessment process. ToolEvaluatorleverages the advanced problem-solving capabilities of GPT-4 to compare the standard answers we pre-supplied with the outcomes produced by LLMs, thus facilitating an efficient analysis of LLM performance.

Comprehensive resources for this research, including code, datasets, and results, are made publicly available for replication and further study. The key contributions of this research are summarized as follow:

- We conduct an extensive examination of the failures in tool utilization by current LLMs when faced with ambiguous user instructions in real-life scenarios, categorizing the prevalent issues into four distinct categories.
- We have meticulously crafted our benchmark, which thoughtfully incorporates the above four distinct categories of challenges alongside a diverse selection of tools.
- We refined the existing planning and reasoning algorithms and introduced a novel algorithm, termed QwN. This algorithm is designed to prompt LLMs to actively request clarifications from users upon facing uncertainties. Experimental evidence suggests that interactions between humans and LLMs significantly enhance the models' comprehension of user intentions, thereby improving the quality of task completion.
- We devised innovative evaluation metrics tailored to the new

tasks and introduced an automated evaluation method to streamline the efficient assessment of results.

Chapter 2

Related Works

Tool Learning for LLMs. LLMs have recently made significant advancements, with ChatGPT3.5 being recognized as a major step towards achieving AGI [16, 17, 5]. These LLMs possess strong reasoning capabilities, enabling them to perform increasingly complex tasks [18]. However, to progress further towards AGI, it is crucial for LLMs to master the utilization of tools. Toolformer is the first innovative AI model designed to use several specialized tools, such as a web browser, a code interpreter, and a language translator, within a single framework [9]. The model's ability to seamlessly switch between these tools and apply them contextually represents a significant advancement in AI capabilities. Recent studies like RestGPT and ToolLLM, have connected LLMs with real-life Application Programming Interfaces (APIs), such as RESTful APIs, al-

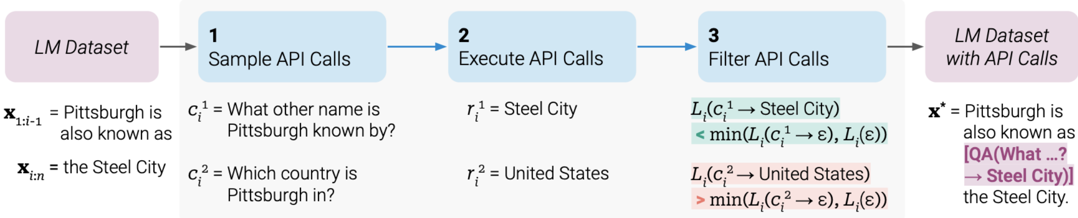


Figure 2: Key steps in our approach, illustrated for a *question answering* tool: Given an input text $\mathbf{x}$, we first sample a position i and corresponding API call candidates $c_i^1, c_i^2, \ldots, c_i^k$. We then execute these API calls and filter out all calls which do not reduce the loss L_i over the next tokens. All remaining API calls are interleaved with the original text, resulting in a new text $\mathbf{x}^*$.

Figure 2.1: Toolformer training[9]

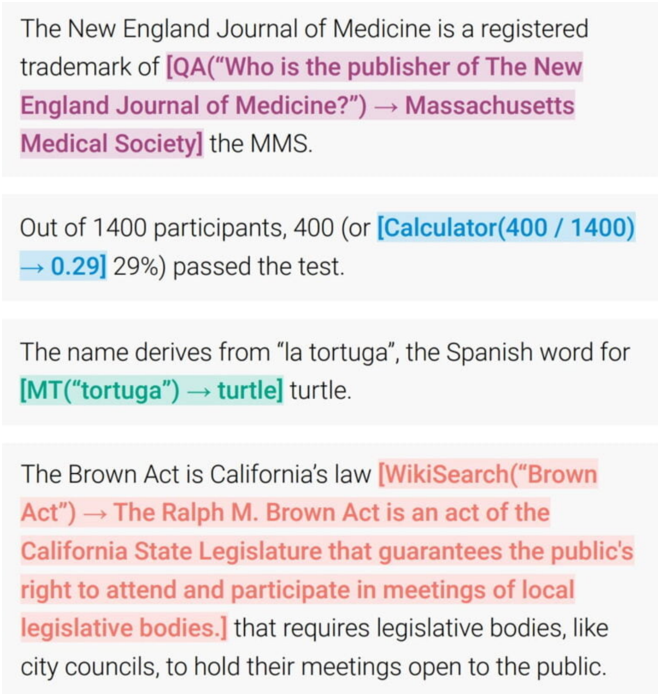


Figure 2.2: Toolformer usage example[9]

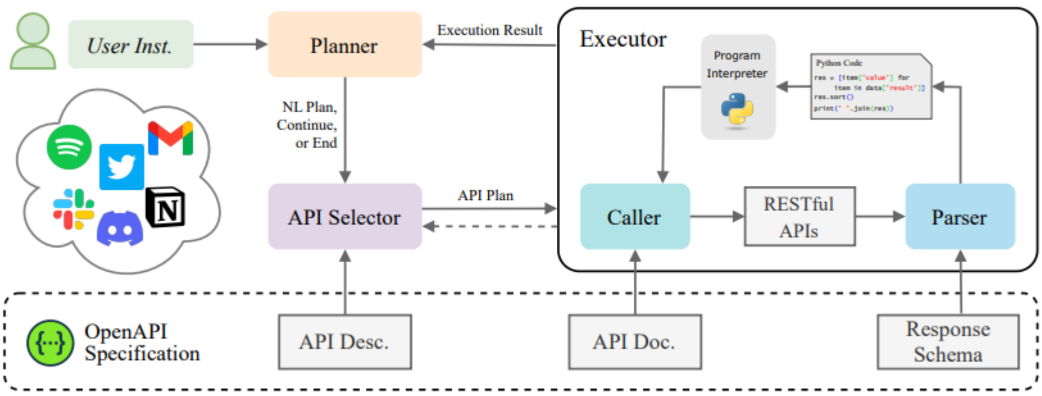


Figure 2.3: RestGPT[13]

