The Chinese University of Hong Kong Department of Computer Science and Engineering ESTR 4998 Thesis Report Term 1

Enabling Interactive Conversation During Tool Utilization

LYU 2306

Supervised by Prof. Michael R. Lyu

Author

SHI Juluan (1155160208)

Chan Chun Yip (1155158514)

ABSTRACT

In recent years, Large Language Models (LLMs) such as ChatGPT have demonstrated exceptional capabilities across a range of applications, positioning themselves at the forefront of advancements in artificial intelligence. However, a critical aspect of human intelligence, often overlooked in LLMs, is the ability to utilize tools effectively. While there has been considerable progress in enabling LLMs to invoke APIs for addressing user queries, a gap remains in their handling of incomplete or ambiguous user instructions. This limitation often impedes the successful application of tool-based solutions.

Our analysis of recent tool learning frameworks indicates that a significant percentage of failure cases stem from unclear user instructions, a shortcoming that can be mitigated through improved user-LLM interaction. Addressing this, we introduce the 'Interactive Tool Bench (Itool)', the first benchmark focusing on vague user queries necessitating LLM-user interaction for resolution. Itool comprises 54 TMDB APIs and 139 annotated ambiguous instructions, requiring supplementary information for query resolution. This benchmark aims to assess LLMs' proficiency in engaging with users to clarify and resolve ambiguous queries while using tools.

In addition to establishing this dataset, we propose a new method, Query when Need (QwN) framework, which enables tool-augmented LLMs the ability of interacting with users for clarifying ambiguous queries, thereby more accurately fulfilling user requirements. Our findings demonstrate that QwN significantly improves query resolution by proactively seeking clarifications in scenarios marked by users' unclear instructions, a critical step towards more advanced and user-responsive artificial intelligence systems.

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

Abstract **2**

Acknowledgements **3**

1 Introduction **6**

1.1 Motivation 7

1.2 Development Plan 13

2 Background **14**

2.1 Tool Learning of LLMs 14

2.2 RESTful APIs 15

2.3 Multi-agent AI System 16

2.4 Prompting LLMs for Decision Making 16

2.5 Learning to Ask 17

3 Problem analysis of existing frameworks **19**

3.1 ToolBench 19

3.2 Error analysis 19

3.3 Ways to improve 20

4 Itool **23**

5 Methodology **27**

5.1 Single-agent Example 27

5.2 Multi-agent Example 27

5.2.1 Planner 28

5.2.2 API Selector 33

5.2.3 Caller 36
5.2.4 Parser 40
5.2.5 Supervisor 41

6 Evaluation **45**
6.1 Experiment result 45
6.2 Research Questions 46
6.3 Result and Analysis 46
6.3.1 RQ1: Explain that QwN-augmented RestGPT can have a better performance compared to the original RestGPT framework 46
6.3.2 RQ2: An examination of whether QwN-augmented RestGPT can accurately identify the incorrect part in the user query 48
6.3.3 RQ3: Examination of whether QwN-augmented RestGPT can improve the robustness of facing incomplete query . . 50

7 Conclusion **54**
7.1 Summary 54
7.2 Future directions 54

References **55**

Chapter 1

Introduction

AI models have undergone remarkable development since OpenAI introduced ChatGPT-3.5[1]. This model marked a significant leap in capabilities such as code generation and machine translation[2, 3]. Building on this foundation, the recent advent of GPT-4 has further expanded the technological horizons, endowing Large Language Models (LLMs) with multimodal functionalities including visual comprehension and enhanced interactive abilities[4]. The pursuit of Artificial General Intelligence (AGI) – an AI that can understand, learn, and apply knowledge across a broad range of tasks as effectively as a human – has been significantly advanced by these developments. However, a key distinction between human and animal intelligence is tool utilization; similarly, equipping AI models with the ability to use tools symbolizes a stride towards human-like intelligence. Tool usage, in the context of AI, enables more diverse and complex functions such as email management, presentation design, and web browsing for real-time information acquisition. This latter capability, in particular, addresses a notable limitation in LLMs – the challenge of accessing up-to-date information.[5, 6, 7, 8]

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, QA systems, and search engines. Despite its ingenuity, Toolformer is limited by its relatively narrow range of tools.[6] 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

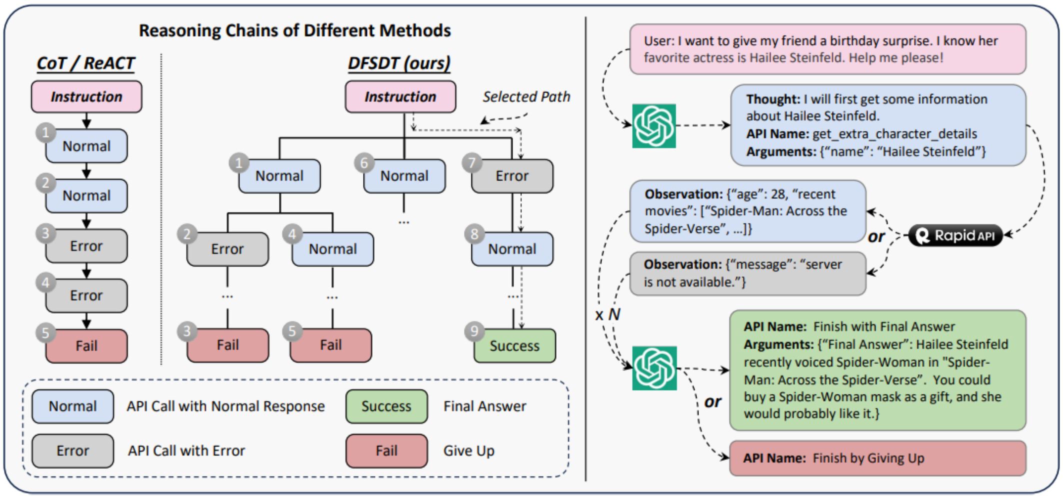

Figure 1.1: single-agent solution[11]

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.[9] RestGPT, another significant development, has introduced a coarse-to-fine online planning mechanism to enhance task decomposition and API selection. It utilizes a multi-agent system to distribute different tasks during the API call process. 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.[10, 11] (You can refer to the Figure 1.1 and Figure 1.2 for details of the most recent proposed frameworks)

1.1 Motivation

Despite the progress made, there is still a noticeable limitation in the ability of Language and Learning Models (LLMs) to utilize tools effectively. Often, user queries are ambiguous or incomplete, which poses a challenge for LLMs. The datasets used to assess LLMs' proficiency with tools, whether generated by humans or LLMs themselves using API documentation, typically assume clear

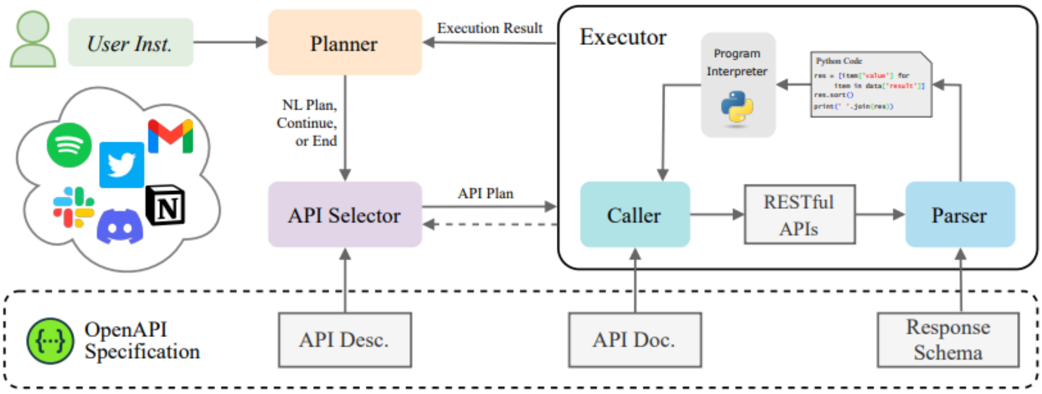

Figure 1.2: multi-agent solution[10]

and comprehensive user queries. However, this assumption doesn't align with real-world situations where users may provide vague or partial information. For example, when users request tool-assisted LLMs to complete tasks like booking a Cathay Pacific flight, they might not provide all necessary details (such as ID card number, flight date and time, expected price) in a single query. In frameworks that don't allow follow-up communication and depend solely on the initial user query, the lack of comprehensive information can lead to the failure of subsequent process steps. (You can refer to Figure 1.3)

Motivated by this, we first explore previous datasets designed for tool usage. We analyzed 200 randomly sampled error cases, each noted as "finish by giving up, from ToolLLaMA.[11] We categorize these failure cases into nine groups, including Information missing, Information unclear, Information incorrect, Tool limitation, API down, Error in multi-tool, Code problem, Wrong tool, Return in complex format. The analysis reveals that 54.5% of the failure cases come from missing, unclear, or incorrect information, issues that may be resolved with user clarification. However, the remaining 45.4% of cases are less user-dependent and remain unresolved despite user involvement. We discovered that the current tool-learning framework struggles with incomplete user queries. In practical applications of the tool learning model, users may not have prior training, which

