TPTU: Task Planning and Tool Usage of Large Language Model-based AI Agents

Jingqing Ruan[†‡]
ruanjingqing@sensetime.com

Yihong Chen[†‡]
chenyihong@sensetime.com

Bin Zhang[†‡]
zhangbin11@sensetime.com

Zhiwei Xu[†‡]
xuzhiwei@sensetime.com

Tianpeng Bao[†]
baotianpeng@sensetime.com

Guoqing Du[†]
duguoqing@sensetime.com

Shiwei Shi[†]
shishiwei@sensetime.com

Hangyu Mao[†*]
maohangyu@sensetime.com

Ziyue Li [+]
zlibn@connect.ust.hk

Xingyu Zeng
zengxingyu@sensetime.com

Rui Zhao
zhaorui@sensetime.com

SenseTime Research

Abstract

With recent advancements in natural language processing, Large Language Models (LLMs) have emerged as powerful tools for various real-world applications. Despite their powers, the intrinsic generative abilities of LLMs may prove insufficient for handling complex tasks, which necessitate a combination of task planning and the usage of external tools. In this paper, we first propose a structured framework tailored for LLM-based AI Agents and then discuss the crucial capabilities necessary for tackling intricate problems. Within this framework, we design two distinct types of agents (i.e., one-step agent and sequential agent) to execute the inference process. Subsequently, we instantiate the framework using various LLMs and evaluate their Task Planning and Tool Usage (TPTU) abilities on typical tasks. By highlighting key findings and challenges, our goal is to provide a helpful resource for researchers and practitioners to leverage the power of LLMs in their AI applications. Our study emphasizes the substantial potential of these models while also identifying areas that need more investigation and improvement. The code and resources will be available on GitHub.

1 Introduction

Large Language Model (LLM) [1] is a recent breakthrough in natural language processing (NLP) research. These models are trained on massive amounts of text data and can solve a wide range of tasks, even those that were not included in their training dataset, known as “emerging” ability. This

[†]These authors contribute equally to this work.

[+]External discussion and ideation.

[‡]These authors work as research interns at SenseTime Research.

[*]The corresponding author.

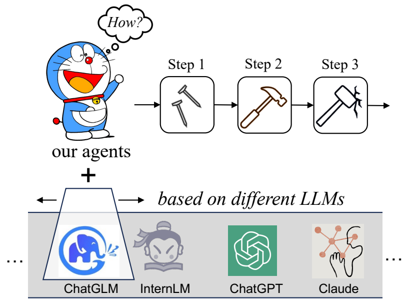


Figure 1: Our LLM-based agents plan tasks and use tools.

ability is especially evident in the tasks of few-shot [2] and zero-shot [3] learning, where LLMs can perform well with minimal or even no fine-tuning to adapt to a new task.

However, the application of LLMs in real-world settings presents unique challenges. On the one hand, LLMs have proved to be incompetent in solving logic problems such as mathematics, and their training data is also out of date (e.g., the knowledge cutoff date for GPT-4 [4] is up to January 2022). Teaching LLMs to use tools such as calculators, calendar, or search engines can help prevent them from hallucinating [5]. On the other hand, despite their impressive problem-solving abilities, the successful integration of these models into complex systems often requires more than just task understanding - it requires the capacity to manipulate various tools and interact effectively with users. This is exemplified in systems like AutoGPT [1], BabyAGI [2], and ChatGPT-plugins [3], which leverage LLMs' capabilities beyond merely generating well-written texts and programs. In these systems, LLMs operate as the central controller, manipulating different tools and interacting with humans, thus taking on the role of Artificial Intelligence Agents (AI Agents). In addition to being central planners, LLMs are often used as intermediaries between macro plans and low-level tool calls or as specific tools. As such, LLMs are seen as a crucial approximation of the linguistic world model in real-world systems.

In this paper, we propose a structured framework for LLM-based AI Agents to evaluate the existing LLMs' planning and tool-using ability and discuss the necessary abilities of such LLM-based AI Agents. Furthermore, we instantiate the framework with different LLMs and evaluate their **T**ask **P**lanning and **T**ool **U**sage (TPTU) abilities on several tasks. As shown in Figure 1, we use the Doraemon as an analogy of our LLM-based agents: Doraemon's magic 4D pocket consists of millions of gadgets (the Tool Set), and Doraemon needs to pick the right tools and solve tasks in a right order. Our main contributions are summarized as follows:

1. We propose a structured framework tailored for LLM-based AI Agents to evaluate the TPTU abilities of the existing open-source LLMs.
2. We design two distinct types of agents, namely, one-step agent and sequential agent, to execute the inference process of conducting sub-tasks in a once-for-all or sequential manner, respectively. We provide detailed empirical results and analysis.
3. Our study reveals significant potential in utilizing LLMs for complex tasks. Furthermore, we conclude four following potential weaknesses of LLM-based agents: failing to output in a specific format, struggling to grasp task requirements, over-utilizing one tool, and lack of summary skills. These observations could spark some insights and shed light on the areas that deserve further investigation and improvement.

2 Method

To the best of our knowledge, the study of "Agent", "Autonomous Agent", "AI Agent" and "Multi-Agent" has been a central part of AI research for decades [6–11], aimed at understanding and building

[1]`https://github.com/Significant-Gravitas/Auto-GPT`
[2]`https://github.com/yoheinakajima/babyagi`
[3]`https://openai.com/blog/chatgpt-plugins`