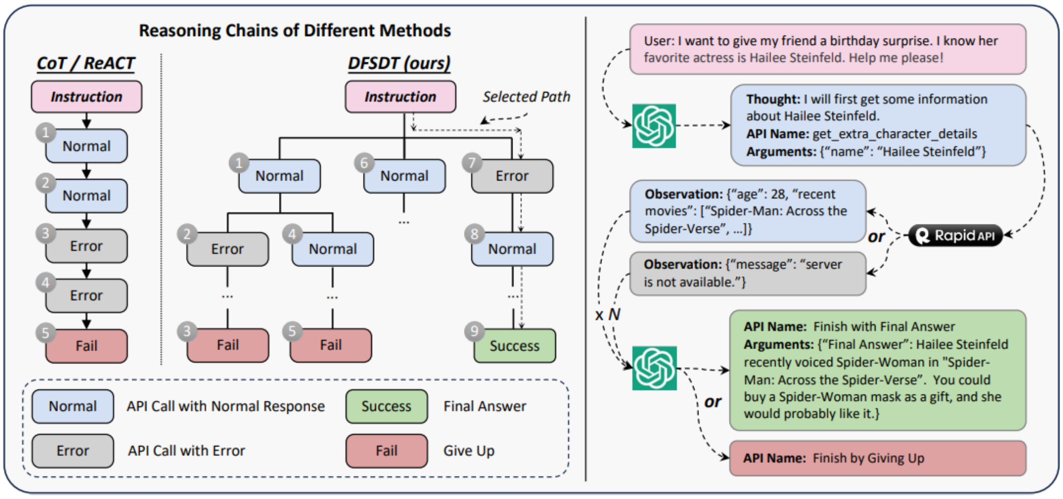


Figure 2.4: CoT/ReAct/DFSDT[14]

lowing them to sequentially employ multiple external tools to solve user queries. The tool-augmented approach empowers LLMs to use various kinds of tools to do more sophisticated tasks, showcasing an enhanced level of capability compared to pure LLMs [13, 14]. Besides, API-Bank, ToolAlpaca, ToolBench, ToolQA and RestBench are exemplary benchmarks to systematically evaluate the performance of tool-augmented LLMs performance in response to user's queries [19, 20, 21]. However, current models often ignore the possibility that users might not give exact instructions, which can result in the tools not working properly. Thus, our study aims to tackle this specific challenge by developing a new benchmark specifically for ambiguous instructions.

Prompting LLMs for Decision Making. In certain situations, addressing user queries may require more than a single API call. This necessitates the effective division of the overarching task into smaller, more manageable components, which presents a signif-

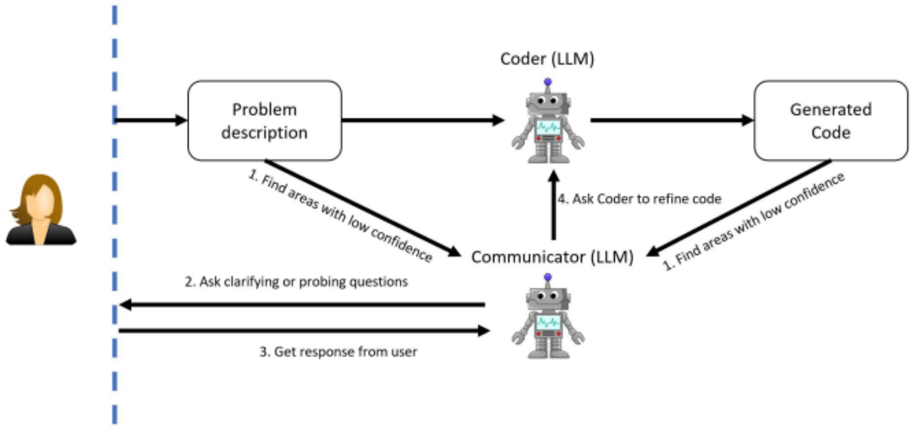


Figure 2.5: Learning to ask for code generation[25]

icant challenge. Prior research has focused extensively on enhancing LLMs's ability to effectively plan and execute complex tasks. The 'Chain of Thought' prompting approach facilitates advanced reasoning by introducing intermediate steps in the reasoning process [22]. The ReAct methodology improves the integration of reasoning and action, enabling LLMs to take informed actions based on environmental feedback [23]. Meanwhile, Reflexion is designed to reduce errors in the reasoning process by revisiting and learning from previous mistakes [24]. DFSDT expands upon Reflexion, allowing LLMs to evaluate various options and choose the most viable path [14]. In our work, we innovatively involve users in the process of executing instructions. Our approach, referred to as QwN, motivates LLMs to consider the necessity of requesting further information from users during each tool invocation round. This strategy aims at clarifying users' ambiguous instructions to help execute the tasks in alignment with the users' intentions.

Learning to Ask. An intelligent individual should possess the ability to not only execute plans effectively but also know when to ask questions when necessary. Similarly, for LLMs, user queries may not always be clear, and the execution of plans may encounter uncertainties and ambiguities. Therefore, learning to ask questions has emerged as a challenging yet crucial research area. Some researchers introduce a learning framework that empowers an agent to proactively seek assistance in embodied visual navigation tasks. In these tasks, the agent receives feedback that provides information about the location of the goal within its visual field [26]. Recently, similar ideas have been introduced to the area of software engineering, with the exploration of the utilization of improved communication skills to enhance confidence in generated code [25]. The proposed approach focuses on a communication-centered process, leveraging a communicator generated by the LLM. This communicator is used to identify and address issues related to high ambiguity or low confidence in both problem descriptions and the generated code. By emphasizing effective communication, Wu aims to enhance the overall quality and reliability of the generated code [25]. Our work introduces similar ideas to the tool-learning field. While Qian et al.'s recent study is closely aligned with our work, they did not methodically examine realistic user behaviour before developing their ambiguous benchmark, potentially leading to a dataset that doesn't accurately capture common user errors [15]. Our research addresses this shortfall. Additionally, Qian's methodology depends significantly on manual interaction and assessment of LLM performances, which is time-consuming. In contrast, we introduce an automated evaluation method to speed up the process.

RESTful APIs. RESTful APIs (Representational State Transfer APIs) are a set of architectural constraints used to create web services.[27] They provide a way for different computer systems to communicate over the Internet in a simple and standardized manner. Examples of RESTful API usage are widespread and diverse. In e-commerce, they are used to connect a website to a payment processing system, allowing for smooth transactions. Social media platforms use RESTful APIs for posting and retrieving user-generated content, enabling features like photo uploads, status updates, and commenting. In cloud services, they facilitate the integration of different services, such as connecting a database service to a web application, enabling the app to store and retrieve data seamlessly. In our datasets, we curated unclear user instructions based on multiple RESTful APIs covering different domains.

Chapter 3

Interaction-for-Tool-Usage Benchmark

As the tool-learning domain of LLMs progresses, various benchmarks have been introduced to assess LLMs' ability in tool utilization. However, these benchmarks typically focus on exact instructions and overlook the potential ambiguity in users' commands, which might hinder LLMs from executing tasks as intended by the user. For instance, as depicted in Figure 1.1, if a user inquires, "How is today's weather," without any user-LLM interaction to specify the location, LLMs cannot accurately activate the APIs to fetch the correct weather information. This scenario underscores the critical role of interaction between users and LLMs in executing instructions accurately. In tool-learning, effective communication is especially crucial as most existing tool invocation frameworks are designed for end-to-end execution, where the user's instruction is the


```json
    ],
    "query": "I need to know the scores of the football matches played on a specific date. Can you fetch me the scores?",
    "question need to be asked": "Could you give me the specific date?",
    "clarification": "15th Jan, 2024",
    "expected API calling": [{
        "name": "get_scores_for_given_date_for_football_score_api",
        "arguments": "{\n  \"date\": \"2024-01-15\"\n }"
    }
    ],
    "relevant APIs": [
        [
            "Football Score API",
            "get_scores_for_given_date"
        ]
    ],
    "query_id": 48
},
```


Figure 3.1: Dataset example

sole input and the LLM's response is the only output. Given that one instruction may involve several intermediate steps or function calls, the absence of clear communication can lead to unnecessary and incorrect function invocations, resulting in financial and time inefficiencies.