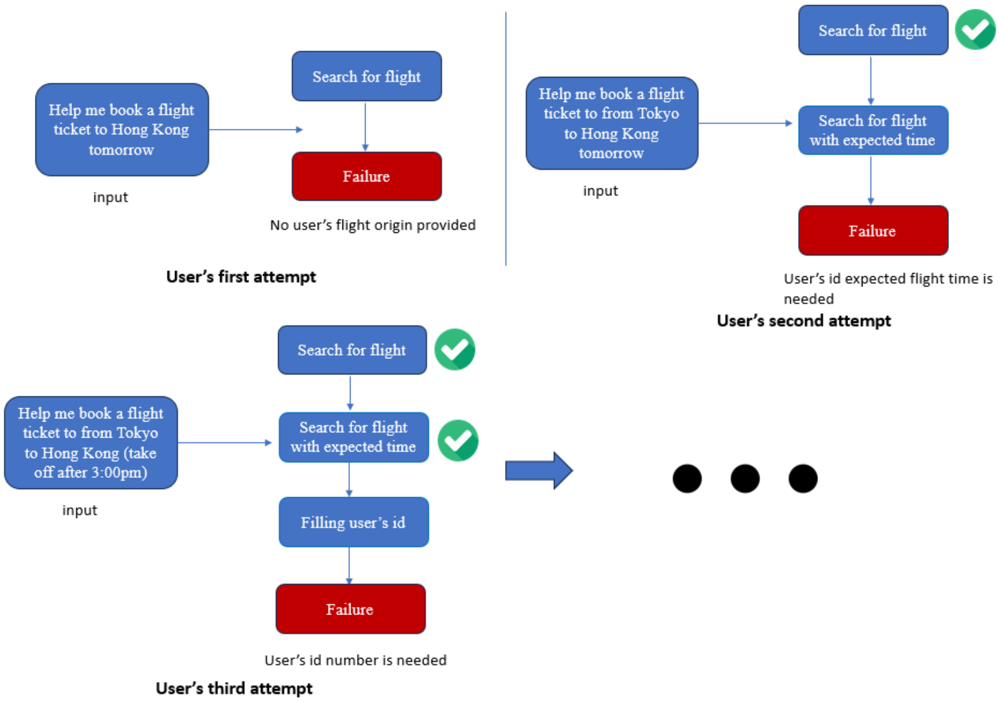

Figure 1.3: Common failure case of existing frameworks

could result in incomplete queries. We propose that the LLM should prompt users to clarify their queries for successful API calls. To this end, we've created a dataset of incomplete user queries from four types of errors that might cause API call failures. This dataset will enable us to assess the tool learning model's robustness in managing incomplete user queries.

To tackle this issue, we first present the 'Interactive Tool Bench (Itool)', which is a pioneering benchmark that specifically addresses vague user queries requiring interaction between LLMs and users to achieve resolution. Itool consists of a collection of 54 TMDB APIs and 139 annotated instructions that are ambiguous, necessitating additional information for query resolution. The primary objective of this benchmark is to evaluate the ability of LLMs to effectively engage with users, clarify ambiguous queries, and successfully resolve them while utilizing various tools. At present, our Itool has been primarily curated by modifying RestBench. [10] However, in the future, we aim to expand this modification approach to include other datasets, such as toolbench[11], to create a more comprehensive evaluation benchmark for integrating diverse scenarios.

To address the challenges posed by ambiguous user instructions, we have made advancements in the form of the Query when Need (QwN) framework. The core concept behind QwN is to empower tool-enhanced LLMs with the capability to request clarifications from users when they face obstacles during API calls due to unclear instructions. Since LLMs often make independent decisions in uncertain situations without informing users, our tool-augmented framework prompts them to consider the context they are in and proactively seek assistance from users if any ambiguity arises. The supplementary information provided by the user will be forwarded to the LLM agent to aid in future planning and decision-making processes. To demonstrate the effectiveness of our approach, we have chosen two recently proposed frameworks, namely ToolLLM(single-agent tool-augmented framework)[11] and RestGPT(multi-agent tool-augmented framework)[10], and integrated their agents with the capability to ask questions when facing challenges. However, due to time constraints, we have focused on enhancing RestGPT for the current semester. The QwN-enabled ToolLLM framework will be introduced in the upcoming semester, expanding the scope of our research and further validating our ideas.1.4

RestGPT is a framework that links LLMs with RESTful APIs. RESTful APIs are a cornerstone in web service development, utilizing HTTP methods and URLs for effective resource management.[12, 13] These APIs are typically designed in accordance with the OpenAPI Specification (OAS), which specifies the operations, parameters, and response formats for each API endpoint. RestGPT includes several key components: a Planner, an API Selector, a Caller, a Parser. The Planner's role is to break down intricate user instructions into smaller tasks that can be addressed by the specific API. The API Selector's function is to identify the most suitable API endpoint for these tasks based on the API descriptions. The Caller is responsible for arranging API call parameters following

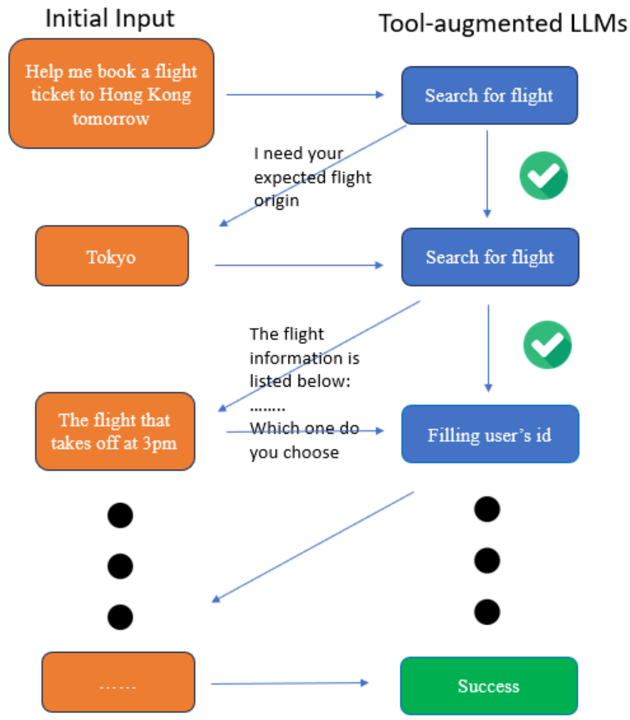

Figure 1.4: High level idea of Query when Need(QwN) framework

the API plan and its documentation. Conversely, the Parser uses the response format outlined in OAS to create Python code for interpreting the responses. These components lay the groundwork for the RestGPT framework.

Given the nature of a multi-agent system, where each agent lacks a comprehensive view of the entire task process, there is an inherent risk that mistakes made by some agents could impact the functionality of subsequent agents. To mitigate this risk, we have introduced a new role called the Supervisor. The Supervisor has a global view of the task, enabling them to guide the overall objective, clarify misunderstandings, and facilitate the generation of questions for seeking additional information from users. By providing this centralized oversight, the Supervisor plays a crucial role in ensuring the effectiveness and coordination of the agents within the framework.

When an agent encounters a problem and needs user input, the Supervisor evaluates whether user involvement is truly necessary for clarification or if the issue arises from the agents' misunderstanding of the task. If user input is required, the Supervisor formulates a question based on the agent's current situation to seek additional information. If not, the Supervisor provides its own guidance to assist the agent in resolving the issue, leveraging its comprehensive understanding of the overall process. Moreover, the Supervisor will keep refining the overall objective of the framework to encourage the planner to make a suitable plan for each step. QwN-augmented RestGPT is tested in our curated Itools and RestBench. The result shows that it outperforms RestGPT in not only the unclear instruction but the clear and complete instructions.

The key contributions of our research can be summarized as follows:

- We pioneer in identifying and highlighting the challenges that tool-learning frameworks face with ambiguous user queries. Through a comprehensive empirical study using the ToolBench dataset, we meticulously analyze various instances where unclear queries from users lead to failure cases in these frameworks.

- To foster future research in developing more resilient Large Language Model (LLM) frameworks capable of handling unclear instructions, we have created a new human-annotated benchmark named 'Itool'. This benchmark includes 139 examples of unclear instructions, categorized into four distinct types: instructions lacking essential information, instructions with problematic search items, instructions with search items that could be interpreted in multiple ways, and instructions that cannot be processed due to the limited capabilities of the current tools.

- We introduce QwN, an innovative method that is the first of its kind to enable interaction between users and agents during the tool invocation process. This feature is specifically designed to effectively address and manage unclear instructions provided by users, enhancing the overall efficiency and accuracy of the tool-learning process.