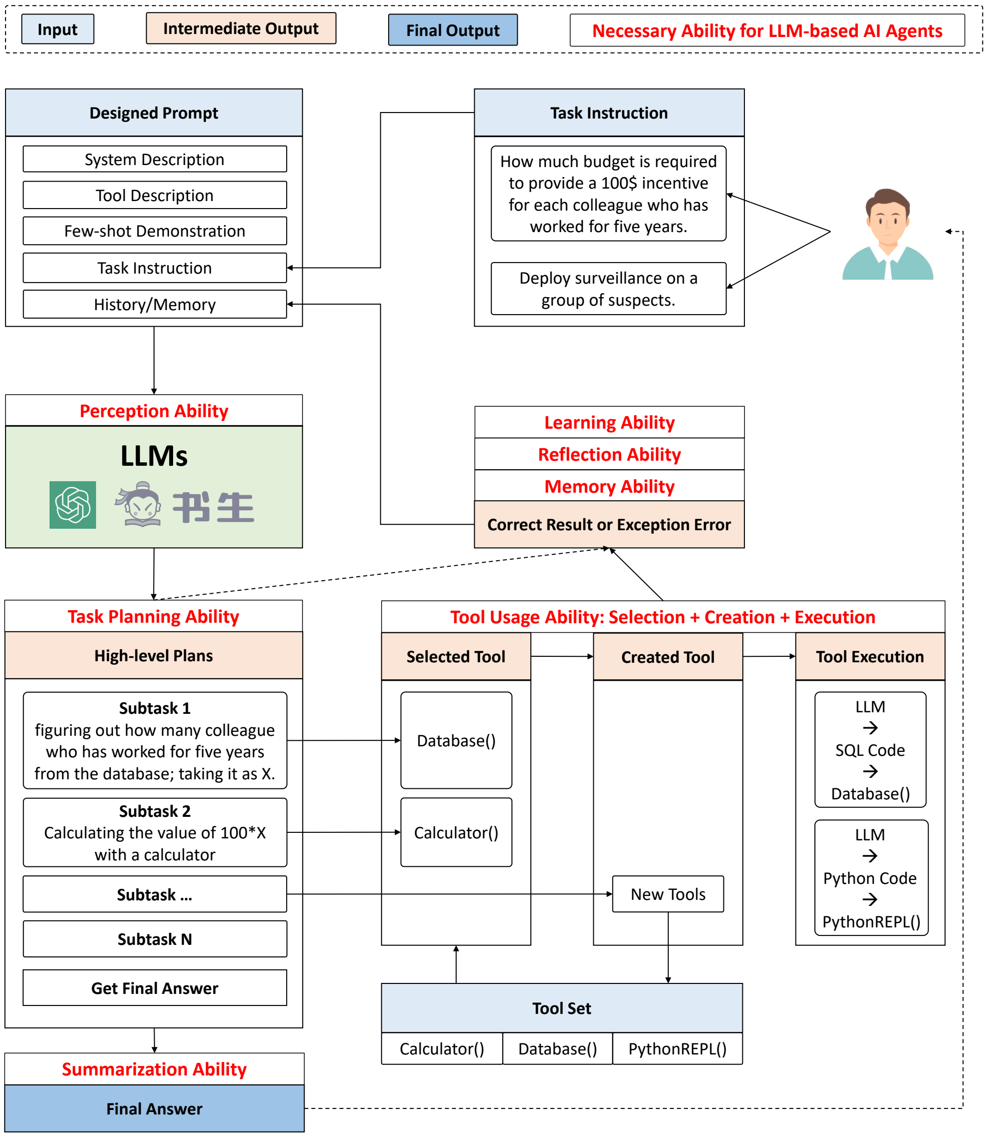

Input
Intermediate Output
Final Output
Necessary Ability for LLM-based AI Agents
Designed Prompt
System Description
Tool Description
Few-shot Demonstration
Task Instruction
History/Memory
Task Instruction
How much budget is required to provide a 100$ incentive for each colleague who has worked for five years.
Deploy surveillance on a group of suspects.
Perception Ability
LLMs
书生
Learning Ability
Reflection Ability
Memory Ability
Correct Result or Exception Error
Task Planning Ability
High-level Plans
Subtask 1
figuring out how many colleague who has worked for five years from the database; taking it as X.
Subtask 2
Calculating the value of 100*X with a calculator
Subtask ...
Subtask N
Get Final Answer
Tool Usage Ability: Selection + Creation + Execution
Selected Tool
Created Tool
Tool Execution
Database()
Calculator()
New Tools
LLM → SQL Code → Database()
LLM → Python Code → PythonREPL()
Tool Set
Calculator()
Database()
PythonREPL()
Summarization Ability
Final Answer

Figure 2: The proposed framework for LLM-based AI Agents.

intelligent and autonomous systems, but there is currently no standardized definition for AI Agents, particularly those that are based on LLMs.

In this paper, the Artificial Intelligence Agent (AI Agent) is defined as a program that employs artificial intelligence techniques to perform tasks that typically require human-like intelligence. AI Agents can take many forms, from simple chatbots to complex autonomous systems that interact with their environment and make decisions in real-time. They can be trained using various machine learning techniques, including supervised, unsupervised, and reinforcement learning. Moreover, AI Agents can also be programmed to perform specific tasks or learn from their experiences to improve their performance over time.

2.1 Agent Framework

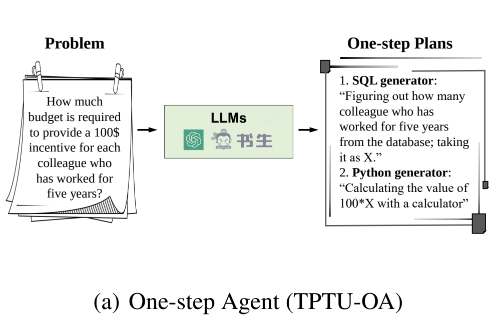
We are particularly interested in the AI Agent that employs the LLM techniques (i.e., LLM-based AI Agent), due to its high efficiency and flexibility in various tasks and domains. Specifically, we design our AI Agent framework with six components as shown in Figure 2:

1. **Task Instruction**. The task instruction is the explicit input of the agent. In practical systems, the task instruction comes from human users of the systems. For example, in a Human Resources (HR) system, the user may give a task instruction: How much budget is required to provide a 100$ incentive for each colleague who has worked for five years?
2. **Designed Prompt**. This is an additional form of input for the agent, derived from tasks that the human users anticipate the AI Agent will complete. Humans can craft specific instructions or demonstrations to steer the LLM-based AI Agents toward generating suitable responses. These guiding inputs could include system instructions, tool descriptions, few-shot demonstrations, chat history, or even error output.
3. **Tool Set**. The tool set refers to the set of external resources, services, or subsystems that the AI Agent can utilize to complete its tasks. It could include databases for information retrieval, APIs for interacting with external systems, other AI models specialized for tasks such as image recognition or sentiment analysis, or even non-AI tools and resources such as web scraping tools or data visualization libraries. The tool set expands the capabilities of the AI Agent, enabling it to access and process information beyond its internal knowledge, interact with other systems, or perform specialized tasks that it may not be capable of on its own. For example, an AI Agent might use a weather API to fetch current weather information, or a Python interpreter to solve the mathematical question.
4. **LLM**. As the system's core component, LLM interprets the task instructions and prompts, interacts with the toolset, and generates intermediate outputs and final answers. In this paper, we utilize publicly available large language models such as ChatGPT, GPT-4, and others.
5. **Intermediate Output**. The intermediate output represents the output generated by the LLM-based AI Agent after it processes the task instructions and prompts, and interacts with the toolset. There are three typical intermediate outputs: (1) the high-level plans to fulfill the original user instruction, (2) selected and created tools to fulfill each subtask in the plans, and (3) the results or errors produced after tool execution. The output can be reviewed and refined, either by the AI Agent itself or with human oversight, to ensure it is accurate and meets the requirements of the task instruction.
6. **Final Answer**. The final answer is the output that the AI Agent provides to the user after all processes (including task planning, tool usage, and error feedback) have been completed.

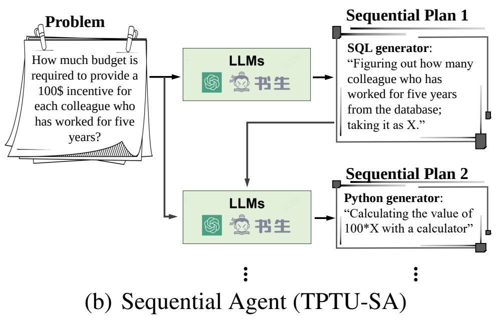
2.2 Agent Ability

For LLM-based AI Agents to assist or replace humans in real-life decision-making tasks, the agents typically require the following abilities:

1. **Perception**. AI Agents must be able to perceive the task instruction from human and system specifications.
2. **Task Planing**. AI Agents should be able to create a step-by-step plan for complex task composition based on the perceived instruction and specifications. This usually involves the generation of critical subtask sequences, and the ability to adjust the plan dynamically in response to changes in the task or environment.
3. **Tool Usage**. On the one hand, AI Agents should possess the capacity to **select** a variety of existing tools or resources to solve complex tasks. On the other hand, AI Agents should **create** new general tools by abstracting task requirements. This ability enables the AI Agent to extend its capabilities beyond LLM itself and the existing tools, by leveraging the vast resources available in the digital world. Finally, AI Agents should be able to **execute** the selected or created tools for meeting the human request according to the resources and constraints of systems.
4. **Learning / Reflection / Memory (from Feedback)**. Learning from feedback, including correct results and exception errors, is also essential for AI Agents. They should incorporate memory, such as logging or chat history, and reflection to adapt their plans or decisions. This allows the agents to continuously improve their performance and efficiency.
5. **Summarization**. After several rounds of interaction with humans, tools, and systems, AI Agents can ultimately complete the original task provided by the users. At this point, AI Agents should be able to summarize the interaction history and provide a final answer that is concise and easy to understand for the users.


Problem
How much budget is required to provide a 100$ incentive for each colleague who has worked for five years?
LLMs
One-step Plans
1. SQL generator: "Figuring out how many colleague who has worked for five years from the database; taking it as X."
2. Python generator: "Calculating the value of 100*X with a calculator"

(a) One-step Agent (TPTU-OA)


Problem
How much budget is required to provide a 100$ incentive for each colleague who has worked for five years?
LLMs
Sequential Plan 1
SQL generator: "Figuring out how many colleague who has worked for five years from the database; taking it as X."
LLMs
Sequential Plan 2
Python generator: "Calculating the value of 100*X with a calculator"

(b) Sequential Agent (TPTU-SA)

Figure 3: The workflows of the One-step Agent and the Sequential Agent.