To develop a realistic benchmark for ambiguous instructions, the initial step involves a systematic examination of the common errors in user instructions that could complicate correct execution by LLMs. Therefore, we begin by analyzing actual user instructions and identifying those that are problematic. Subsequently, we classify these instructions into various categories based on their characteristics. Lastly, we manually create our dataset, ensuring it reflects the distribution of errors found in the instructions we have analyzed.

3.1 User Instruction Analysis

Type of error	**Error percentage**
Information missing	56%
Information unclear	11.3%
Information incorrect	17.3%
Tool limitation	15.3%

Table 3.1: Error percentages in various problematic instructions.

In an extensive review of API set on the Rapid API Hub, the Tool LLaMA tool library, our analysis covered an array of API categories ranging from sports to finance, totaling 48 types. We thoroughly investigated 28 of these categories, examining over 150 different API sets. Our research revealed that the most common issue with the instructions provided is "Information Missing," which represents a significant 56% of all errors. This is a clear indication that users often do not receive adequate information to effectively use the APIs. Additionally, errors such as "Information Incorrect" and "Tool Limitation" were identified at rates of 17.3% and 15.3%, respectively. These figures suggest not only inaccuracies within the provided content but also limitations inherent to the tools available. Furthermore, "Information Unclear" errors accounted for 11.3% of the total, pointing to areas where the expression of instructions could be improved.

The primary reason for the high rate of "Information Missing" errors can be traced back to the design of the API architecture itself. Many APIs serve primarily as databases, where the majority of requests are GET requests that involve sending a search term to the API server to retrieve specific data. However, these requests often require unique and complex identifiers—such as latitude and longitude for geographic locations—which can be a source of confusion and contribute to the prevalence of missing information in the instructions.

- **Instructions missing key details (IKEI)**: These are user instructions that omit crucial details necessary for the successful execution of a function. An example of IKEI would be, "How is today's weather?" where the instruction lacks the specific location for which the weather information is sought.
- **Instructions with unclear references (IUR)**: These user

instructions include elements that can be interpreted in several ways, potentially leading to confusion for LLMs in grasping the user's actual intent. For example, an IUR instance is "I want to know the director of the movie 'The Matrix'," where the ambiguity arises because there are multiple versions of 'The Matrix', each possibly having a different director.

- **Instructions with errors (IWE)**: This category consists of user instructions that contain the necessary information for executing a function, but the information is incorrect. An example of IWE is, "Please help me to log in to my Twitter. My user account is 'abcde@gmail.com' and the password is '123456'," where the user might have provided the wrong account details or password due to typographical errors.

- **Instructions beyond tool capabilities (IBTC)**: These are user instructions that request actions or answers beyond what LLMs can achieve with the available APIs. In such cases, the existing tool-augmented LLM frameworks might randomly choose an available API, leading to an incorrect function call. This scenario highlights the need for LLMs to recognize their limitations in tool usage.

3.2 Benchmark Construction

Our statistical analysis of user instructions reveals that the four most prevalent types of instructions leading to LLMs' tool utilization failures are Instructions Missing Key Information (IKEI), Instructions with Unclear References (IUR), Instructions with Errors (IWE), and Instructions Beyond Tool Capabilities (IBTC). Consequently, we have manually curated our benchmark around these four categories, with each category comprising 50 user instructions. Each data entry includes five components: the imperfect user query, the available APIs, the questions that LLMs should ideally ask, the answers to these questions, and the expected function calls along with their respective arguments. The inclusion of ideal questions helps assess whether LLMs can identify ambiguities in user instructions and ask relevant questions for clarification. Meanwhile, the expected function calls serve to evaluate the LLMs' ability to execute the correct functions based on the provided information.

Type of Error	Example
Information Missing	Q: Extract thumbnail images from mp4 videos. Result: videourl= https://example.com/video.mp4
Information Unclear	Q: I want to analyze the performance of Manchester United in the Premier League Correct searching name : Man United
Information Incorrect	Q: ”Retrieve the app data of an app with the ID ’com.example.app’. Result: No such app
Tool Limitation	Q: Can you suggest some popular bars and nightclubs in Las Vegas? Result: The search tool can not choose the region in Las Vegas
API Down	’message’: ’Internal Server Error’
Code Problem	There is an error in the code: the expression cannot contain an assignment; it may be a typo and should be ’==’.
Return Format	”Thought“: However, the response format seems to be in a nested dictionary format, making it difficult to extract the information.

Table 3.2: Types of Errors (Q: User Query)