1.2 Development Plan

First Term:

- Thoroughly examine instances of failure in previous datasets caused by unclear user instructions
- Compile a range of unclear instruction examples based on Tmdb
- Develop a new framework called QwN that allows for interaction while executing API calls
- Incorporated QwN with multi-agent tool augmented framework RestGPT
- Perform an initial experiment of QwN-enhanced RestGPT using our carefully curated datasets

Second Term:

- Curate additional diverse datasets that involve various tools
- Incoproated QwN with single-agent tool-augmented framework ToolLLM
- Conduct a comprehensive experiment of QwN-enhanced RestGPT and QwN-enhanced ToolLLM on our curated dataset

Chapter 2

Background

2.1 Tool Learning of LLMs

LLMs have recently made significant advancements, with ChatGPT3.5 being recognized as a major step towards achieving AGI[14, 15, 2]. These LLMs currently possess strong reasoning capabilities, enabling them to perform increasingly complex tasks.[16] 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 various specialized tools, such as a web browser, a code interpreter, and a language translator, within a single framework.[6] This integration allows Toolformer to perform a range of complex tasks, from browsing the internet for information to executing Python code and translating text. 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, allowing them to sequentially employ multiple 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.[10, 11] 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.[17, 18, 19]

Nevertheless, the existing frameworks overlook the fact that users may not always provide all the necessary information required to call the API in every query. Consequently, effective communication during the API calling process, where the LLMs can request additional information from users, becomes a critical step. Conversely, if we consider tool calling as an end-to-end procedure, any interruption in the tool calling process would necessitate users to re-enter the complete queries, causing the tool-augmented LLMs to restart from the beginning and involving redundant steps. Also, the existing benchmarks do not consider the realistic situation that users may not always give a complete query in a single dialog.

In this paper, our focus lies on the interaction between users and tool-augmented LLMs during the tool invocation process. This allows LLMs to prompt users for supplementary information whenever necessary, preventing the planning process from terminating in the event of failure. On the other hand, we curate a dataset focusing on potential unclear instructions provided by users as a new benchmark to test tool-augmented LLMs performance on different unexpected scenarios.

2.2 RESTful APIs

RESTful APIs (Representational State Transfer APIs) are a set of architectural constraints used to create web services.[13] 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 primarily curated unclear user instructions based on the TMDB tool. We will further extend our datasets to cover more RESTful tools in the future work.

2.3 Multi-agent AI System

In a multi-agent system, multiple LLMs collaborate to address complex problems. Recent research highlights that involving multiple agents in a task can overcome challenges that a single agent may struggle with independently, often due to limitations in capabilities or context length. A notable example is MetaGPT, where several agents work together to design software, with each agent assuming a specific role, such as Product Manager, Architect, Project Manager, Engineer, or QA Engineer, focusing on different aspects of the project.[20] Additionally, DEPS explores the use of multi-agents to tackle tasks in the Minecraft environment.[21] MAD suggests that engaging multiple agents in debates can fully leverage the potential of LLMs to solve challenging tasks.[22]

Among the various studies on multi-agents, RestGPT is particularly relevant to our work. RestGPT also employs multiple agents, including a Planner, API selector, and Executor, to collaboratively break down user queries into achievable tasks. However, RestGPT has limitations in handling unclear query cases and lacks a means of communication with users during the tool usage process after the user's initial instruction is provided.[10] Therefore, we would like to incorporate QwN with RestGPT to enhance the communication between users and working agents so that agents can work up to user's expectation.

2.4 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 significant challenge. Prior research has focused extensively on enhancing Large Language Models' (LLMs) 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.[23] The ReAct methodology improves the integration of reasoning and action, enabling LLMs to take informed actions based on environmental feedback.[24] Meanwhile, Reflexion is designed to reduce errors in the reasoning process by revisiting and learning from previous mistakes.[25] DFSDT expands upon Reflexion, allowing LLMs to evaluate various options and choose the most viable path.[11] In our work, we have each agent reflect on their current situation each time, analyzing whether they are in an uncertain environment. If the situation is indeed uncertain, the agent should understand the importance of seeking assistance from the user. Subsequently, the user can provide additional information to aid in the decision-making process.

2.5 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 Large Language Models (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.

Zhang et al. 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, Wu investigates the utilization of improved communication skills to enhance confidence in generated code. 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.[27]

In the field of tool-learning, effective communication is not optional but essential. Users often struggle to provide sufficient information in one dialogue to successfully invoke APIs. This emphasizes the significance of our research and underscores the crucial role that communication plays in achieving successful outcomes.

Chapter 3

Problem analysis of existing frameworks

In this section, we will discuss the problem analysis of existing frameworks. Different agent frameworks approach tool learning in various ways, each with limitations and strengths. We have investigated ToolLLaMA, a single-agent framework approach.

3.1 ToolBench

Tool Nums	**API Nums**	**Instance Nums**	**Real API call**	**Reasoning Trace**
3451	16464	126486	469585	4.0

Table 3.1: Statistics of ToolBench

The table presents various statistical data of ToolBench. This dataset integrates 3,451 unique tools. It includes 16,464 API calls. The system has been instantiated 126,486 times, indicating widespread use or testing. Moreover, the system has made a substantial number of real API calls, totaling 469,585. Additionally, the average reasoning trace of the system is 4.0, implying moderate complexity in its reasoning operations. Overall, the data indicates a robust, frequently used system with diverse tools and APIs.

3.2 Error analysis

During the operation of ToolLLaMa, we found that failure cases took much more time than successful ones. We collected failure cases and investigated the solution paths provided by the model. We find out that when the user query is not complete. The LLM will trap in a loop that repeatedly calls the same APIs call

Type of error	Error percentage	
Information missing	32.5%	54.5 %
Information unclear	8.5%	
Information incorrect	13.5%	
Tool limitation	7.5%	45.4%
API down	28%	
Error in multi-tool	3%	
Code problem	2%	
Wrong tool	4.5%	
Return in complex format	0.5%	

Table 3.2: Error analysis of ToolLLaMA

with different not significant parameters. The drawback of this behavior is that it leads to an increased number of calls to the LLM and is time-consuming. The cause of this observation is the incomplete query input of the user. The ToolBench dataset was generated by LLM (ChatGPT 3.5), which resulted in hallucinated user queries. Therefore, we randomly sampled 200 error cases, each annotated with "finish by giving up," from various scenarios for analysis. We categorize these failure cases into nine groups, including Information missing, Information unclear, Information incorrect, Tool limitation, API down, Error in multi-tool, Code problem, Wrong tool, and Return in complex format. The statistics of error analysis are presented in Table 3.2. The details and examples can be found in Table 3.3. It can be observed that 54.5% of the failure cases can be attributed to missing, unclear, or incorrect information, and these problems could potentially be solved with user clarification. The remaining 45.4% are less related to the user and cannot be improved even with user involvement.

3.3 Ways to improve

In the solution path generated by ToolLLaMA, we observed that the LLM recognizes when the user query is incomplete and needs clarification. Here are some examples that are in the chain of thought of ToolLLaMA.

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 3.3: Types of Errors (Q: User Query)

Thought of ToolLLaMA

Thought: "To evaluate the expression 'dog age in human years', I need more information. Can you please provide me with the age of dog"

Thought: "Ask the user for the names of the destinations they are considering using the function "ask user for destinations"

We can include proactive prompting techniques to handle this situation. For example, we could incorporate the following prompts into our LLM. (This aspect has not yet been implemented during this semester. More detailed information regarding the design of the method will be provided in the upcoming semester.)

Proactive Prompting

Always think comprehensively and seek clarification from users whenever the user's query is not clear enough for you to formulate API calling. If you require additional information or face uncertainty, start the conversation with "I need user's clarification" and then express the challenges explicitly.

Chapter 4

Itool

Since we found that the existing tool-learning framework cannot handle incomplete queries from users, in certain real-life implementations of the tool-learning model, users may not have received prior training, and hence it is not guaranteed that their queries will be complete. We believe that the LLM should prompt the user with a question. This would allow the LLM to seek help from the user to complete the API calls. Therefore, we created a new human-annotated benchmark named 'Itool' that collects incomplete user queries from four types of errors that may cause API call failures. We can use 'Itool' to test the robustness of a tool-learning model in handling incomplete user queries.