To endow AI Agents with the aforementioned abilities, some techniques that can be used include chain-of-thought (CoT) and vector databases, as shown in Table 1.

Table 1: A simple illustration of the techniques for endowing the key ability.

Ability	**Possible Techniques**
Perception	Multi-input Fusion
Task Planing	Zero-shot CoT and Few-shot CoT
Tool Usage	Text Matching / Code Generation / Action Grounding
Learning / Reflection / Memory	RLHF / Multi-agent Debate / Vector Database
Summarization	Attention Mechanism and Natural Language Generation

2.3 Agent Design

Task planning and tool usage represent the cornerstone of LLM's abilities. Others like perception, learning / reflection / memory (from feedback), and summarization are indeed critical, but they primarily serve to enhance and support these two core competencies. Therefore, concentrating on these two key competencies - **T**ask **P**lanning and **T**ool **U**sage (TPTU) - we have devised two distinct types of AI Agents, as depicted in Figure 3:

- The first one, named the **O**ne-step **A**gent (TPTU-OA), adopts a global perspective to interpret the original problem, effectively breaking it down into a sequence of subtasks in a single instance. This strategy fully harnesses the model's comprehensive understanding capabilities to map out the problem-solving steps for all subtasks at once. TPTU-OA underscores the significance of a holistic understanding and planning of the overall task, albeit it might lack flexibility when dealing with individual subtasks.
- The second type, referred to as the **S**equential **A**gent (TPTU-SA), emphasizes tackling the current subtask at hand. Upon successfully resolving of the ongoing subtask, this agent requests the LLMs to provide the succeeding subtask. TPTU-SA enables the model to maintain a clear and concentrated focus throughout the problem-solving journey, tackling issues incrementally. So such a methodology allows for continuous feedback and progress within the confines of addressing a broader problem.

These two distinct agent models represent two disparate problem-solving strategies - the sequential and one-step resolution[4]. In our subsequent experiments, we aim to understand their respective strengths and weaknesses, and how they can be best utilized to leverage the capabilities of LLMs in real-world scenarios.

[4]One can combine the two strategies to design a hierarchical agent, but this is beyond the scope of our paper.

3 Evaluation

We instantiate the proposed LLM-based AI Agent framework (TPTU-OA and TPTU-SA) with different LLMs and evaluate their performance on typical tasks.

3.1 Preparations

Before beginning our evaluation, we first outline the preparations. We will give detailed descriptions of the datasets, available tools, and popular large language models.

Datasets Before building the dataset, we first clarify the motivations behind our choice of tools for evaluation. The selection was guided by two primary factors: *the number of tools* to be evaluated and *the specific tools* to be included. Regarding the number of tools, it is important to state that our proposed evaluation framework is extensible. It can incorporate any number of tools as pluggable components to be managed by the LLM-based AI Agents. Some previous work typically dispatches a small number of API tools, or even a tool to solve a single task. Therefore, for the sake of simplicity and focus in our evaluation, we have decided to primarily assess two tools (which can be called multiple times) within a single scenario. Querying in a huge database is boring. Furthermore, such scenarios are quite common in daily life where the formulation and resolution of a question often involve SQL and mathematical tools. Recognizing the importance of these tools, we have chosen to focus our evaluation on SQL and Python generators, which represent the capabilities of database querying and mathematical computation, respectively. To this end, we have prepared 120 question-answer pairs that vary in complexity. These pairs provide a rigorous assessment of the LLM-based AI Agents in understanding, generating, and utilizing these essential tools. For further information on these queries and their corresponding demonstrations, please refer to Appendix A.

Tools In addition to the SQL generator and Python generator mentioned above, the tool set also includes 10 other tools that are not related to the task. More details can be found in Appendix B.

LLMs The LLMs evaluated in this paper are listed in Table 2, elaborated in Appendix C.

Table 2: The LLMs evaluated in this paper.

Organization	**Model Name**	**Model Parameters**
OpenAI	ChatGPT[12]	200B
Anthropic	Claude[13]	>52B
Shanghai AI Lab	InternLM	120B
IDEA	Ziya-13B	13B
Tsinghua University	ChatGLM-130B[14]	130B
-	Chinese-Alpaca-Plus-33B[15, 16]	33B

3.2 Evaluation on Task Planning Ability

In this section, to evaluate the planning capabilities of the LLM-based AI Agents, we have structured the evaluations as follows.

For TPTU-OA, 1) we begin by examining the agents' ability to plan the order of tool use. This is followed by 2) an evaluation of the agents' capacity to not only plan the sequence of tools but also the corresponding subtask descriptions. Subsequently, 3) we conduct a specialized planning evaluation where the agents must generate multiple sequences of key-value pairs of the form {tool: subtask description} in complex problem teardowns. Moreover, 4) we expand the toolset with additional, unrelated tools to further challenge and reassess the planning ability of the LLM-based AI Agents. Due to page limitations, we will only provide a detailed introduction to the third evaluation mentioned above, which is the most comprehensive. The details of the previous evaluations can be found in Appendix D.

For TPTU-SA, we follow the regime that the agent should generate multiple sequences of key-value pairs of the form {tool: subtask description} for evaluation.

3.2.1 TPTU-OA: The Planning of Tool-Subtask Pair

We prompt the agent to generate multiple sequences, each consisting of a key-value pair in the format of {tool: subtask description} that associates a tool with its respective subtask description. This allows us to simultaneously plan the tool choice and subtask without the risk of improper matching. Moreover, it offers the flexibility to update the planned sequences in real-time based on evolving problem feedback, enhancing adaptability and efficiency when addressing complex tasks.

With this consideration, we have designed a unique prompt that encourages this advanced problem-solving strategy. The key improvement in this prompt is its directive for the LLM-based AI Agents to stringently adhere to the predefined dictionary format. To facilitate this, we offer several demonstrations in our desired format, serving as references for the language model to follow.

Table 3: The evaluation results for the planning of Tool-Subtask pair.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	75%	90%	20%	0%	5%	55%

After feeding the prompt to these LLM-based AI Agents, we get results shown in Table 3. Analyzing the results from Tables 18 and 3, we observe a marked improvement of 52.9% when the tool-subtask pairs are generated in a unified format compared to separate generation of tools and subtasks.

This significant performance enhancement can likely be attributed to the close coupling between tools and their associated subtasks in our unified generation strategy. When tools and subtasks are generated separately, there is a potential disconnect or lack of coherence between the two, which could lead to less accurate or efficient solutions. In contrast, by generating tool-subtask pairs together, we ensure that each tool is directly tied to its relevant subtask, leading to a more coordinated and effective problem-solving approach. This might explain the observed increase in overall performance.

3.2.2 TPTU-SA: The Planning of Tool-Subtask Pair Generation

Upon identifying the drawbacks of first generating a list of tools and then generating corresponding subtask descriptions, we decided to focus subsequent tests on the generation of tool-subtask pairs. Consequently, in this section, we evaluate the capability of TPTU-SA to generate these tool-subtask pairs. To achieve the goal of recursively generating tool-subtask pairs, we have designed prompts as illustrated in Table 27 of Appendix F.