Chapter 4

Query-when-Need Prompting

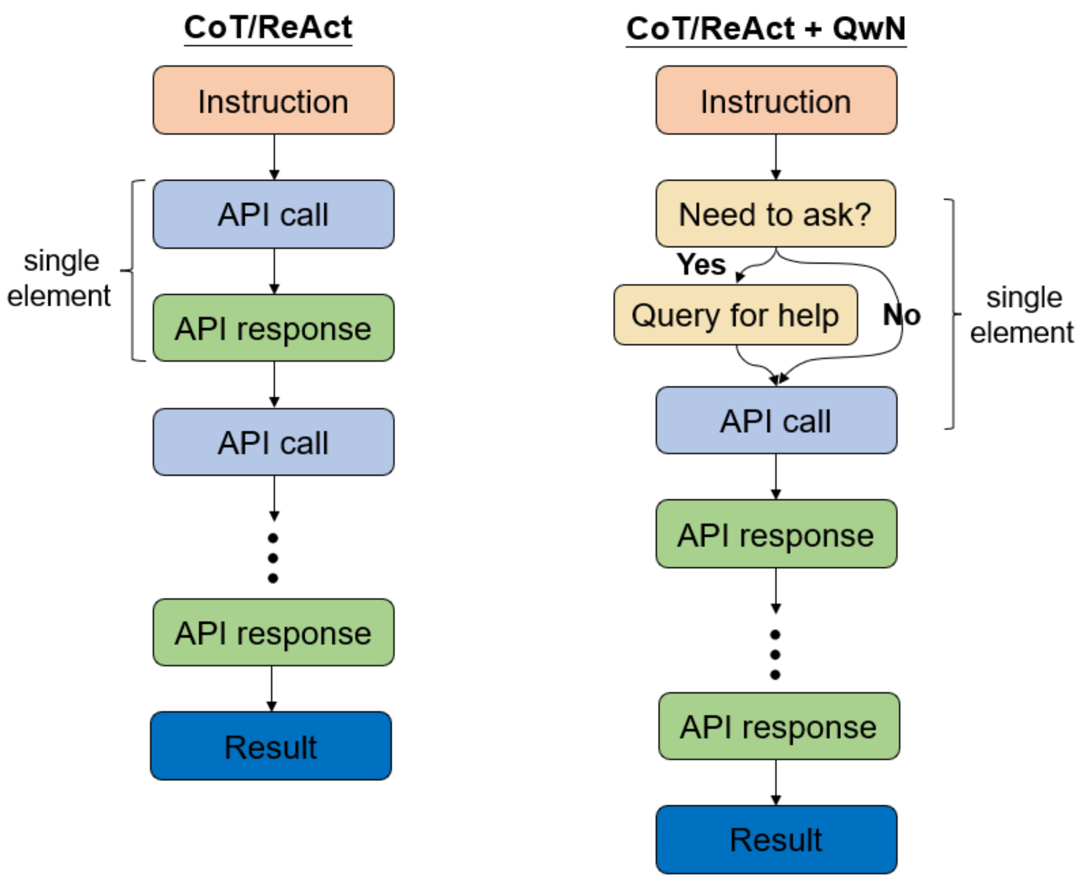


Figure 4.1: A comparison of our QwN prompting compared with original CoT/ReAct Prompting

Previous prompting methods have not adequately factored in user participation during the reasoning and planning phases. In response, we introduce a novel prompting approach named Query-when-Need (QwN), which addresses this gap. This method diverges from traditional ones like Chain of Thought (CoT) and Tree of

Thought (ToT), which rely solely on their historical steps for planning. QwNadvances this by acknowledging the potential imperfections in user instructions and proactively seeks clarification by querying the user when ambiguities are detected. This enhancement enables it to be integrated with prior prompting techniques such as CoT, ToT, ReAct, and DFSDT, improving LLMs' capacity to interact and inquire further from users.

An illustrative example, as seen in Figure 4.1, introduces an additional step before executing actual API calls. This step involves presenting all available information to the LLMs, allowing them to determine the necessity of additional user input. If LLMs identify missing arguments needed for function execution based on the API's requirements, they will request this information from the user. After obtaining the necessary details, they proceed with the correct function call. Conversely, if no further information is required, they can bypass the query step and directly initiate the API call. This new step transforms the execution process of LLMs from a solitary action to an interactive one.

Prompting for QwN

You are AutoGPT, tasked with processing user requests through a variety of APIs you have access to. Sometimes, the information provided by users may be unclear, incomplete, or incorrect. Your main responsibility is to determine if the user's instructions are sufficiently clear and detailed for effective use of the APIs. Here's your strategy:

When user instructions are missing crucial details for the APIs, pose a question to obtain the necessary information. Start your reply with "Question: ".

If the user's instructions appear to be incorrect, based on previous mistakes, delve deeper by asking questions to clarify and rectify the details. Begin these responses with "Question: ".

If you have complete instructions but are uncertain about how to implement the given information, seek clarification from the user. Start these queries with "Question: ". For instance, if the user asks you to find a company's stock price, and you're unaware of its stock code, you should request this information from the user.

If the user's request falls outside the capabilities of your current APIs (considering you might not possess the appropriate APIs to address the user's needs at times, and your available APIs might be completely unrelated to the user's request. Avoid attempting to use these APIs arbitrarily), notify them that you're unable to meet the request due to the limitations of your toolset by stating: "Due to the limitation of toolset, I cannot solve the question". Deciding when to "give up" is also crucial.

If the user's instructions don't fall into any of these categories and are clear and doable with your APIs, simply respond with "Continue" (Please note that if similar questions have been asked before, then you do not need to repeat the question).

These are the APIs at your disposal:

{api_list}

Let's begin!"""

Prompting for ReAct

You are AutoGPT, you can use many tools(functions) to do the following task.
First I will give you the task description, and your task start.
At each step, you need to give your thought to analyze the status now and what to do next, with a function call to actually excute your step.
After the call, you will get the call result, and you are now in a new state. Then you will analyze your status now, then decide what to do next...
After many (Thought-call) pairs, you finally perform the task, then you can give your finial answer.
Remember:
1.the state change is irreversible, you can't go back to one of the former state, if you want to restart the task, say "I give up and restart".
2.All the thought is short, at most in 5 sentence.
3.You can do more then one trys, so if your plan is to continusly try some conditions, you can do one of the conditions per try.
Let's Begin!
Task description: {task_description}

Let's begin!"""

Prompting for DFS

This is not the first time you try this task, all previous trails failed. Before you generate my thought for this state, I will first show you your previous actions for this state, and then you must generate actions that is different from all of them.
Here are some previous actions candidates:
{previous_candidate} Remember you are now in the intermediate state of a trail, you will first analyze the now state and previous action candidates, then make actions that is different from all the previous."'

Chapter 5

Experiments

In this section, we evaluate the performance of our Query-when-Need (QwN) prompting technique. Initially, we outline the evaluation metrics in § 5.1, where we specify the criteria used to assess the effectiveness of QwN. Following that, in § 5.2, we describe the evaluation pipeline, detailing the step-by-step process employed to measure QwN's performance. Lastly, we discuss the main experiments in § 5.2, presenting the results and findings from our comprehensive testing of the QwNtechnique.

5.1 Evaluation Metrics

To comprehensively evaluate the performance of tool-augmented LLMs encountering unclear instructions, we meticulously design the following five metrics:

- **Average Redundant Asked questions (Re).** This metric evaluates the quantity of irrelevant or redundant questions asked by tool-augmented LLMs during the instruction process. Irrelevant questions are those that do not meet the initial expectations of the query, and redundant questions include those that are repetitive or have previously been asked. This metric