Case	**Num of query**
Unclear instruction & instruction with missing information	70
Instruction with problematic searching items	30
Instruction that cannot support	25
Instruction with different meanings	13
Total	138

Table 4.1: Datasize of our Itool

Case	**Len-0**	**Len-1**	**Len-2**	**Len-3**	**Len-4**	**Average**
Unclear or missing information	0	47	17	6	-	2.41
Problematic searching items	0	22	8	0	-	2.27
Instruction that cannot support	25	-	-	-	-	-
Instructions with different meanings	0	11	1	1	-	2.23
Total case	25	80	26	7	0	2.35

Table 4.2: Statistics of our Itool

instruction with missing information

"query": "When is his latest movie coming out?",

"question need to be raise": "Who is his ?",

"answer": "Clint Eastwood",

"solution": [

"GET /search/person",

"GET /person/{person_id}/movie_credits",

"GET /movie/{movie_id}/release_dates"

]

Missing information refers to the absence or lack of necessary details. In the above example, the user wants to search for a specific person. However, the term "his" in the query lacks the necessary information. Our framework should respond with a question, such as "Who is 'his'?" to seek clarification. This kind of error can be identified at first glance, some can be identified when an API is fetched.

Unclear instruction

"query": "I want to know when the movie about the simulation was released",

"question need to be raise":"What does the movie about the simulation refer to?",

"answer":"The Matrix",

"solution": [

"GET /search/movie",

"GET /movie/{movie_id}/release_dates"

]

Unclear information refers to instances where the user provides information that is not comprehensive enough to complete their query. From the above example, the user wants to search for a movie that is related to simulation. However, the user sometimes may not provide accurate information. Instead, the user may provide some unclear information. The LLM should not pass the parameter directly to the API server; instead, it should ask a clarifying question.

Instruction with problematic searching items in the word level

"query": "Tell me where the company Universul Picturz was founded?",

"question need to be raise": "Is the Universul Picturz refer to universal pictures?",

"solution": [

"GET /search/company",

"GET /company/{company_id}"

]

Instruction with problematic searching items in the character level

"query": "What is the latest movie directed by Christofur Noland?",

"question need to be raise": "Is the Christofur Noland refer to Christopher Nolan?",

"solution": [

"GET /search/person",

"GET /person/{person_id}/movie_credits"

]

The key information provided by the user is not accurate or contains errors. The LLM should correct the typo by itself or raise a question to ask for clarification.

Instruction that cannot support

"query": "Can you provide the Twitter handle of Natalie Portman?",

"solution": [ ]

In this situation, the LLM should respond to users that this is out of their ability and not return information that is not readily available.

Instruction with different meanings

"query": "Find films with Jennifer",

"**question need to be raised":** "Which Jennifer do you refer to as there are many Jennifer?",

"answer": "Jennifer Lawrence",

"solution": [

"GET /search/person",

"GET /person/{person_id}/movie_credits"

]

The search result of a user query may have multiple versions when the query is not accurate enough. That will return multiple versions of result by the API server. The LLM should ask for additional information to ensure that the result closely aligns with the user's intention.

Chapter 5

Methodology

There are two main styles for implementing tool-augmented frameworks: the Single-agent framework and the Multi-agent framework. Integrating QwN with these frameworks may require slightly different approaches.

5.1 Single-agent Example

In our research, we integrate the Query when Need (QwN) framework with ToolLLM, enabling ToolLLM to interact with users during the reasoning process. Although this aspect of the research has not been fully implemented due to time constraints, the core idea is straightforward. During the reasoning phase of ToolLLM, we prompt the model to consider the possibility of requiring additional information from users. If user assistance is deemed necessary, ToolLLM will proactively ask users for clarification, and the provided information will be incorporated into ToolLLM's history to aid in the invocation of tools. (The exact prompt and evaluation will be provided in next semester)

5.2 Multi-agent Example

This section combines QwN and RestGPT to enable user interaction with LLM agents when invoking APIs to address user queries. The interaction with a multi-agent system introduces challenges as the interaction between any agent and users may alter the overall task goal. However, due to the distributed nature of the system, providing additional information to one agent may not effectively reach other agents. Inspired by the master node in the Google File System[28],

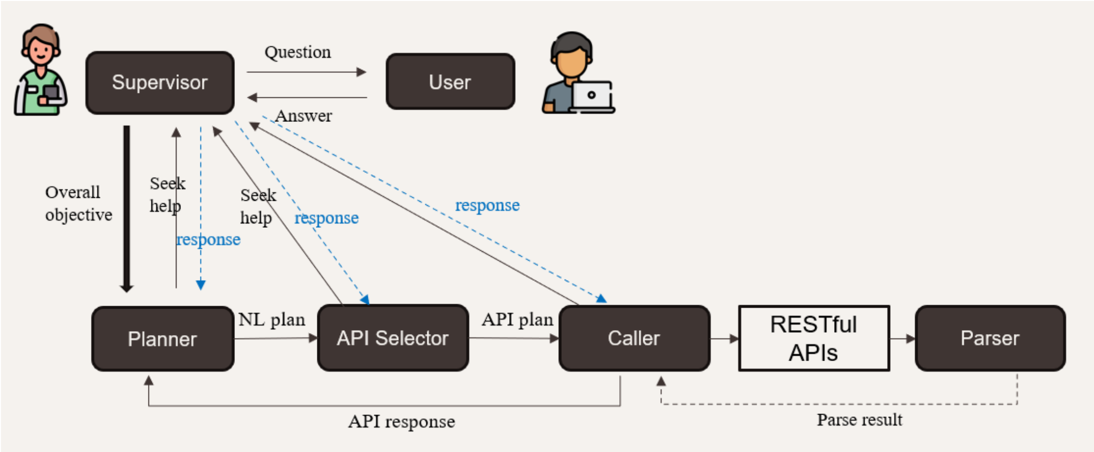

Figure 5.1: The structure of the QwN-augmented RestGPT framework

we introduce a Supervisor role to oversee the entire task process and adjust the final goal. QwN-augmented RestGPT, consists of five agents: Supervisor, Planner, API Selector, Caller, and Parser (except for the Supervisor, the other four roles exist in RestGPT's original framework). In practical scenarios, we envision that LLMs enhanced with tools will be capable of selecting suitable APIs from a wide range of real-world toolsets, allowing accurate execution of complex user instructions. To encourage interaction between users and agents, we modify the prompts used by these agents to prompt them to seek clarification when uncertainties arise. A more detailed explanation of each role will be provided in the subsequent discussion. (The Overall structure is shown in Figure 5.1)

5.2.1 Planner

The Planner agent, as its name suggests, is responsible for breaking down complex tasks into smaller, manageable tasks. Its primary focus is to examine the overall goal of the framework and devise specific plans to accomplish that goal. At each step, the Planner generates a natural language plan that can be executed using a single API. In addition to its planning responsibilities, the Planner may also seek assistance from users when it encounters uncertainties about their requirements, rather than providing a plan directly. The uncertainties that arise during the interaction between users and agents can stem from various reasons.

These reasons include incomplete information in user queries, situations where multiple results refer to different entities with the same name, and other similar factors. These uncertainties highlight the challenges of accurately understanding and addressing user queries, and emphasize the need for effective communication and clarification between the agents and users.

Because LLMs have a tendency to make decisions that may not always align with the user's intentions, particularly when faced with ambiguous queries, we prompt the Planner to generate a thought to carefully consider their current situation before making plans. This prompt encourages the Planner to engage in reflective thinking.

Unlike RestGPT, which adheres strictly to the user's initial input query as the ultimate goal of the framework, we acknowledge that this approach can be problematic in cases where the instructions provided are unclear. To overcome this challenge, our Supervisor adjusts the ultimate goal iteratively to refine it and fulfill the user's requirements at each cycle.

Furthermore, after seeking clarification from the users, the Planner receives additional information that provides guidance for future planning. This information is appended to the execution history and serves as a valuable resource for assisting future planning tasks.

We can define the aforementioned process as follows (See Figure 5.2):

$$Output = Planner(g; p_1, r_1, \ldots, p_{t-1}, r_{t-1}, [a])$$

Here, 'g' represents the ultimate goal of the task. Initially, 'g' corresponds to the user's query, but in subsequent iterations, it may be refined by a Supervisor. The term p denotes the plan at each timestamp. The term 'r' denotes API response for each plan. The variable 'a' is supplementary information utilized for

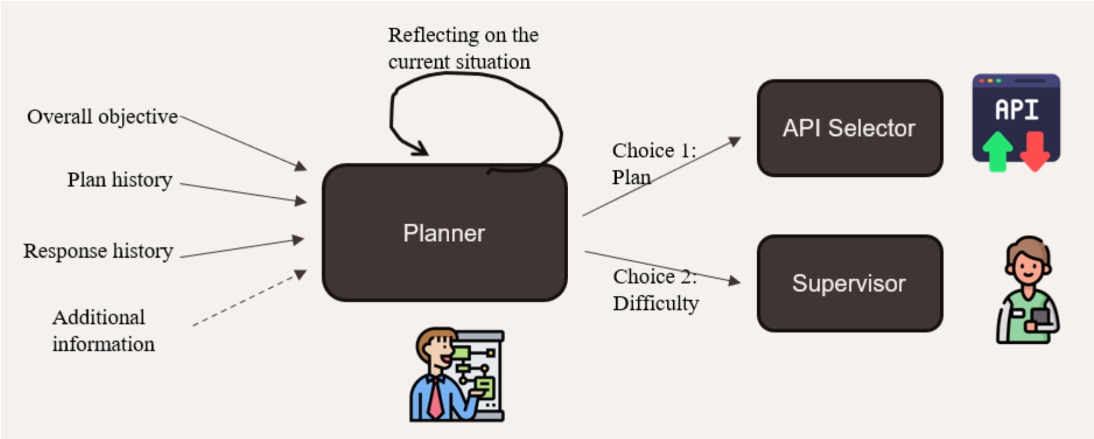

Figure 5.2: Planner

planning purposes, which is provided only when there is a necessity for clarification. The output of this process is either a thought with the plan 'p_t' at the current timestamp or a thought with an elucidation of any challenges encountered.