Table 4: The evaluation results for the planning of Tool-Subtask with the sequential agent.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	80%	100%	10%	0%	0%	65%

The evaluation results are shown in Table 4. Compared with results shown in Table 3, TPTU-SA generally performs better than TPTU-OA especially for high–performing LLMs (e.g., ChatGPT, Claude and InternLM). We propose the following potential reasons for this observation:

1. **Sequentiality Mimics Human Problem-Solving**: In real-world scenarios, humans tend to solve complex problems by breaking them down into smaller, manageable subtasks which are often handled sequentially. Sequential agents are designed to mimic this step-by-step approach, which might inherently suit complex problem-solving better.
2. **Richer Contextual Understanding**: Sequential agents are exposed to the outcome of each previous subtask before moving on to the next one. This iterative process could facilitate a richer understanding of the problem context, enabling more accurate task planning and tool usage.
3. **Flexibility in Task Management**: In comparison to one-step agents, sequential agents might have more flexibility in managing tasks. They have the opportunity to correct errors or adjust their strategy after each step, which can lead to improved overall performance.

4. **Improved Learning From History**: The sequential process provides a history of actions and results which can be beneficial in learning. The agent can use this history to make better predictions about what tool to use next or what subtask to tackle, leading to more accurate and efficient problem-solving.

These points of analysis suggest that the structure and operation of sequential agents inherently confer certain advantages in complex problem-solving scenarios, leading to their superior performance.

3.3 Evaluation on Tool Usage Ability

We evaluate the end to end performance of two agents (TPTU-OA and TPTU-SA) based on different LLMs. The role of the agents is to break down complex questions into simpler sub-questions and plan corresponding tools to solve them, based on the available toolset and corresponding tool descriptions. We also evaluate the effectiveness of single-tool usage for SQL generation and mathematical code generation, and the results and discussion can be found in Appendix E.

We now aim to utilize the one-step agent and sequential agent, which we designed, to conduct an evaluation involving multiple tools. Corresponding prompts for each agent type have been crafted and are presented in Table 32 and Table 33 of Appendix F, respectively.

In this phase of the evaluation, we need to automatically invoke the respective tools through code and produce the results. Given that user interface-based LLMs lack the capability to call external tools, we will only utilize the following four API-based LLMs (ChatGPT, Ziya, Chinese-Alpaca, and InternLM) for this comprehensive evaluation of external tool usage ability.

Table 5: The evaluation results for end-to-end ability of multiple tools.

Model	ChatGPT	Ziya	Chinese-Alpaca-Plus	InternLM
TPTU-OA	50%	0%	0%	15%
TPTU-SA	55%	0%	0%	20%

With agents mentioned above, the final results are presented in Table 5. The evaluation results demonstrate varying levels of task planning and tool usage capabilities among the four API-based LLMs. In the TPTU-OA evaluation, ChatGPT achieved a performance rate of 50%, significantly outperforming the other models, with InternLM at 15%, while both Ziya and Chinese-Alpaca did not manage to complete any tasks successfully, resulting in a score of 0%. In the TPTU-SA evaluation, an overall slight improvement was observed. ChatGPT maintained its leading position, with a slightly improved performance rate of 55%. InternLM also exhibited better performance, achieving a score of 20%, whereas Ziya and Chinese-Alpaca-Plus again failed to register any successful task completion.

These results reflect a notable discrepancy in the performance of LLMs when it comes to using external tools. ChatGPT and InternLM have demonstrated some ability to navigate these tasks, but their performance rates suggest there is significant room for improvement. Ziya and Chinese-Alpaca-Plus' performance indicates a struggle to effectively utilize external tools in their current state.

The differential performance between the TPTU-OA and TPTU-SA evaluation hints at the possible impact of the agent design on the LLMs' task execution ability. In particular, the performance increase under the sequential agent framework suggests that breaking down tasks into sequential steps might help LLM-based AI Agents better utilize external tools. This insight could prove valuable in future improvements and developments of LLM-based AI Agents. However, even with this approach, it is clear that LLM-based AI Agents are far from perfect when it comes to effectively using external tools for complex tasks. This finding underlines the importance of further investigation and improvement in this domain.

3.4 Insightful Observations

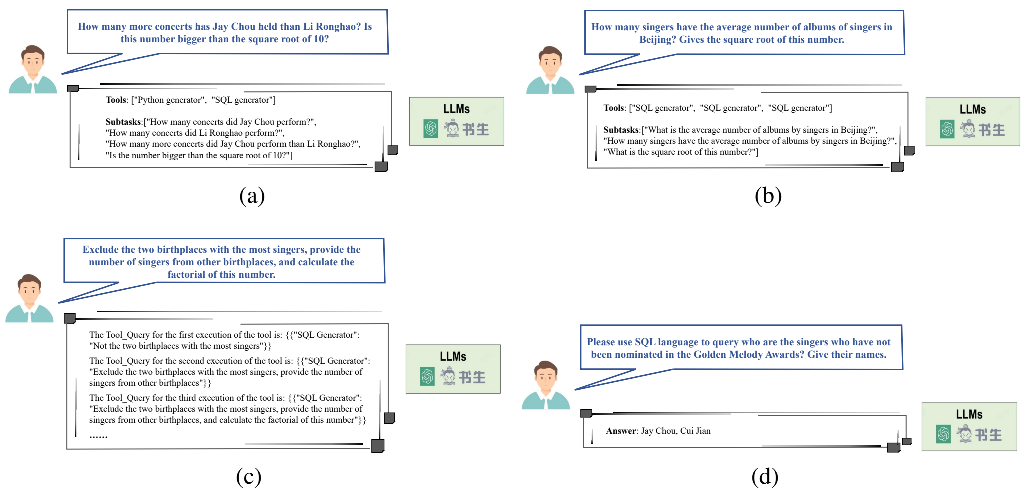
Upon closer observation of our experimental results, we have identified several phenomena that deserved further exploration. These findings serve to broaden our understanding of LLM-based agents' behavior and capabilities and provide essential insights that could shape future research in this field. In the following, we will dissect these phenomena as shown in Figure 4, casting light on the weaknesses of LLM-based agents in the context of task planning and tool usage.

Misunderstanding Output Formats LLMs frequently encounter difficulty when output is required in specific formats such as lists or dictionaries. One such example includes inconsistencies between the number of tools and corresponding subtasks, leading to formatting issues that hinder the correct execution of tasks.

Struggling to Grasp Task Requirements LLMs might incorrectly disintegrate subproblems or apply unsuitable tools to carry out the subproblem. For example, an LLM might attempt to solve a purely mathematical problem by employing an SQL tool or could misunderstand similar terms like cube extraction and cube roots.

Endless Extensions LLMs tend to overutilize a particular tool, even in instances where a single use would suffice for the correct result. This issue can lead to extended and nonsensical planning, where the same subtask is repeatedly solved.

Lack of Summary Skills LLMs do not take into account the responses to subproblems, relying instead on their internalized knowledge to generate the final answer. This may lead to a scenario where the final response only addresses a portion of the original query.



(a)

How many more concerts has Jay Chou held than Li Ronghao? Is this number bigger than the square root of 10?

Tools: ["Python generator", "SQL generator"]

Subtasks:["How many concerts did Jay Chou perform?",
"How many concerts did Li Ronghao perform?",
"How many more concerts did Jay Chou perform than Li Ronghao?",
"Is the number bigger than the square root of 10?"]

(b)

How many singers have the average number of albums of singers in Beijing? Gives the square root of this number.

Tools: ["SQL generator", "SQL generator", "SQL generator"]

Subtasks:["What is the average number of albums by singers in Beijing?",
"How many singers have the average number of albums by singers in Beijing?",
"What is the square root of this number?"]

(c)

Exclude the two birthplaces with the most singers, provide the number of singers from other birthplaces, and calculate the factorial of this number.

The Tool_Query for the first execution of the tool is: {{"SQL Generator": "Not the two birthplaces with the most singers"}}

The Tool_Query for the second execution of the tool is: {{"SQL Generator": "Exclude the two birthplaces with the most singers, provide the number of singers from other birthplaces"}}

The Tool_Query for the third execution of the tool is: {{"SQL Generator": "Exclude the two birthplaces with the most singers, provide the number of singers from other birthplaces, and calculate the factorial of this number"}}

......

(d)

Please use SQL language to query who are the singers who have not been nominated in the Golden Melody Awards? Give their names.