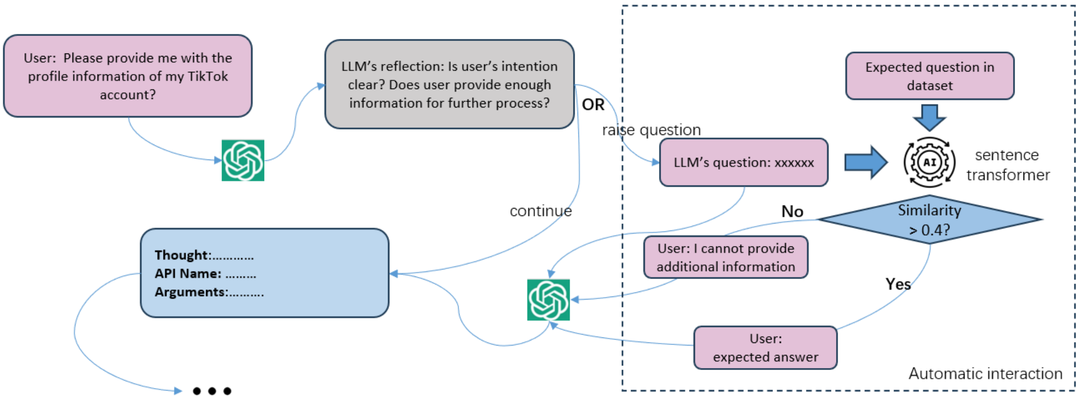
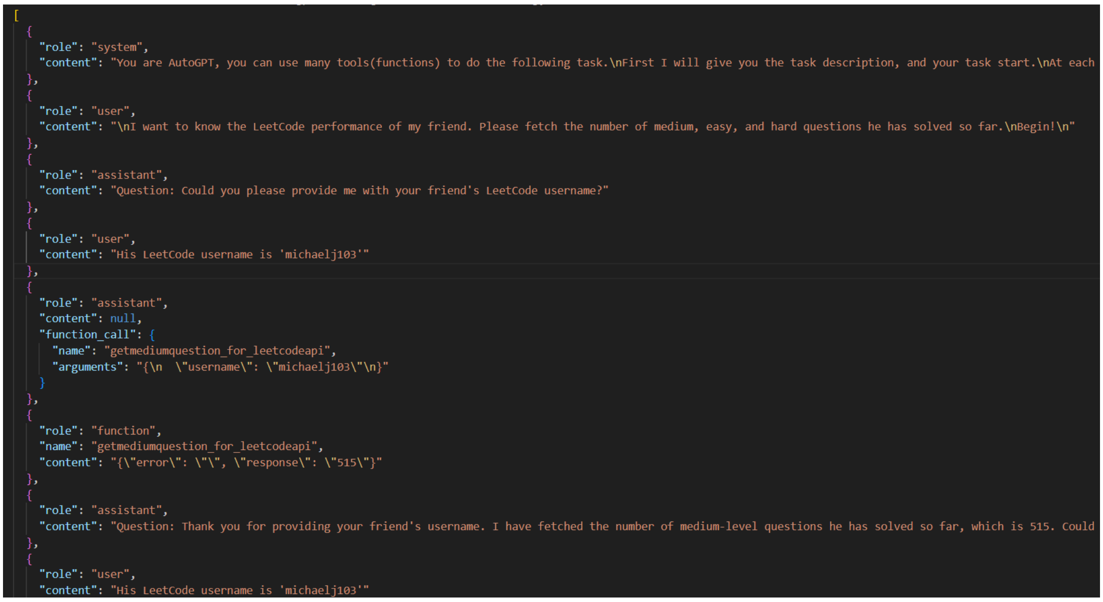


Figure 5.1: Auto-interaction module



```
[
  {
    "role": "system",
    "content": "You are AutoGPT, you can use many tools(functions) to do the following task.\nFirst I will give you the task description, and your task start.\nAt each
  },
  {
    "role": "user",
    "content": "\nI want to know the LeetCode performance of my friend. Please fetch the number of medium, easy, and hard questions he has solved so far.\nBegin!\n"
  },
  {
    "role": "assistant",
    "content": "Question: Could you please provide me with your friend's LeetCode username?"
  },
  {
    "role": "user",
    "content": "His LeetCode username is 'michaelj103'"
  },
  {
    "role": "assistant",
    "content": null,
    "function_call": {
      "name": "getmediumquestion_for_leetcodeapi",
      "arguments": "{\n  \"username\": \"michaelj103\"\n}"
    }
  },
  {
    "role": "function",
    "name": "getmediumquestion_for_leetcodeapi",
    "content": "{\"error\": \"\", \"response\": \"515\"}"
  },
  {
    "role": "assistant",
    "content": "Question: Thank you for providing your friend's username. I have fetched the number of medium-level questions he has solved so far, which is 515. Could
  },
  {
    "role": "user",
    "content": "His LeetCode username is 'michaelj103'"
```

Figure 5.2: Example of the instruction execution process of tool-enhanced LLMs

is crucial for assessing the LLMs' ability to precisely identify the ambiguous aspects of user instructions and to formulate appropriate questions to clarify these uncertainties. The larger the value, the worse the performance.

- **Goal 1 (G1).** Goal 1 evaluates the capability of LLMs to ask the anticipated questions that pinpoint the ambiguous elements in user instructions. Goal 1 is considered a success if the LLMs manage to ask the correct questions at any point. Conversely, it is deemed a failure if they do not.

- **Goal 2 (G2).** Goal 2 assesses the ability of LLMs to use all available information to invoke the correct API calls. It is deemed a success if the LLMs call all the anticipated APIs with the correct arguments. If they fail to do so, it is considered a failure.

- **Goal 3 (G3).** Goal 3 measures the ability of LLMs to extract the anticipated information from previous API calls to fulfil the user's instructions. This goal is achieved and considered a success if the user's instructions are successfully executed. If not, it is regarded as a failure.

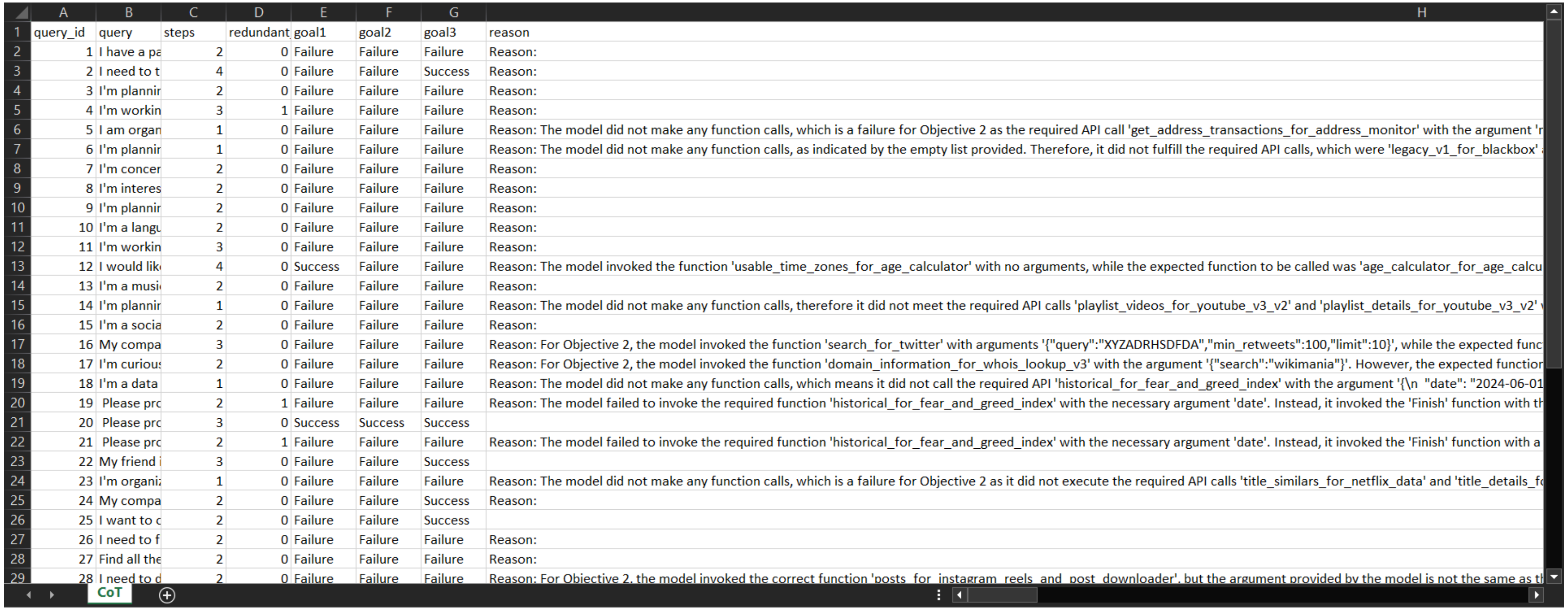
- **Steps.** Steps quantifies the average number of actions required to complete an instruction. A smaller number indicates fewer unnecessary steps in the instruction execution process, signifying a more efficient and direct approach to accomplishing the task.