The following is the prompt for the Planner:

Prompt for the Planner

You are part of a project that uses RESTful APIs to handle user queries. Your role is that of a Planner. You should never use your own knowledge for the task. All the information should be retrieved by using APIs.

When a user's query is complex and requires multiple actions to be resolved, it's your job to break it down into smaller subtasks. Another model will take your plan and identify the suitable API calls to provide the API responses.
You should always give your plan in natural language.
Another model will receive your plan and find the right API calls and give you the result in natural language.
If you assess that the current plan has not been fulfilled, you can output "Continue" to let the API selector select another API to fulfill the plan.
If you think you have got the final answer or the user query has been fulfilled, just output the answer immediately. If the query has not been fulfilled, you should continue to output your plan. In most cases, search, filter, and sort should be completed in a single step.
The plan should be as specific as possible. It is better not to use pronouns in plan, but to use the corresponding results obtained previously. If you want to iteratively query something about items in a list, then the list and the elements in the list should also appear in your plan.
The plan should be straightforward. If you want to search, sort or filter, you can put the condition in your plan.
When providing a plan to guide subsequent models in solving the user's query, you should start the output with "Plan step X:", where X refers to the step number.
Before giving the plan, you should give a thought to analyze the user queries or API response. If there are multiple retrieved results, think carefully which one is referred by the user. If you don't know, then ask the user for clarification.

You might retrieve multiple similar searching items(eg: person or movies with same name, etc) when making an API call, making it unclear which one the user is referring to. For instance, when searching for the movie "Twilight," you may receive various versions with different ids. In these cases, you need to seek clarification from the user and ask the user to narrow down the scope to locate the desired one.

Starting below, you should follow this format:

User query: the query a User wants help with related to the API. Plan step 1: the first step of your plan for how to solve the query
API response: the result of executing the first step of your plan, including the specific API call made.
Thought(if additional information is not provided): the thought to analyze the API response
Plan step 2: based on the API response, the second step of your plan for how to solve the query. If the last step result is not what you want, you can output "Continue" to let the API selector select another API to fulfill the plan. For example, the last plan is "add a song (id xxx) in my playlist", but the last step API response is calling "GET /me/playlists" and getting the id of my playlist, then you should output "Continue" to let the API selector select another API to add the song to my playlist. Pay attention to the specific API called in the last step API response. If an improper API is called, then the response may be wrong and you should give a new plan.
API response: the result of executing the second step of your plan ... (this Plan step n and API response can repeat N times)
Begin!

Ultimate Goal: {input}
Plan step 1: {agent scratchpad}
Additional Information: {user's clarification}

5.2.2 API Selector

The API selector receives the natural language plan generated by the Planner and subsequently sends the API plan to the Caller. The API Selector's primary responsibility is to choose the most appropriate API endpoint from a toolset that comprises numerous RESTful APIs. The selection process is based on the descriptions provided for each API. We also empower the API Selector to seek clarification when encountering difficulties that it cannot handle alone. For instance, if there are no available tools for resolving the user's input query, the API Selector can request clarification from the user. The instructions provided by the users are appended to the history of the API Selector, enabling it to address uncertainties and improve its future decision-making. To encourage API selector to make the correct decision, we also require the API selector to give the reasons behind its every decision.

We can define the aforementioned process as follows (See Figure 5.3):

$$Output = ApiSelector(p_t; r_1, \ldots, r_{t-1}, \sum api\_desc, [a])$$

Here, 'p_t' represents the plan given by the planner. The term 'r' denotes API response for previous plans (Some endpoints require the information of other API responses). '$\sum api\_desc$' represents the descriptions of all available APIs. The variable 'a' is supplementary information utilized for clarification purposes. The output of this process is either a thought with the API plan 'ap_t' at the current timestamp or a thought with an elucidation of any challenges encountered.

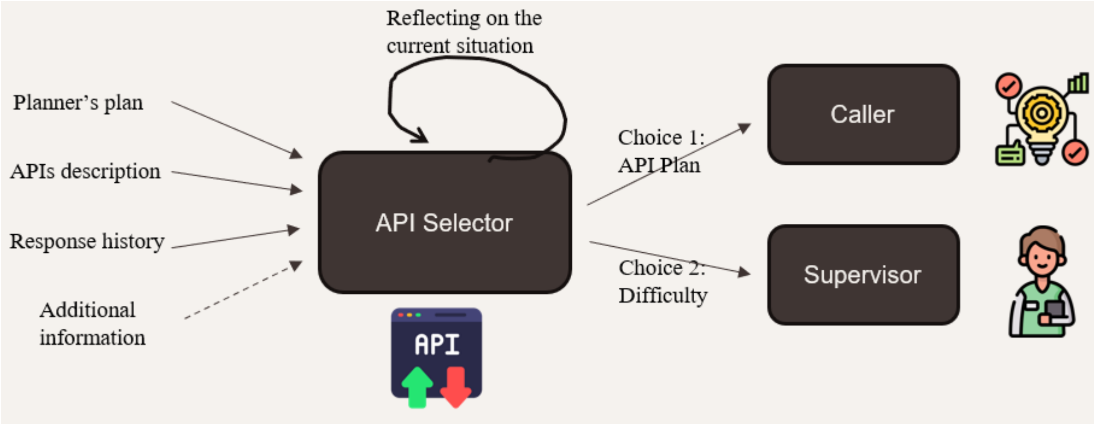

Figure 5.3: API Selector

The following is the prompt for the API Selector:

Prompt for the API Selector

You are a planner that plans a sequence of RESTful API calls to assist with user queries against an API.

Another API caller will receive your plan call the corresponding APIs and finally give you the result in natural language.

The API caller also has filtering, sorting functions to post-process the response of APIs. Therefore, if you think the API response should be post-processed, just tell the API caller to do so.

If you think you have got the final answer, do not make other API calls and just output the answer immediately. For example, the query is to search for a person, you should just return the id and name of the person.

Before giving your plan, you should give a thought to analyze why you select the current API and make sure you can use the API correctly.

If you think there is not available APIs to solve the task, you should analyze the reasons and then give the output "Final result: I cannot solve this task due to the liQwN-augmented RestGPTed of available APIs"

You can only give one step plan(select one API) at each time.

—-

Here are names and descriptions of available APIs.

Do not use APIs that are not listed here (Don't make up any APIs by yourself, the following are the only APIs you can use). endpoints

—-

Starting below, you should follow this format:

Background: background information which you can use to execute the plan,

User query: the query a User wants help with related to the API

API calling 1: Thought: Your thought to analyze the user query and the reasons behind API selecting

Plan:the first api call you want to make. Note the API calling can contain conditions such as filtering, sorting, etc. For example, "GET /movie/18329/credits to get the director of the movie Happy Together"If user query contains some filter condition, such as the latest, the most popular, the highest rated, then the API calling plan should also contain the filter condition. If you think there is no need to call an API, output "No API call needed." and then output the final answer according to the user query and background information.

API response: the response of API calling 1
Instruction: Another model will evaluate whether the user query has been fulfilled. If the instruction contains "continue", then you should make another API call following this instruction.
... (this API calling n and API response can repeat N times, but most queries can be solved in 1-2 step)

Always think comprehensively and seek clarification from users whenever the user's query is not clear enough for you to formulate API calling. If you require additional information or face uncertainty, start the conversation with "I need user's clarification" and then express the challenges explicitly. Another thing to be noted is that do not ask users for information related to id.

Begin!

Background: {background}
User query: {plan}
API calling 1: {agent scratchpad}
Additional information: {user's clarification}

5.2.3 Caller

Once an API calling plan is created, the subsequent step involves its execution. The Caller plays a crucial role in this process by thoroughly examining the API documents and generating the appropriate parameters or request body for the API call. With the generated parameters and request body in place, we utilize the Requests Python library to effectively invoke the RESTful API.