Answer: Jay Chou, Cui Jian

Figure 4: Several phenomena that deserved further exploration. (a) Inconsistencies between the number of tools and corresponding subtasks. (b) Solving a purely mathematical problem by employing a SQL generator. (c) Unnecessary repetition of subtasks. (d) Answering questions using common sense instead of generating code.

By identifying and addressing these common issues, we stand a better chance at improving and refining LLMs, thereby unlocking their full potential.

4 Related Work

The integration of specialized tools with LLM has unlocked substantial potential in addressing intricate tasks. In this section, we offer a concise synopsis of the relevant research pertaining to tool learning based on LLMs.

The initial advancements in tool learning have been constrained by the capabilities of artificial intelligence (AI) models. [17] Traditional deep learning approaches exhibit limitations in terms of comprehension of tool functionality and user intentions, and common sense reasoning abilities. Consequently, these limitations directly result in a notable decline in the stability and precision of tool learning methodologies. Recently, the advent of LLM has marked a pivotal juncture in the realm of tool learning. LLMs encompass a broad spectrum of common sense cognitive capabilities and exhibit remarkable proficiencies in natural language processing, reasoning, and interactive decision-making [18–22]. These attributes furnish indispensable prerequisites for LLMs to comprehend user intentions and effectively employ tools in tackling intricate tasks [23]. Simultaneously, the advancement of fine-tuning [24–28] and in-context learning [29, 30] technology has offered robust support to LLM in addressing increasingly intricate challenges. In addition, tool usage can mitigate the inherent limitations of LLMs, encompassing the acquisition of up-to-date information from real-world events, refined mathematical computational abilities, and the mitigation of potential hallucinatory phenomena [31].

Within the realm of embodied intelligence [32–34], LLM engages in direct interactions with tangible tools like robots in order to enhance their cognitive abilities, optimize work productivity, and expand functional capacities. LLM possesses the capability to automatically devise action steps based on user intentions, enabling the guidance of robots in the completion of tasks [35–43], or alternatively, to directly generate underlying code that can be executed by robots [44–48]. Palm-E [40] introduced a multimodal language model which seamlessly integrates sensor data into its framework, enabling efficient planning of robot actions and task completion. Code as Policies (CaP) [48] facilitates the transformation of natural language instructions into code fragments that can be directly compiled and executed on robots. As for Inner Monologue [38], LLM incorporates diverse environmental feedback to construct inner monologues, thereby formulating effective robot control strategies.

In addition to directly changing the real environment through interaction with tools in the physical world, LLM can also utilize software tools such as search engines [49–57], mobile [58, 59], Microsoft Office [60, 61], calculators [62–64], deep models [65–73] and other versatile APIs [74, 5, 75–78] to enhance model performance or complete complex workflows through flexible control of the software. Toolformer [5] employs a self-supervised methodology to fine-tune the language model, enabling it to acquire the ability to automatically invoke APIs. ART [79] leverages CoT [80] and In-context Learning [73, 31] techniques to automatically generate multi-step reasoning processes for new tasks, while also selecting and utilizing the most appropriate available tool at each step. ASH [52] utilizes LLM for sequence hierarchical decision-making to achieve web navigation tasks. WebGPT [56] and WebCPM [54] use network search to assist in implementing Question Answering tasks. In addition, RCI [81] recursively criticizes and improves itself to execute computer tasks guided by natural language according to the prompting scheme.

The usage of tools is contingent upon the accessibility of external tools. Recently, efforts have been made to employ LLM as a tool creator in order to generate tools that can be utilized for diverse requests [82–89]. This development has consequently raised the demands placed on LLM. And these created tools are typically implemented as Python or SQL functions.

5 Conclusion

In this paper, we have introduced a structured framework specially designed for LLM-based AI Agents, with an emphasis on their abilities in task planning and tool usage. This framework, coupled with our design of two distinct types of agents assigned for the inference process, allows for a comprehensive evaluation of the capabilities of current open-source LLMs, thereby yielding critical insights into their effectiveness. Furthermore, our research highlights the significant potential of LLMs in managing complex tasks, revealing the exciting prospects they hold for future research and development. As we continue to explore and improve upon these models, we move closer to unlocking their full potential in a wide range of real-world applications.

References

[1] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

[4] OpenAI. Gpt-4 technical report, 2023.

[5] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[6] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1:7–38, 1998.

[7] Nicholas R Jennings and Michael Wooldridge. Applying agent technology. *Applied Artificial Intelligence an International Journal*, 9(4):357–369, 1995.

[8] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International workshop on agent theories, architectures, and languages*, pages 21–35. Springer, 1996.

[9] Cristiano Castelfranchi. Modelling social action for ai agents. *Artificial intelligence*, 103(1-2):157–182, 1998.

[10] Jacques Ferber and Gerhard Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-wesley Reading, 1999.

[11] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11:387–434, 2005.

[12] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[13] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[14] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*, 2022.

[15] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[16] Yiming Cui, Ziqing Yang, and Xin Yao. Efficient and effective text encoding for chinese llama and alpaca. *arXiv preprint arXiv:2304.08177*, 2023.

[17] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[18] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*, 2023.

[19] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712*, 2023.

[20] Chaoning Zhang, Chenshuang Zhang, Chenghao Li, Yu Qiao, Sheng Zheng, Sumit Kumar Dam, Mengchun Zhang, Jung Uk Kim, Seong Tae Kim, Jinwoo Choi, et al. One small step for generative ai, one giant leap for agi: A complete survey on chatgpt in aigc era. *arXiv preprint arXiv:2304.06488*, 2023.

[21] Fei Yu, Hongbo Zhang, and Benyou Wang. Nature language reasoning, a survey. *arXiv preprint arXiv:2303.14725*, 2023.

[22] Zekun Wang, Ge Zhang, Kexin Yang, Ning Shi, Wangchunshu Zhou, Shaochun Hao, Guangzheng Xiong, Yizhi Li, Mong Yuan Sim, Xiuying Chen, et al. Interactive natural language processing. *arXiv preprint arXiv:2305.13246*, 2023.

[23] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023.

[24] Wenhao Yu, Chenguang Zhu, Zaitang Li, Zhiting Hu, Qingyun Wang, Heng Ji, and Meng Jiang. A survey of knowledge-enhanced text generation. *ACM Computing Surveys*, 54(11s):1–38, 2022.

[25] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[26] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[27] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[28] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. Gpt understands, too. *arXiv preprint arXiv:2103.10385*, 2021.

[29] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[30] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

[31] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023.

[32] Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2): 230–244, 2022.

[33] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019.

[34] Stan Franklin. Autonomous agents as embodied ai. *Cybernetics & Systems*, 28(6):499–520, 1997.

[35] Weiyi Zhang, Yushi Guo, Liting Niu, Peijun Li, Chun Zhang, Zeyu Wan, Jiaxiang Yan, Fasih Ud Din Farrukh, and Debing Zhang. Lp-slam: Language-perceptive rgb-d slam system based on large language model. *arXiv preprint arXiv:2303.10089*, 2023.

[36] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on Robot Learning*, pages 492–504. PMLR, 2023.

[37] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pages 287–318. PMLR, 2023.

[38] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[39] Boyuan Chen, Fei Xia, Brian Ichter, Kanishka Rao, Keerthana Gopalakrishnan, Michael S Ryoo, Austin Stone, and Daniel Kappler. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11509–11522. IEEE, 2023.

[40] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[41] Naoki Wake, Atsushi Kanehira, Kazuhiro Sasabuchi, Jun Takamatsu, and Katsushi Ikeuchi. Chatgpt empowered long-step robot control in various environments: A case application. *arXiv preprint arXiv:2304.03893*, 2023.

[42] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. *arXiv preprint arXiv:2307.06135*, 2023.

[43] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. *arXiv preprint arXiv:2212.04088*, 2022.

[44] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[45] Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, et al. Open-world object manipulation using pre-trained vision-language models. *arXiv preprint arXiv:2303.00905*, 2023.