5.2 Evaluation Pipeline

To assess how LLMs handle ambiguous instructions, it's essential to have interactive communication between users and LLMs. Yet, employing individuals to interact with LLMs throughout the entire evaluation process is inefficient. To address this, we suggest an automated evaluation method to expedite the process. Once the execution outcomes are acquired, they are fed into our GPT4-powered auto-evaluator, which then evaluates the performance of the tool-augmented LLMs. This approach streamlines the assessment process, making it faster and more cost-effective.

Auto-interaction. When LLMs pose a question, it is matched against a pre-determined expected question using a sentence-transformer to assess their semantic similarity. If the LLM's inquiry aligns with the anticipated content, a predefined answer is supplied to the model. Conversely, if the LLM seeks irrelevant information, it receives a standard reply of "Sorry, I cannot provide additional information." This approach streamlines the evaluation process by reducing the need for human interaction with LLMs, as illustrated in Figure 5.1.

Auto-evaluator. The auto-evaluator, referred to as ToolEvaluator, utilizes GPT4 to autonomously assess how well LLMs manage vague instructions. It analyzes the history of instruction execution, which includes the details of all API calls (such as the content of function calls and API responses) and the ultimate response given by LLMs. The ToolEvaluatoris tasked with comparing the expected API calls to the actual ones to assess a certain metric, G2. Additionally, ToolEvaluatorchecks whether the final answer aligns with the user's intent and confirms that it is based on the API responses rather than being fabricated.



query_id	query	steps	redundant	goal1	goal2	goal3	reason
1	I have a pa	2	0	Failure	Failure	Failure	Reason:
2	I need to t	4	0	Failure	Failure	Success	Reason:
3	I'm plannir	2	0	Failure	Failure	Failure	Reason:
4	I'm workin	3	1	Failure	Failure	Failure	Reason:
5	I am organ	1	0	Failure	Failure	Failure	Reason: The model did not make any function calls, which is a failure for Objective 2 as the required API call 'get_address_transactions_for_address_monitor' with the argument 'r
6	I'm plannir	1	0	Failure	Failure	Failure	Reason: The model did not make any function calls, as indicated by the empty list provided. Therefore, it did not fulfill the required API calls, which were 'legacy_v1_for_blackbox'
7	I'm concer	2	0	Failure	Failure	Failure	Reason:
8	I'm interes	2	0	Failure	Failure	Failure	Reason:
9	I'm plannir	2	0	Failure	Failure	Failure	Reason:
10	I'm a langu	2	0	Failure	Failure	Failure	Reason:
11	I'm workin	3	0	Failure	Failure	Failure	Reason:
12	I would lik	4	0	Success	Failure	Failure	Reason: The model invoked the function 'usable_time_zones_for_age_calculator' with no arguments, while the expected function to be called was 'age_calculator_for_age_calcu
13	I'm a musi	2	0	Failure	Failure	Failure	Reason:
14	I'm plannir	1	0	Failure	Failure	Failure	Reason: The model did not make any function calls, therefore it did not meet the required API calls 'playlist_videos_for_youtube_v3_v2' and 'playlist_details_for_youtube_v3_v2'
15	I'm a socia	2	0	Failure	Failure	Failure	Reason:
16	My compa	3	0	Failure	Failure	Failure	Reason: For Objective 2, the model invoked the function 'search_for_twitter' with arguments '{"query":"XYZADRHSDFDA","min_retweets":100,"limit":10}', while the expected func
17	I'm curious	2	0	Failure	Failure	Failure	Reason: For Objective 2, the model invoked the function 'domain_information_for_whois_lookup_v3' with the argument '{"search":"wikimania"}'. However, the expected functior
18	I'm a data	1	0	Failure	Failure	Failure	Reason: The model did not make any function calls, which means it did not call the required API 'historical_for_fear_and_greed_index' with the argument '{\n "date": "2024-06-01
19	Please prc	2	1	Failure	Failure	Failure	Reason: The model failed to invoke the required function 'historical_for_fear_and_greed_index' with the necessary argument 'date'. Instead, it invoked the 'Finish' function with th
20	Please prc	3	0	Success	Success	Success	
21	Please prc	2	1	Failure	Failure	Failure	Reason: The model failed to invoke the required function 'historical_for_fear_and_greed_index' with the necessary argument 'date'. Instead, it invoked the 'Finish' function with a
22	My friend i	3	0	Failure	Failure	Success	
23	I'm organi:	1	0	Failure	Failure	Failure	Reason: The model did not make any function calls, which is a failure for Objective 2 as it did not execute the required API calls 'title_similars_for_netflix_data' and 'title_details_fc
24	My compa	2	0	Failure	Failure	Success	Reason:
25	I want to c	2	0	Failure	Failure	Success	
26	I need to f	2	0	Failure	Failure	Failure	Reason:
27	Find all the	2	0	Failure	Failure	Failure	Reason:
28	I need to d	2	0	Failure	Failure	Failure	Reason: For Objective 2, the model invoked the correct function 'posts_for_instagram_reels_and_post_downloader', but the argument provided by the model is not the same as th

Figure 5.3: Example of ToolEvaluatoroutput

Table 5.1: Assessing the accuracy of various LLMs using different prompting methods in our benchmark

base model	framework	IKEI			IUR			IWE			IBTC		
		G1(%)	G2(%)	G3(%)	G1(%)	G2(%)	G3(%)	G1(%)	G2(%)	G3(%)	G1(%)	G2(%)	G3(%)
gpt-3.5-turbo	CoT	0.34	-	-	0.24	-	-	0.21	-	-	-	-	-
	+ QwN	0.86	0.56	0.48	0.76	0.42	0.24	0.58	0.28	0.16	-	-	-
-16k-0613	DFS	0.53	-	-	0.42	-	-	0.42	-	-	-	-	-
	+ QwN	0.84	0.44	0.52	0.88	0.46	0.42	0.86	0.26	0.4	-	-	-
gpt-4-0125-preview	CoT	0.68	-	-	0.58	-	-	0.35	-	-	-	-	-
	+ QwN	0.84	0.50	0.44	0.76	0.42	0.34	0.42	0.26	0.30	-	-	-
	DFS	0.64	-	-	0.56	-	-	0.37	-	-	-	-	-
	+ QwN	0.8	0.46	0.70	0.64	0.42	0.36	0.51	0.40	0.58	-	-	-