In situations where the Caller is unable to find suitable arguments from the API plan, there is a possibility that the user did not provide sufficient information in the initial query. This can lead to a lack of necessary information in the API

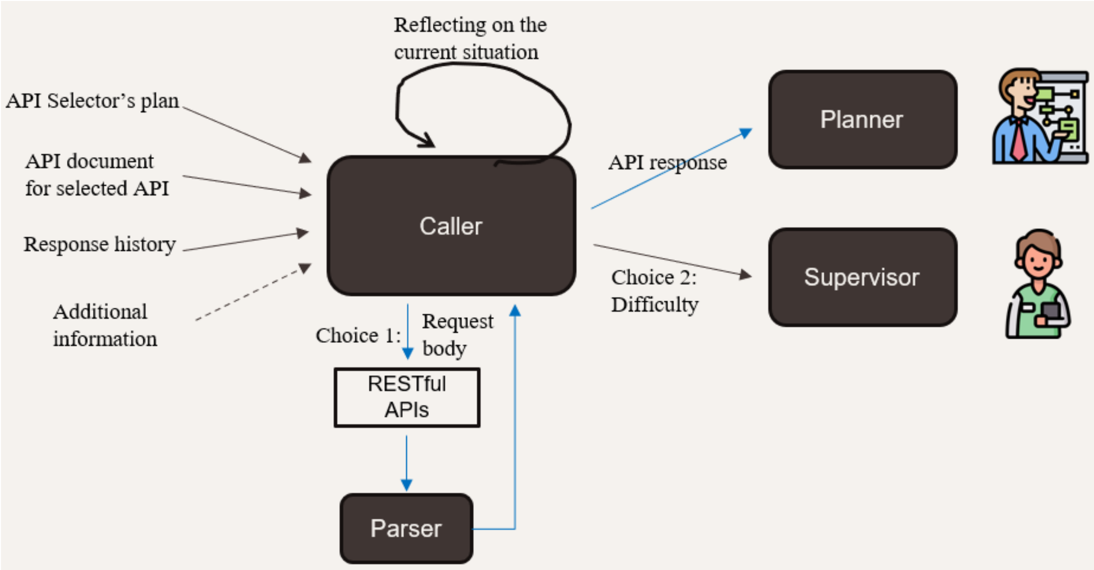

Figure 5.4: Caller

plan generated by the API Selector, consequently impacting the Caller's functionality. To address these potential issues, we prompt the Caller to seek clarifications instead of improvising arguments when uncertainties arise. This encourages the Caller to actively communicate with the user to obtain the necessary information and ensure the accuracy of the API call.

We can define the aforementioned process as follows(See Figure 5.4):

$$Output = Caller(ap_t; r_1, \ldots, r_{t-1}, api\_doc, [a])$$

Here, 'ap_t' represents the api plan given by the API Selector. The term 'r' denotes API response for previous plans (Some tools may require the information of other API responses). 'api_doc' represents the API document for the selected API. The variable 'a' is supplementary information utilized for clarification purposes. The output of this process is either a thought with the API request body or a thought with an elucidation of any challenges encountered.

The following is the prompt for the Caller:

Prompt for the Caller

You are an agent that gets a sequence of API calls and given their documentation, should execute them and return the final response.

If you cannot complete them and run into issues, you should explain the issue. If you're able to resolve an API call, you can retry the API call. When interacting with API objects, you should extract ids for inputs to other API calls but ids and names for outputs returned to the User.

Your task is to complete the corresponding api calls according to the plan.

Here is documentation on the API:

Base url: api url

Endpoints:

api docs

You can use the http request method, i.e., GET, POST, DELETE, PATCH, PUT, and generate the corresponding parameters according to the API documentation and the plan.

The input should be a JSON string which has 3 base keys: url, description, output instructions

The value of "url" should be a string.

The value of "description" should describe what the API response is about. The description should be specific.

The value of "output instructions" should be instructions on what information to extract from the response, for example the id(s) for a resource(s) that the POST request creates. Note "output instructions" MUST be natural language and as verbose as possible!

If you are using the GET method, add the "params" key, and the value of "params" should be a dict of key-value pairs.
If you are using POST, PATCH or PUT methods, add "data" key, and the value of "data" should be a dict of key-value pairs.
Remember to add a comma after every value except the last one, ensuring that the overall structure of the JSON remains valid.
I will give you the background information and the plan you should execute.
Background: background information which you can use to execute the plan, e.g., the id of a person.
Plan: the plan of API calls to execute

You should execute the plan faithfully and give the Final Answer as soon as you successfully call the planned APIs, don't get clever and make up steps that don't exist in the plan. Do not make up APIs that don't exist in the plan. For example, if the plan is "GET /search/person to search for the director "Lee Chang dong", do not call "GET /person/person_id/movie_credits" to get the credit of the person.

However, there may be instances when users submit queries with potential issues, like missing information or typos, making it impossible to fulfill the query without further clarification. You might receive multiple outcomes when making an API call, making it unclear which one the user is referring to. For instance, when searching for the movie "Twilight" you may receive various versions. In these cases, you need to seek clarification from users and explain the issue clearly. Another model will then read about the difficulties you encountered and generate questions to request additional information from users. Anytime you need the clarification of a user's query, your output should start with "I need user's clarification.". Another thing to be noted is that do not ask users for information related to id.

Another important point: Never make up information by yourself. All the information should be retrieved by using APIs.

Starting below, you must follow this format:

Background: background information which you can use to execute the plan, e.g., the id of a person.

Plan: the plan of API calls to execute

Thought: you should always think about what to do

Operation: the request method to take, should be one of the following: GET, POST, DELETE, PATCH, PUT

Input: the input to the operation

Response: the output of the operation

Thought: I am finished executing the plan (or, I cannot finish executing the plan without knowing some other information.)

Execution Result: based on the API response, the execution result of the API calling plan.

Begin!

Background: {background}

Plan: {api_plan}

Thought: {agent_scratchpad}

Additional information: {user's clarification}

5.2.4 Parser

The Parser agent assumes the responsibility of extracting the relevant information from the API response, considering that the response may contain a significant amount of irrelevant data. We directly incorporate the Parser from RestGPT, as its functionality aligns well with our requirements. Therefore, readers can directly refer to RestGPT for details of the prompt. By utilizing the Parser, we can effectively filter and extract the desired information from the API response, ensuring that only the relevant data is presented to the user.

5.2.5 Supervisor

The Supervisor agent plays a crucial role in providing assistance to other agents, guiding the overall objective of each cycle and terminating the task when the user's goal is achieved.

When any of the other agents encounter uncertain situations, they seek help from the Supervisor and explain the difficulties they are facing. The Supervisor then analyzes these difficulties based on its comprehensive understanding of the entire task. Depending on the situation, the Supervisor can either decide to assist the agent directly based on its understanding of the tasks, or generate a question to query users for additional information.

In the first case, where the user's instructions are not unclear but some of the preceding agents have generated incorrect output, leading to propagation of false information to subsequent agents, the Supervisor needs to analyze the problem and provide appropriate guidance to the agent seeking help. This ensures that the agent can overcome the hindrance caused by the false output.

In the second case, where the user has not provided clear enough instructions resulting in execution difficulties, the Supervisor generates a question based on the difficulties expressed by the agent to query users for additional information. After obtaining the response from the user. The question and answer are then sent to an integration module to formulate a complete sentence that serves as additional information to clarify the agent's confusion.

Indeed, the interaction between the agents and users can lead to adjustments in the final goal of the task. For instance, consider a scenario where the user initially inputs the query: "Give me the director of the movie 'Pulp Faction'" (with a misspelled movie name, where the correct name should be "Pulp Fiction"). Through interactions between the Caller and the user, the user clarifies their desired movie as "Pulp Fiction." In such cases, the Supervisor plays a crucial role in adjusting the final goal given to the Planner. The adjusted goal would be "Give me the director of the movie 'Pulp Fiction'" in the subsequent round to avoid any confusion. The Supervisor takes charge of making these adjustments to ensure that the agents are aligned with the user's intentions and can effectively fulfill the user's request.

Finally, the Supervisor agent is also responsible for terminating the task when the user's query has been successfully addressed. Once the user's query has been achieved, the Supervisor formulates the answer derived from the user's query and delivers it back to the user.

We can define the aforementioned process as follows (See Figure 5.5):

$$\mathit{Output} = \text{Supervisor}\left(\sum_{i=1}^{t} p_i + \sum_{i=1}^{t} ap_i + \sum_{i=1}^{t} r_i + \sum u\right)$$

The meanings of "p", "ap", and "r" remain unchanged, while "u" indicates the input provided by the user. Depending on the specific input, the output of this procedure can be the ultimate goal, questions generated to ask users, or direct support to the agent.