[46] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[47] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res*, 2:20, 2023.

[48] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[49] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020.

[50] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[51] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

[52] Abishek Sridhar, Robert Lo, Frank F Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation. *arXiv preprint arXiv:2305.14257*, 2023.

[53] Hiroki Furuta, Ofir Nachum, Kuang-Huei Lee, Yutaka Matsuo, Shixiang Shane Gu, and Izzeddin Gur. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*, 2023.

[54] Yujia Qin, Zihan Cai, Dian Jin, Lan Yan, Shihao Liang, Kunlun Zhu, Yankai Lin, Xu Han, Ning Ding, Huadong Wang, et al. Webcpm: Interactive web search for chinese long-form question answering. *arXiv preprint arXiv:2305.06849*, 2023.

[55] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35: 20744–20757, 2022.

[56] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[57] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[58] Bryan Wang, Gang Li, and Yang Li. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2023.

[59] Danyang Zhang, Lu Chen, and Kai Yu. Mobile-env: A universal platform for training and evaluation of mobile interaction. *arXiv preprint arXiv:2305.08144*, 2023.

[60] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhaoxiang Zhang. Sheetcopilot: Bringing software productivity to the next level through large language models. *arXiv preprint arXiv:2305.19308*, 2023.

[61] Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, et al. Tablegpt: Towards unifying tables, nature language and commands into one gpt. *arXiv preprint arXiv:2307.08674*, 2023.

[62] Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji-Rong Wen. Chatcot: Tool-augmented chain-of-thought reasoning on\\chat-based large language models. *arXiv preprint arXiv:2305.14323*, 2023.

[63] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

[64] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

[66] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

[67] Zhaoyang Liu, Yinan He, Wenhai Wang, Weiyun Wang, Yi Wang, Shoufa Chen, Qinglong Zhang, Yang Yang, Qingyun Li, Jiashuo Yu, et al. Internchat: Solving vision-centric tasks by interacting with chatbots beyond language. *arXiv preprint arXiv:2305.05662*, 2023.

[68] Yingqiang Ge, Wenyue Hua, Jianchao Ji, Juntao Tan, Shuyuan Xu, and Yongfeng Zhang. Openagi: When llm meets domain experts. *arXiv preprint arXiv:2304.04370*, 2023.

[69] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

[70] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.

[71] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[72] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023.

[73] Liangyu Chen, Bo Li, Sheng Shen, Jingkang Yang, Chunyuan Li, Kurt Keutzer, Trevor Darrell, and Ziwei Liu. Language models are visual reasoning coordinators. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*, 2023.

[74] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2023.

[75] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[76] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434*, 2023.

[77] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023.

[78] Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *arXiv preprint arXiv:2305.11554*, 2023.

[79] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

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

[81] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*, 2023.

[82] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

[83] Ryan Hardesty Lewis and Junfeng Jiao. Computegpt: A computational chat model for numerical problems. *arXiv preprint arXiv:2305.06223*, 2023.

[84] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR, 2023.

[85] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[86] Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Disentangling abstract and concrete reasonings of large language models through tool creation. *arXiv preprint arXiv:2305.14318*, 2023.

[87] Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, Wang You, Ting Song, Yan Xia, et al. Low-code llm: Visual programming over llms. *arXiv preprint arXiv:2304.08103*, 2023.

[88] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433*, 2023.

[89] Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueting Zhuang. Data-copilot: Bridging billions of data and humans with autonomous workflow. *arXiv preprint arXiv:2306.07209*, 2023.

A Detailed Description of Dataset

Simple SQL queries: These queries typically involve basic operations such as SELECT, FROM, WHERE, GROUP BY, etc. They are used to retrieve, filter, group, and sort data from a single table. We give the Schema of two tables in the SQL database in Table 6 and 7 and list several examples in Table 8.

Table 6: Schema of the Person table

Person	
Column Name	**Type**
id	TEXT
name	TEXT
age	INTEGER
sex	TEXT
school	TEXT
phone	TEXT
qualifications	TEXT
ability	TEXT

Table 7: Schema of the School table

School	
Column Name	**Type**
id	TEXT
name	TEXT
info_985	TEXT
info_211	TEXT

Table 8: Demostrations of simple SQL queries.

Table ID	**Question**	**Answer**	**SQL reference**
Person	Average ages	35.16	select avg(age) from Person
Person	How many men	12	select count(*) from Person where sex = 'male'
School	How many schools are both '985' and '211' institutions?	11	select count(*) from School where info_985 = 'yes' and info_211 = 'yes';

Complex nested SQL queries: These queries contain subqueries, which are SQL queries nested inside a larger query. Nested queries can be used in various clauses such as SELECT, FROM, WHERE, and HAVING. They provide a way to perform multiple operations or calculations across multiple tables. We give the Schema of two tables in the SQL database in Table 9, 10, 11, and 12 and list several examples in Table 13.

Table 9: Schema of GoldenMelodyAwards

GoldenMelodyAwards	
Column Name	**Type**
Nominated_Count	INTEGER
Competing_Count	INTEGER
Awards_Count	INTEGER
Award_Name	TEXT
Host	TEXT
Year	TIME

Table 10: Schema of the AwardNominees table

AwardNominees	
Column Name	**Type**
Singer_ID	INTEGER
Nominated_Work	TEXT
Award_Name	TEXT
Award_Edition_ID	INTEGER

Complex nested queries utilizing multiple tools: These are advanced queries that involve multiple tools, such as SQL queries, python code generation, user-defined functions, etc. We give the Schema

Table 11: Schema of the Singers table

Singers	
Column Name	**Type**
Name	TEXT
Song_Count	INTEGER
Album_Count	INTEGER
Fan_Count	INTEGER
Gender	TEXT
Singer_ID	INTEGER

Table 12: Schema of the RecordCompanies table

RecordCompanies	
Column Name	**Type**
Record_Company	TEXT
Signing_Date	TIME
Singer_ID	INTEGER

Table 13: Demostrations of complex nested SQL queries.

Table ID	Question	Answer	SQL reference
GoldenMelody-Awards	Golden Melody hosts, excluding the two with the least awards.	"26th Golden Melody", "27th Golden Melody"	select Award_Name from GoldenMelodyAwards where Host not in (select Host from GoldenMelodyAwards group by Host order by avg (Awards_Count) asc limit 2)
AwardNominees & Singers	Names of singers never nominated for Golden Melody Awards.	"Jay Chou", "Jian Cui"	select Name from Singers where Singer_ID not in (select Singer_ID from AwardNominees)
RecordCompanies & Singers	Name and gender of singers without a record company.	"Penny Tai:Femal"	select Name, Gender from Singers where Singer_ID not in (select Singer_ID from RecordCompanies);
GoldenMelody-Awards	How many times is the 27th Golden Melody count of the 28th's?	1	select a.Awards_Count / b.Awards_Count from (select Awards_Count from GoldenMelodyAwards where Award_Name == '27th Golden Melody') a , (select Awards_Count from GoldenMelodyAwards where Award_Name == '28th Golden Melody') b

of two tables in the SQL database in Table 14, and 15 and list several examples in Table 16. For verifying the planning ability of the LLM-based AI Agents, we select this type of query.

Table 14: Schema of the Journal table

Journal	
Column Name	**Type**
Name	TEXT
First_Issue_Date	TIME
Journal_ID	INTEGER
Category	TEXT
Sponsor_Organization	TEXT
Country	TEXT
s Language	TEXT
Publication_Count	INTEGER

Table 15: Schema of the CoverPersonality table

CoverPersonality	
Column Name	**Type**
Person_ID	INTEGER
Journal_ID	INTEGER
Count	INTEGER

Table 16: Demostrations of complex nested queries utilizing multiple tools.