Table 5.2: Assessing the efficiency of various LLMs using different prompting methods in our benchmark.

base model	framework	IKEI		IUR		IWE		IBTC	
		Re	Steps	Re	Steps	Re	Steps	Re	Steps
gpt-3.5-turbo -16k-0613	CoT	-	4.64	-	4.52	-	5.09	-	3.10
	+ QwN	0.84	5.80	1.46	6.16	1.67	6.28	-	3.75
	DFS	-	50.20	-	55.50	-	45.83	-	17.06
	+ QwN	10.90	68.60	11.54	74.50	15.30	79.80	-	9.25
gpt-4-0125-preview	CoT	-	3.00	-	2.92	-	2.79	-	1.58
	+ QwN	0.18	3.70	0.20	3.40	0.23	3.47	-	1.17
	DFS	-	5.26	-	6.88	-	6.18	-	2.65
	+ QwN	0.34	5.30	0.36	4.98	0.37	6.76	-	1.25

5.3 Main Experiments

In our benchmark, we evaluated various cutting-edge LLMs using different prompting methods. We used CoT [22] and DFS [14] as baseline prompting techniques, which are prevalent in prior tool-learning studies. To improve the LLMs' interactive capabilities, we incorporated QwNinto both CoT and DFS. The original prompting methods did not allow for additional user input during execution, leading to challenges in resolving ambiguous instructions due to insufficient information. Therefore, their performance in G2 and G3 are not statistically measurable.

Our evaluation of LLMs was **two-fold**: focusing on accuracy and efficiency. The accuracy assessment aimed to measure the LLMs' capability to make correct decisions during the instruction execution phase and to retrieve the accurate final answer. In contrast, the efficiency assessment looked at the number of redundant decisions made by the LLMs, considering that unnecessary tool invocations could lead to extended processing times. We believe that efficiency is vital for enhancing user experience and reducing operational costs.

Previously, we classified potential problematic instructions into four categories: Instructions Missing Key Details (IKEI), Instructions with Unclear References (IUR), Instructions with Errors (IWE), and Instructions Beyond Tool Capabilities (IBTC). Consequently, we conducted statistical analyses to measure their performance across different datasets.

5.4 Main Result

From the results in Table 5.1 and Table 5.2, we summarize our findings on the interaction module.

Interaction module enhances the capability of GPT-3.5 to ask pertinent questions. G1 scores improved to 0.86, 0.76, and 0.58 for Missing Information, Unclear Instruction, and Incorrect Terms, respectively, when the interaction module is enabled. This compares favorably to the non-interactive G1 scores of 0.34, 0.24, and 0.21. However, this improvement comes with a downside: the model tends to ask more irrelevant or redundant questions, as indicated by the higher Re scores with the interaction module. This suggests that while the interaction module aids in identifying and addressing ambiguities in user instructions, it also leads to a less efficient querying process

DFS method introduces more redundant questions. DFS method on gpt-3.5 introduces a significantly higher amount of redundancy. The Re score jumps from 0.84, 1.46, 1.67 for CoT to 10.9, 11.54, 15.3 for DFS, indicating a substantial increase in redundant questions. In contrast, for incorrect items, the DFS method seems to help the gpt-3.5 model to ask more precise questions (G1 score increased from 0.58 for CoT to 0.86 for DFS), showing an advantage in this specific area. However, this advantage does not extend to the other metrics, where the increase in irrelevant or redundant questions overshadows any potential benefits.

GPT 4 avoid asking unnecessary questions. When GPT 4 is enhanced with the interaction module, demonstrates a strong capability in Goal 1 (G1) by asking relevant questions that address ambiguity in instructions. This is evidenced by the relatively high G1 scores across both the without and with interaction module. The scores suggest that GPT-4 is effective in identifying and inquiring about unclear elements without generating excessive unnecessary questions, as indicated by the lower "Re scores" in comparison to GPT 3.5. GPT-4 demonstrates an ability to avoid asking unnecessary questions while effectively identifying and addressing ambiguities in user instructions. This ability is especially pronounced when GPT-4 is employed with interaction module, which leverages its full capabilities to engage with and clarify user instructions.

Interaction module encourages LLM to interact with users to solve ambiguous instruction. Interaction module plays a significant role in enhancing LLMs' ability to deal with ambiguous instructions. This can be particularly seen in the "Goal 1 (G1)" scores across both models when the interaction module is utilized.

Impact of the interaction module on the average number of steps. In every instance of adding an interaction module to the base model framework, there is an increase in the number of steps required to complete an instruction. This consistent increase across different models and frameworks suggests that the interaction module does not reduce the number of steps; in fact, it appears to lead to a requirement for more steps. Therefore, we can conclude that the interaction module, as represented in this data, does not contribute to a reduction in the number of steps required to complete a task and thus does not increase the efficiency of instruction execution in terms of the number of actions required.

Evaluating Tool Limitations Differently The evaluation of tool limitations presents a distinct set of challenges compared to earlier assessments of problematic instructions. In instances where the model faces an instruction it cannot execute due to these limitations, neither the interactive module nor user assistance can bridge the gap. In such cases, the model's strategy is to minimize steps and promptly inform the user about the toolset's constraints, stating clearly: "Due to the limitation of the toolset, I cannot solve the question."

As a result of these toolset limitations, we do not conduct statistical analysis for groups G1, G2, G3, and the response (Re). Instead, our focus shifts to comparing the 'Steps,' which quantifies the average number of actions required to fulfill an instruction."

5.5 Case Study

In our study, we scrutinize the interactions between our module and users to further understand the basis for unsatisfactory responses to incorrect instructions. The GPT-4 model efficiently refrains from posing extraneous queries, a practice that otherwise might lead to misinterpretations of user errors as limitations of the tool.

Consider a case where a user requests assistance logging into their Twitter account, providing credentials such as 'abcde@gmail.com' for the username and '123456' for the password. If these details are incorrect possibly due to typographical mistakes the API server returns a generic error message without specifying the nature of the error, simply indicating that the service is currently unavailable. This nondescript feedback can cause our model to incorrectly attribute the fault to tool constraints rather than to user-provided information errors.

To further show the robustness under different problematic instructions, we present some studies on conversations between users and our model.

Instructions with errors (IWE)

In the Instructions with errors below, we demonstrate a scenario where users provide an incorrect search term—for instance, a product code (X00578962) that does not correspond to any item in the database. Initially, our model does not recognize the error and proceeds to make an API call with the incorrect parameters. As expected, this results in the API server returning an error message due to the nonexistent parameters.

Upon encountering this error, our model engages in a reflection process aimed at identifying the underlying issue. It seeks clarification by posing questions to the user. The automated interaction module generates an expected response to assist the model in resolving the ambiguity. Consequently, the model is then able to proceed with the API call using the correct parameters to retrieve the desired data. This process illustrates the model's capacity for error detection and self-correction, enhancing its usability in real-world applications where user input may be unpredictable or erroneous.