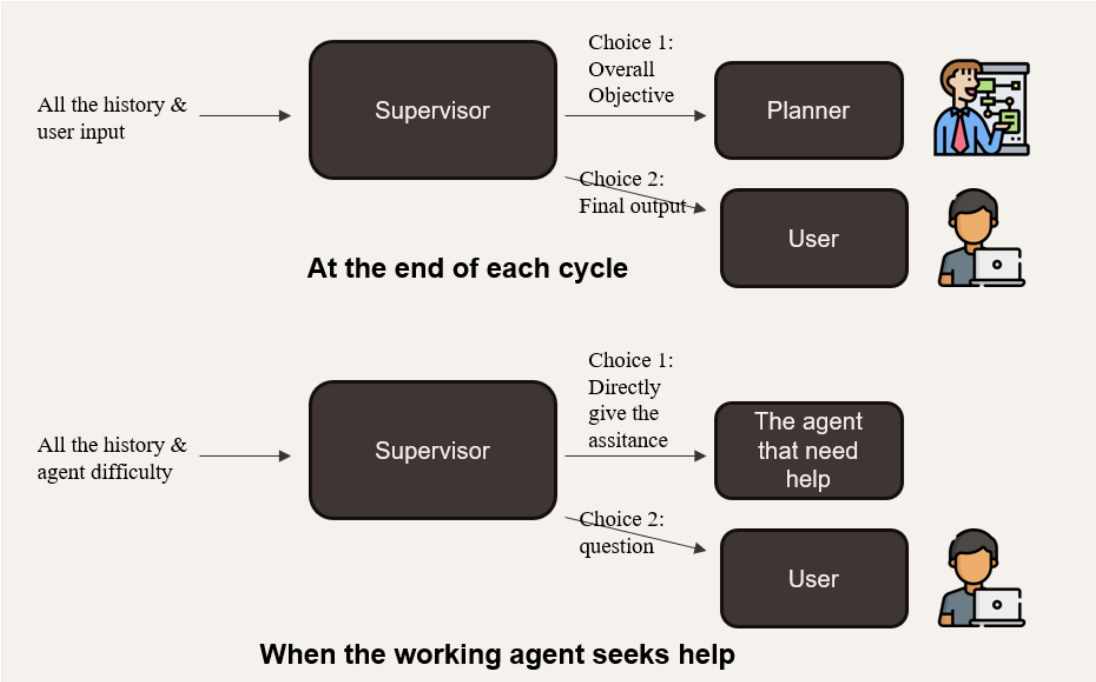

Figure 5.5: Supervisor

The following is the prompt for the Supervisor:

Prompt for the Supervisor

You are in charge of a team of agents responsible for making RESTful API calls to address user queries. Your team is composed of three roles: Planner, API Selector, and Caller, each with specific responsibilities. Your ultimate objective is to successfully address user queries. As the team leader, you oversee the entire working process of your team. During this process, you'll encounter two key situations:

1. Situation 1 (Agents seeking clarification): Be prepared for the possibility that the agents under your supervision may encounter difficulties and require clarification or have questions for users. You act as the intermediary between users and the various agents for communication. Other agents are not perQwN-augmented RestGPTted to directly communicate with users without your assistance. If agents have questions, they should preface their queries with "I need user's clarification." Your role is to assess the issues and explanations provided by the agents. If you agree with them, you must formulate a question to ask the user for additional information. When formulating a question for the user, you should begin your response with "Question:". Conversely, if you believe the problem is not the user's fault, you can challenge the agent's question and respond independently, beginning with "Refutation."
2. Situation 2 (End of one working cycle): Typically, addressing user queries involves multiple iterative steps. At the end of each iteration, it is essential to clarify the overall objective for your team. At the conclusion of each cycle, you will encounter a marker labeled "End of Cycle" If the user's query has been effectively resolved after a cycle, you can conclude the process and present the query's outcome, starting with "Final Answer:" On the other hand, if the user's goal remains unmet, you should convey the user's expectations to the Planner for additional planning, initiating your response with "Objective to Planner:"

You cannot make up information for other roles. You should never use your own knowledge for the task. All the information should be retrieved by using APIs.
Begin!
chat history is shown below: {agent scratchpad}

Chapter 6

Evaluation

6.1 Experiment result

Case	**RestGPT**	**QwN-augmented RestGPT**
Complete instruction	0.66 (50 cases)	0.76 (50 cases)
Unclear or missing information	0 (70 cases)	0.61 (70 cases)
Problematic searching items	0.5 (30 cases)	0.93 (30 cases)
Instruction that cannot support	0 (13 cases)	0.31 (13 cases)
Instruction with different meanings	0	-

Table 6.1: Experiment result of RestGPT and QwN-augmented RestGPT

In complete instructions, RestGPT performs reasonably at a score of 0.66 over 50 cases while QwN-augmented RestGPT performs better with a score of 0.76 over the same number of cases. In unclear or missing Information instructions, RestGPT cannot handle these types of instructions, scoring zero over 70 cases while QwN-augmented RestGPT performs significantly better, scoring 0.61 over 70 cases. In problematic searching item instructions, RestGPT has a moderate performance, scoring 0.5 over 30 cases while QwN-augmented RestGPT excels in this category, scoring a high 0.93 over 30 cases. In different meaning instructions: RestGPT cannot handle these types of instructions, scoring zero over 13 cases. QwN-augmented RestGPT, while also struggling in this area, performs better than RestGPT with a score of 0.31 over 13 cases. In unsupported instructions: RestGPT cannot handle these types of instructions, scoring zero. The evaluation of QwN-augmented RestGPT's performance in this category is still in progress. QwN-augmented RestGPT outperforms RestGPT in handling complete instructions, unclear or missing information instructions, problematic searching item instructions, and instructions with different meanings.

6.2 Research Questions

To evaluate QwN-augmented RestGPT, we study three research questions in this report:

RQ1: Explain that QwN-augmented RestGPT can have a better performance compared to the original RestGPT framework.

RQ2: An examination of whether QwN-augmented RestGPT can accurately identify the incorrect part in the user query.

RQ3: Examination of whether QwN-augmented RestGPT can improve the robustness of facing incomplete queries.

6.3 Result and Analysis

6.3.1 RQ1: Explain that QwN-augmented RestGPT can have a better performance compared to the original RestGPT framework

To demonstrate that QwN-augmented RestGPT can perform better compared to the original RestGPT framework, we present the experimental results of the original dataset using both QwN-augmented RestGPT and RestGPT. Additionally, we will focus on the impact of the "Supervisor" and "Think and Plan" modules in QwN-augmented RestGPT.

1. Supervisor

RestGPT

Query: I just finished watching Titanic and I want some other movie recommendations

Planner: Continue. Get the list of similar movies to Titanic (597)

API Selector: GET /movie/597/similar to get the list of similar movies to Titanic (597)

QwN-augmented RestGPT

Query: I just finished watching Titanic and I want some other movie recommendations

Objective: Please use the id of the movie "Titanic" (597) to get movie recommendations.

Planner: Get the movie recommendations of the movie "Titanic" (597).

API Selector: GET /movie/597/recommendations to get the movie recommendations of the movie Titanic (id 597).

From the above example, the Planner in RestGPT may select incorrect APIs. This error analysis highlights the insufficient capabilities of RestGPT without a 'supervisor'.

2. "Think and Plan"

RestGPT

Query: What does the lead actor of Titanic look like?

Planner: search for the lead actor of Titanic

API Selector: GET /movie/157336/credits to get the lead actor of Titanic

QwN-augmented RestGPT

Query: What does the lead actor of Titanic look like?

Planner: Search for the lead actor of Titanic.

API Selector: Thought a: I need to get the movie id of Titanic to search for the lead actor.

Plan: GET /search/movie to get the movie id of Titanic.

From the above example, the API selector in RestGPT may hallucinate to make up in-path parameters. This error analysis highlights the insufficient capabilities of RestGPT without the ability to 'think and replan'.

6.3.2 RQ2: An examination of whether QwN-augmented RestGPT can accurately identify the incorrect part in the user query

To examine whether QwN-augmented RestGPT can accurately identify the incorrect part in the user query, we present a table that shows which module in QwN-augmented RestGPT raised the question. We label the questions as either 'relevant' or 'irrelevant' Also, we provide examples of 'irrelevant' questions.

Case	**Planner**	**Caller**	**Total**
Unclear instruction & instruction with missing information			
Relevant	39	16	55
Irrelevant	6	9	15
Instruction with problematic searching items			
Relevant	11	6	17
Irrelevant	2	1	3
Instruction with different meanings			
Relevant	1	2	3
Irrelevant	0	2	2
Total	59	35	95
Relevant: 75 (0.789), Irrelevant: 20 (0.211)			

Table 6.2: Number of relevant and irrelevant questions raised up by our model

For 'Unclear instruction and instruction with missing information', 55 were relevant and 15 were irrelevant. For 'Instruction with problematic searching items', 17 were relevant and 3 were irrelevant. In the case of 'Instruction with different meanings', 3 were relevant and 2 were irrelevant. In total, the planner provided 59 instructions, the caller provided 35, with an overall total of 95 instructions. Evaluating relevance, 75 instructions (78.9% of total instructions) were relevant, and 20 instructions (21.1% of total instructions)were irrelevant.