Table ID	Question	Answer	Planning Tools	SQL reference	Code reference
Journal & Cover-Personality	Calculate the exponential of 3 and list the names and languages of journals with no cover personality.	[20.08, "The Economist: Chinese, Reader's Digest: English."]	["PythonREPL", "SQL Generator"]	select Name, Language from Journal where Journal_ID not in (select Journal_ID from CoverPersonality)	import math; return math.exp(3)
CoverPersonality & Journal	Compute 4's factorial, compare with GCD of 212 and list the names and languages of journals with no cover personality.	[4, "The Economist: Chinese, Reader's Digest: English."]	["PythonREPL", "SQL Generator"]	select Name, Language from Journal where Journal_ID not in (select Journal_ID from CoverPersonality)	import math; math.gcd(math.factorial(4), 212)
Journal	Calculate the square root of 24, and query for the language whose average number of published issues exceeds the overall average	[4.8989795, "English"]	["PythonREPL", "SQL Generator"]	select Language from Journal group by Language having avg (Publication_Count) > (select avg (Publication_Count) from Journal)	import math; math.sqrt(24)
CoverPersonality	Compute the log base 10 of 5, then identify cover figures appearing less than the overall max frequency across journals.	[0.69897, "Qing Hai, Xiaoming Huang, Cristiano Ronaldo, Kobe Bryant"]	["PythonREPL", "SQL Generator"]	select Person_ID from CoverPersonality where Count < (select max (Count) from CoverPersonality)	import math; math.log10(5)

B Detailed Description of Tools

We have defined a total of 12 available tools for the selection of the LLM-based AI Agents for evaluation. They are defined as follows:

- SQL generator: Given an input question and a database, create a syntactically correct SQLite query statement.
- Python generator: Given an input question and some information, generate a syntactically correct Python code.
- Weather query tool: Given a location, output the current real-time weather at that location.
- Image generator: Given a text description, generate a related image.
- Text extractor: Given a link to an image, extract the corresponding text and its position coordinates.
- Translator: Given a piece of text, translate it into other languages.
- Bing Searcher: Given a piece of text, conduct a search on the Bing browser and return content.
- Shell generator: Given an input question and some information, generate a syntactically correct Shell code.
- Java generator: Given an input question and some information, generate a syntactically correct Java code.
- Wikipedia searcher: Given a piece of text, conduct a search on Wikipedia and return content.
- Office software: Given a text description, automatically generate corresponding long documents or spreadsheets or PPTs.
- Movie player: Given a movie name, automatically play the corresponding movie resources.

C Detailed Description of LLMs

- **GPT** series developed by OpenAI boasts a powerful language model with a vast number of parameters, enabling it to tackle intricate problems efficiently. This paper aims to evaluate the performance of ChatGPT, which balances the performance with costs (the number of OpenAI API calls).
- **Claude** is committed to maintaining honesty and ensuring user safety, which is developed by Anthropic. With its impressive size, Claude ranks among the largest language models globally and poses a formidable challenge to ChatGPT as a strong competitor.
- **InternLM**, a sophisticated language model developed by Shanghai AI Lab, boasts a multi-round dialogue capability and an impressive ability to comprehend super-long text. This language model is meticulously designed to cater to the nuances of the Chinese language, enabling it to comprehensively understand and effectively process Chinese text. Here, we adopted the version with 120 billion parameters.
- **Ziya** is an expansive and robust pre-training model developed by IDEA, derived from the LLaMa with 13 billion parameters. This comprehensive model exhibits a wide range of capabilities, including translation, programming, and mathematical calculations. Notably, it stands out as a bilingual LLM, highlighting its ability to effectively process and comprehend text in Chinese.
- **ChatGLM**, developed by Tsinghua University, is an open-source dialogue language model that supports bilingual Q&A in Chinese and English, with a particular focus on Chinese optimization. Built on the General Language Model (GLM) architecture and utilizing model quantization technology, the ChatGLM can be easily deployed on consumer-grade graphics cards, enabling local implementation by users.
- **Chinese-Alpaca-Plus** is achieved by extending LLaMA's existing vocabulary with an additional 20,000 Chinese tokens from Meta AI (formerly known as Facebook AI Research Laboratory). In this version, we use a model with 33 billion parameters. The training text has been expanded to 120GB, and the fine-tuning instruction data has been increased to 4.3M.

D Additional Evaluation on Task Planning Ability

D.0.1 TPTU-OA: Tool Order Planning

Here, we utilize two kinds of tools for problem-solving: the SQL generator, which retrieves data from databases, and the Python generator, adept at addressing mathematical questions.

To validate the capacity of the LLM-based AI Agents to strategically plan for the tool order, we designed the prompt as shown in Table 23 of Appendix F. This design is motivated by the goal to assess the ability of LLM-based AI Agents to understand complex problems, subsequently decomposing them into a sequence of simpler tasks executed by appropriately selected tools. Specifically, we require the LLM-based AI Agent to follow our instructions, select tools from our pre-defined tool set with detailed function descriptions, conform to the given format strictly, and understand the demonstrations to learn from them.

Upon feeding these prompts into the LLM-based AI Agents under evaluation, we obtained the following accuracy rates for the tool planning, as shown in Table 17.

Table 17: The evaluation results for the planning of tool order generation.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	100%	100%	45%	45%	20%	80%

The results of our experiments indicate that models, notably Ziya and ChatGLM, frequently grapple with the generation of lists in the correct format. For other models, the predominant challenges lie in generating tools in the correct sequence or in the occasional omission of necessary tools. Nonetheless, the issue of parsing list formats is generally negligible.

These findings suggest that the majority of LLM-based AI Agents possess a fundamental capability to analyze the tool needs of a given problem and understand its task requirements. To further explore whether these LLM-based AI Agents can effectively break down the original problem into subtasks, we proceed to the following section.

D.0.2 TPTU-OA: Tool Order Planning and Subtask Description Generation

Simply planning the order of tool usage is not sufficient to fully address a problem. To truly solve it, we need to provide a guide or instructions for the usage of each tool, that is, a decomposed subtask description. Therefore, we can decompose the original complex problem into two separate sequences. One sequence represents the order in which the tools are utilized, while the other sequence corresponds to the subtask descriptions that each tool in the tool sequence aims to resolve. A problem is only truly solved when both the tool and subtask description sequences have been successfully planned. In order to verify whether LLM-based AI Agents truly have the ability to solve complex problems, we designed a new prompt as shown in Table 24 of Appendix F. The main improvement is to plan the corresponding subtask description for each tool after the tool planning is completed.

Table 18: The evaluation results for the planning of tool order and subtask description generation.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	55%	15%	10%	10%	0%	45%

After feeding the prompt to these LLM-based AI Agents, we get results shown in Table 18.

Although the generation of tool sequences and their corresponding subtask descriptions might be an effective way to problem-solving, there is a significant decrease in accuracy for all LLMs as can be seen. We hypothesize that there are a few potential drawbacks to this method:

1. **Difficulty in Error Tracking and Debugging**. Generating the complete tool and subtask sequences may make it more challenging to track and debug errors. If an error arises within the sequence, it might require a total regeneration instead of a simple modification or repair to the erroneous part.

2. **Tool-Subtask Pairing Issue**. If all tool sequences and subtask descriptions are generated independently, there's an inherent risk of misalignment between the tools and their corresponding subtasks. This could potentially lead to an improper pairing, which, in turn, could result in a flawed or ineffective solution that fails to appropriately resolve the given problem.
3. **Lack of Flexibility**. The approach may lack this flexibility when facing complex problems requiring adjustments to the tool or subtask sequence.
4. **Dependency on Global Information**. Generating the entire tool and subtask sequences requires a global understanding and planning of the entire problem. However, in some instances, certain parts of the problem might not be clear at the early stages of problem-solving, which could pose challenges within this framework.

D.0.3 TPTU-OA: The Planning of Tool-Subtask Pair with Unrelated Tools

As far, our analysis and evaluation have been primarily focused on the LLM-based AI Agents' proficiency in planning with specific tools. However, we are also interested in how it would perform when faced with many irrelevant or similar tools. Therefore, for a more comprehensive evaluation, we expanded the prompt in Table 25 to include an additional ten unrelated tools given in Appendix B, as illustrated in Table 26.