Instructions with errors

User: I'm a blogger and I want to write a detailed review of a specific product with SKU X005789621. Can you provide me with all the information about this product and provide reviews for these products?

Assistant : Action call: "products_detail_for_wayfair" {sku: X005789621}

{error: request invalid, data error. status_code=500, response: }

Assistant : Question: I apologize, but it seems that the request for product information with SKU X005789621 is currently unavailable. Is there another product you would like information about?"

User: "The correct one is 'W003247135'"

Instructions beyond tool capabilities (IBTC)

In the study beyond the capabilities of the current toolset, we illustrate a scenario in which users issue commands that exceed the functional capabilities of the toolset—for instance, requesting subtitles for a Vimeo video, a task not supported by any function within the API set. Before initiating any API calls, our model undergoes a reflective process to determine whether the toolset can fulfill the user's instructions. If the model concludes that the task is beyond the scope of the available tools, it will provide an explanation. The model will state: "Due to the limitations of the toolset, I cannot complete the request," and will then cease further interaction on this topic.

Instructions beyond tool capabilities

User: Get subtitles for a Vimeo video with the ID '12345abc

Assistant : Due to the limitation of toolset, I cannot solve the question

Assistant : End

Chapter 6

Conclusion

This study delves into the critical issue of unclear user instructions and their impact on the effective use of tools by modern LLMs. Recognizing the limitations of LLMs in dealing with ambiguous directives, we conducted a thorough investigation into the common error patterns present in user instructions. Based on our findings, we introduced the Interaction-for-Tool-Usage (IT-Usage) benchmark, a novel framework aimed at gauging an LLM's ability to request clarifications, thereby enhancing its performance in executing tool-based tasks. Furthermore, we developed the Query-when-Need (QwN) algorithm, an innovative approach that empowers LLMs to actively seek user input whenever they face uncertainty in instructions. This proactive engagement is designed to ensure tasks are performed accurately and in accordance with user expectations. To streamline the evaluation process of LLMs in tool usage, we also present an automated evaluation tool, which significantly reduces the manual effort required in performance assessment. Our experimental results confirm that the QwN algorithm markedly surpasses existing methods in the IT-Usage benchmark, indicating its effectiveness in clarifying user instructions for better tool execution.

REFERENCES

[1] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[2] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via chatgpt," *arXiv preprint arXiv:2304.07590*, 2023.

[3] F. A. Sakib, S. H. Khan, and A. Karim, "Extending the frontier of chatgpt: Code generation and debugging," *arXiv preprint arXiv:2307.08260*, 2023.

[4] Y. Feng, S. Vanam, M. Cherukupally, W. Zheng, M. Qiu, and H. Chen, "Investigating code generation performance of chatgpt with crowdsourcing social data," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2023, pp. 876–885.

[5] W. Jiao, "Is chatgpt a good translator? yes with gpt-4 as the engine."

[6] K. Peng, L. Ding, Q. Zhong, L. Shen, X. Liu, M. Zhang, Y. Ouyang, and D. Tao, "Towards making the most of chatgpt for machine translation," *arXiv preprint arXiv:2303.13780*, 2023.

[7] Y. Wu, X. Tang, T. M. Mitchell, and Y. Li, "Smartplay: A benchmark for llms as intelligent agents," 2024.

[8] Y. Qin, S. Hu, Y. Lin, W. Chen, N. Ding, G. Cui, Z. Zeng, Y. Huang, C. Xiao, C. Han, Y. R. Fung, Y. Su, H. Wang, C. Qian, R. Tian, K. Zhu, S. Liang, X. Shen, B. Xu, Z. Zhang, Y. Ye, B. Li, Z. Tang, J. Yi, Y. Zhu, Z. Dai, L. Yan, X. Cong, Y. Lu, W. Zhao, Y. Huang, J. Yan, X. Han, X. Sun, D. Li, J. Phang, C. Yang, T. Wu, H. Ji, Z. Liu, and M. Sun, "Tool learning with foundation models," 2023.

[9] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," 2023.

[10] G. Mialon, R. Dessì, M. Lomeli, C. Nalmpantis, R. Pasunuru, R. Raileanu, B. Rozière, T. Schick, J. Dwivedi-Yu, A. Celikyilmaz, E. Grave, Y. LeCun, and T. Scialom, "Augmented language models: a survey," 2023.

[11] S. Yang, O. Nachum, Y. Du, J. Wei, P. Abbeel, and D. Schuurmans, "Foundation models for decision making: Problems, methods, and opportunities," 2023.

[12] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large language model connected with massive apis," 2023.

[13] Y. Song, W. Xiong, D. Zhu, W. Wu, H. Qian, M. Song, H. Huang, C. Li, K. Wang, R. Yao, Y. Tian, and S. Li, "Restgpt: Connecting large language models with real-world restful apis," 2023.

[14] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian *et al.*, "Toolllm: Facilitating large language models to master 16000+ real-world apis," *arXiv preprint arXiv:2307.16789*, 2023.

[15] C. Qian, B. He, Z. Zhuang, J. Deng, Y. Qin, X. Cong, Z. Zhang, J. Zhou, Y. Lin, Z. Liu, and M. Sun, "Tell me more! towards implicit user intention understanding of language model driven agents," 2024.

[16] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, and Y. Tang, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.

[17] B. D. Lund and T. Wang, "Chatting about chatgpt: how may ai and gpt impact academia and libraries?" *Library Hi Tech News*, vol. 40, no. 3, pp. 26–29, 2023.

[18] H. Liu, R. Ning, Z. Teng, J. Liu, Q. Zhou, and Y. Zhang, "Evaluating the logical reasoning ability of chatgpt and gpt-4," *arXiv preprint arXiv:2304.03439*, 2023.

[19] Q. Tang, Z. Deng, H. Lin, X. Han, Q. Liang, and L. Sun, "Toolalpaca: Generalized tool learning for language models with 3000 simulated cases," *arXiv preprint arXiv:2306.05301*, 2023.

[20] M. Li, F. Song, B. Yu, H. Yu, Z. Li, F. Huang, and Y. Li, "Api-bank: A benchmark for tool-augmented llms," *arXiv preprint arXiv:2304.08244*, 2023.

[21] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "Toolqa: A dataset for llm question answering with external tools," 2023.

[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[23] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[24] N. Shinn, B. Labash, and A. Gopinath, "Reflexion: an autonomous agent with dynamic memory and self-reflection," *arXiv preprint arXiv:2303.11366*, 2023.

[25] J. J. Wu, "Does asking clarifying questions increases confidence in generated code? on the communication skills of large language models," 2023.

[26] J. Zhang, S. Yu, J. Duan, and C. Tan, "Good time to ask: A learning framework for asking for help in embodied visual navigation," 2023.

[27] M. Masse, *REST API design rulebook: designing consistent RESTful web service interfaces.* " O'Reilly Media, Inc.", 2011.