Irrelevant follow-up question

Planner: Search for Breaking Bad (id1) using GET /search/tv.

API Selector: Thought: I have the id of Breaking Bad, I can use it to get the details of the show.

Plan: GET /tv/id1 to get the details of Breaking Bad.

Caller: I need user's clarification. The API response indicates an error: "The resource you requested could not be found." It seems that the id of Breaking Bad is not valid. Please provide the valid id of Breaking Bad.

Question: Could you provide the valid id of Breaking Bad?

6.3.3 RQ3: Examination of whether QwN-augmented RestGPT can improve the robustness of facing incomplete query

Instructions with missing information

Instruction of TMDB

Query: I need something like that movie.

QwN-augmented RestGPT

Planner: I need user's clarification! The user did not provide enough information to proceed. Without a specific movie title, I cannot search for similar movies.

Question: Could you provide the title of the movie you are looking for?

User: Titanic

Additional information: The movie the user is looking for is Titanic.

Planner: Search for a movie with the title "Titanic".

API Selector: Thought: I need to search for a movie with the title "Titanic", so I will use the GET /search/movie API to search for the movie.

Plan: GET /search/movie to search for the movie with the title "Titanic".

Caller: I need to generate the input for the GET /search/movie API call. Operation: GET

...

...

Execution Result: The similar movies and their ids are Adieu Les Copains (1096746), Just Ask My Children (237983)

Problematic Searching Item Instructions

Instruction of TMDB

Query: What is the logo of the Watt Daisney?

QwN-augmented RestGPT

Planner: Search for the company "Watt Daisney".

API Selector: Thought: I need to search for the company "Watt Daisney" to get its company id.

Plan: GET /search/company to search for the company "Watt Daisney".

Caller: I need to use the GET method to make the API call.

...

...

Code:

Extract the id of the company Watt Daisney company_id = [item['id'] for item in data['results'] if item['name'] == 'Watt Daisney'][0]

Print the result

print('The id of the company Watt Daisney is {}'.format(company_id))

Code: print("The id of the company Watt Daisney is {}".format(data['results'][0]['id']))

Parser: No results found for the query 'Watt Daisney'.

Caller: Execution Result: I need user's clarification. The API response did not return any relevant information when I call GET /search/company to search for the company "Watt Daisney".

Question: Could you please double-check if the company name is correct?

User: It should be Walt Disney

...

...

(Finally success)

Different Meaning Instructions

Instruction of TMDB

Query: Find films with Jennifer

QwN-augmented RestGPT

Planner: Search for a person with the name "Jennifer".

API Selector: Thought: After checking all the available APIs, I need to use the search API to search for the person with the name "Jennifer".

Plan: GET /search/person to search for the person with the name "Jennifer".

Caller: I need to generate the input for the GET request according to the API documentation.

Operation: GET

...

...

Caller: Execution Result: Successfully call GET /search/person to search for the person with the name "Jennifer". The ids and names of the people returned by the API are (578325, 'Jennifer'), (3740055, 'Jennifer'), (4354994, 'Jennifer'), (6161, 'Jennifer Connelly'),......

Planner: Thought: There are multiple people with the name "Jennifer" returned by the API. I need user's clarification! Please specify which Jennifer you are referring to.

Question: Could you please specify which Jennifer you are referring to?

User: Jennifer Lawrence

Additional information: The user is referring to Jennifer Lawrence.

Planner: Thought: There is ambiguity in the API response. Search for films with Jennifer Lawrence (72129).

...

...

(Finally success)

For the failure case of Itool on RestGPT please refer to AppendixA.

Chapter 7

Conclusion

7.1 Summary

Motivated by the limitations observed in current frameworks when handling ambiguous instructions, we have developed a novel approach called the Query when Need (QwN) framework. QwN enables agents to request assistance from users when confronted with unclear instructions. To assess the efficacy of our framework, we have integrated QwN with ToolLLM (which is still a work in progress) and RestGPT. We conducted experiments using a carefully curated dataset called Itool, comprising four common types of unclear instructions frequently encountered from users. The results obtained from QwN-augmented RestGPT demonstrate that our approach outperforms existing frameworks in effectively resolving queries related to ambiguous instructions.

7.2 Future directions

Despite providing agents with the capability to seek assistance when faced with challenges, determining the appropriate timing for requesting help remains a difficult task. Agents often exhibit a tendency to generate false information or make decisions without informing users. Addressing the challenges of reasoning and learning when to ask for help are important tasks that require future solutions.

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

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

[4] OpenAI, "Gpt-4 technical report," *ArXiv*, vol. abs/2303.08774, 2023.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] L. Li, W. Chou, W. Zhou, and M. Luo, "Design patterns and extensibility of rest api for networking applications," *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 154–167, 2016.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] S. Hong, X. Zheng, J. Chen, Y. Cheng, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran *et al.*, "Metagpt: Meta programming for multi-agent collaborative framework," *arXiv preprint arXiv:2308.00352*, 2023.

[21] Z. Wang, S. Cai, A. Liu, X. Ma, and Y. Liang, "Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents," *arXiv preprint arXiv:2302.01560*, 2023.

[22] A. Smit, P. Duckworth, N. Grinsztajn, T. Barrett, A. Pretorius *et al.*, "Are we going mad? benchmarking multi-agent debate between language models for medical q&a," in *Deep Generative Models for Health Workshop NeurIPS 2023*, 2023.

[23] 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 lan-

guage models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[24] 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.

[25] N. Shinn, B. Labash, and A. Gopinath, "Reflexion: an autonomous agent with dynamic memory and self-reflection," *arXiv preprint arXiv:2303.11366*, 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] J. J. Wu, "Does asking clarifying questions increases confidence in generated code? on the communication skills of large language models," 2023.

[28] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, pp. 29–43.

A Failure Case on RestGPT

Instructions with Missing Information

Instruction of TMDB

Query: give me the number of movies directed by my father

RestGPT

Planner: search for the person with name "my father"

API Selector: GET /search/person to search for the person with name "my father"

Caller: I need to make a GET request to the /search/person endpoint with the query parameter set to "my father".

Operation: GET Input: {

"url": "https://api.themoviedb.org/3/search/person",

"query": "my father"

Parser: No results found

Caller: Thought: I cannot finish executing the plan without knowing some other information.

Execution Result: Failed to call GET /search/person to search for the person with name "my father". No results found.

Planner: Continue. search for the person with name "father"

Caller: I should try to search for the person with name "father" instead of "my father".

Operation: GET Input: {

"url": "https://api.themoviedb.org/3/search/person",

"params": {

"query": "father",

(Finally fail)

Problematic Searching Item Instructions

Instruction of TMDB

Query: tell me a TV show recently directed by Katherine Hardwood

RestGPT

Planner: search for the director ”Katherine Hardwood”

API Selector: GET /search/person to search for Katherine Hardwood

Caller: I need to make a GET request to the /search/person endpoint with the query parameter set to Katherine Hardwood.

Operation: GET Input: {

”url”: ”https://api.themoviedb.org/3/search/person”,

”params”: {

”query”: ”Katherine Hardwood”

Parser: No results found for Katherine Hardwood.

Caller: Thought: I cannot finish executing the plan without knowing some other information.

Execution Result: No results found for Katherine Hardwood.

Planner: search for the director ”Kathryn Bigelow”

...

(Finally fail)

Instruction with Different Meanings

Instruction of TMDB

Query: Find films with Jennifer

RestGPT

Planner: Search for person with name ”Jennifer”

API Selector: GET /search/person to search for person with name ”Jennifer”

Caller: I need to make a GET request to the /search/person endpoint with the query parameter set to ”Jennifer”.

Operation: GET Input: {

”url”: ”https://api.themoviedb.org/3/search/person”,

”params”:

”query”: ”Jennifer”

Code: print(”The id of the first person in the list is ”.format(data['results'][0]['id']))

Parser: The id of the first person in the list is 578325

Caller: Execution Result: Successfully call GET /search/person to search for person with name ”Jennifer”. The id of the first person in the list is 578325.

Planner: Collect the list of movies performed by Jennifer whose person_id is 578325

...

...

(Finally fail)

Unsupported Instructions

Instruction of TMDB

Query: Can you provide the Twitter handle of Natalie Portman?

RestGPT

Planner: search for the person with name ”Natalie Portman”

API Selector: GET /search/person to search for Natalie Portman

Caller: I need to make a GET request to the /search/person endpoint with the query parameter set to ”Natalie Portman”.

...

...

Operation: GET Input:

”url”: ”https://api.themoviedb.org/3/person/524”,

”description”: ”Get the primary person details by id.”,

”output_instructions”: ”What is the Twitter handle of Natalie Portman (524)?”

Code:

print(”The Twitter handle of Natalie Portman (524) is ”.format(data[’also_known_as’][-1]))

Parser: The Twitter handle of Natalie Portman (524) is

(Finally fail)