Table 19: The evaluation results for the planning of Tool-Subtask pair with unrelated tools.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	70%	90%	10%	0%	5%	50%

After feeding the prompt to these LLM-based AI Agents, we get results shown in Table 19. The results from our expanded evaluation demonstrate that even when presented with irrelevant or similar tools and descriptions, LLM-based AI Agents consistently avoid selecting these unrelated tools (i.e., the accuracy has remained unchanged or exhibited only a marginal decrease compared with Table 3). This outcome indicates the effectiveness of our designed prompt, which successfully guides the LLM-based agents to understand the appropriate tool sequence for complex problem decomposition.

This observation reinforces the notion that a well-structured and informative prompt can efficiently guide AI Agents to understand the core essence of the problem, thereby enabling them to sift through irrelevant information and focus on key tasks. This successful discrimination against unrelated tools also points towards the models' ability to understand the specific context of a problem and select the appropriate tools, thereby enhancing the overall problem-solving process.

E Additional Evaluation on Tool Usage Ability

Our aim is to systematically assess how effectively these models can use various tools, focusing on their proficiency with SQL and other coding languages.

E.1 The Effectiveness of simple SQL Creation

Using the schemas provided in Table 6 and Table 7, we construct questions similar to those in Table 8, and refer readers to Appendix A. These questions are posed to various LLMs using our specifically designed prompts in Appendix F.

Following the tailored prompts, the LLMs are evaluated based on their responses to the presented queries. The results of this comprehensive assessment are compiled and exhibited in Table 20.

Table 20: The evaluation results for simple SQL questions.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	90%	100%	50%	30%	20%	90%

This verifies the capabilities of each LLM in handling varying simple single-table SQL queries, thus providing a basis for comparison and analysis.

E.2 The Effectiveness of Complex Nested SQL Creation

Using the schemas provided in Table 9, 10, 11, and 12, we construct questions similar to those in Table 13, and refer readers to Appendix A. For complex nested SQL questions, to further verify the SQL tool creation capability of LLMs, we have designed two types of prompts. One is the direct-guidance type, which explicitly informs the model that it needs to generate nested SQL query statements, as shown in Table 29 in Appendix F.

The other is based on the Chain-of-Thought (CoT) [80] approach, which leverages the model's ability to reason step by step to comprehend and craft SQL tools. This method guides the model to sequentially generate SQL query clauses based on the problem context, thus breaking down the complex query generation task into smaller and manageable subtasks. This approach provides the model with a structured way to handle complex SQL tasks and showcases its capacity to engage in incremental reasoning and problem-solving.

The design of these two types of prompts serves as the backbone of our evaluation for complex nested SQL questions. While the direct-guidance approach focuses on testing the model's raw ability to generate SQL queries when explicitly instructed, the CoT-based approach evaluates a more nuanced capability: the model's reasoning and problem-solving skills in a step-by-step manner. Both these methods present unique challenges and offer valuable insights into the strengths and potential areas of improvement for the large language model's SQL tool generation ability. Subsequently, we will explore these two dimensions based on our experimental evaluations shown in Table 21.

Table 21: The evaluation results for complex nested SQL questions.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Direct-based	80%	100%	50%	60%	0%	60%
CoT-based	80%	100%	40%	70%	0%	50%

From the above results in Table 21, it is clear that different models possess varying levels of proficiency in handling complex nested SQL tasks. Some models, like Claude, exhibit a robust capability in SQL generation, no matter whether the approach is direct or CoT-based. Most of these models demonstrate the SQL tool usage capability.

Specifically, some models such as ChatGLM show a distinct preference for the CoT-based approach, their performance improves when problems are broken down into smaller, manageable subtasks. This suggests that these models may have a stronger ability in sequential problem-solving and benefit more from step-by-step guidance. Conversely, models like Ziya and InternLM show a drop in performance when tasks are guided in the CoT-based format. This might indicate challenges in managing dependencies between subtasks or handling the continuity in sequential problem-solving. Lastly, Chinese-Alpaca-Plus shows significant room for improvement in complex SQL generation tasks. This shows that not all models are equally suited to handle advanced problem-solving involving nested SQL queries.

Overall, these findings underscore the importance of tailoring evaluation and training methodologies to the individual strengths and weaknesses of each model. By adopting this approach, we can better understand the performance variations across different models and provide targeted improvements to enhance their problem-solving abilities. Furthermore, this analysis highlights the potential of LLM-based agents in real-world applications, and the need to push their boundaries through continued research and development.

E.3 The Effectiveness of Mathematical Code Creation

Following our evaluation of the LLM's proficiency in creating complex SQL queries, we now shift our focus to another tool creation: the creation of mathematical code. To the best of our knowledge, while large language models possess significant capabilities, they often fall short of providing highly accurate solutions to mathematical problems. Guiding these LLMs to generate mathematical code, and subsequently leveraging external tools to execute and derive the solutions, could significantly enhance their ability to tackle mathematical challenges.

In the upcoming section, we will conduct a detailed evaluation of guiding these LLMs to generate mathematical code. We aim to shed light on the true capability of these models in generating mathematical code and to elucidate the extent to which they can be utilized to aid in mathematical problem-solving. The prompt about how to guide LLMs is shown in Table 31 in Appendix F.

Table 22: The evaluation results for mathematical questions.

Model	ChatGPT	Claude	Ziya	ChatGLM	Chinese-Alpaca-Plus	InternLM
Accuracy	90%	85%	50%	0%	55%	95%

The results shown in Table 22 indicate that the capabilities of LLM-based agents to generate mathematical code vary considerably. High-performing models like ChatGPT, Claude, and InternLM display excellent proficiency, suggesting their potent ability to solve complex mathematical tasks. Middle-tier models, such as Ziya, show moderate success, indicating the potential for improvement and adaptability with the right training and optimization. Surprisingly, Alpaca demonstrated a notable proficiency in mathematical tasks, despite its poor performance in SQL generation, suggesting a possible inclination towards mathematical problems. In contrast, ChatGLM struggles significantly with mathematical code generation, underlining a potential weak spot in its capabilities and the need for focused improvement in this area.

Overall, these results underscore the task-dependent nature of LLMs' capabilities and highlight the importance of recognizing their individual strengths and weaknesses for optimal model guidance and enhanced problem-solving.

F Prompts Design

Table 23: The evaluation prompt for tool order planning.

```
You are a strategy model. Given a problem and a set of tools, you need to
↪  generate a sequence of tools to determine the solution to the problem.

Each tool in the toolset is defined as follows:
SQL Generator: Given an input problem and a database, it creates a
↪  syntactically correct SQLite query statement.
Python Generator: Given an input problem and some information, it generates
↪  a syntactically correct Python code snippet.

Please use the following format:
Question: This is the original question.
Error: This is the previously generated error output.
Tool: These are the tools to be selected and the order in which they are
↪  called. Please note to generate a Tool different from the Error.
Result: The final result output by the tool.

Here are some examples of mapping problems to tools:

Question: What is the square of the number of albums by Jolin Tsai?
Error: None
Tool: ["SQL Generator", "Python Generator"]
Result: 100

Question: First, calculate the square of 40, denoted as A, and then find
↪  the names of all the singers whose total number of fans is less than A.
Error: None
Tool: ["Python Generator", "SQL Generator"]
Result: ['Jolin Tsai']

Let's get started:

Question: {question}
Error: {error}
Tool:
```

Table 24: The evaluation prompt for tool order and subtask description planning.

```
You are a strategy model. Given a problem and a set of tools, you need to
↪  generate a sequence of tools to determine the solution to the problem.

Each tool in the toolset is defined as follows:
SQL Generator: Given an input problem and a database, it creates a
↪  syntactically correct SQLite query statement.
Python Generator: Given an input problem and some information, it generates
↪  a syntactically correct Python code snippet.

Please use the following format:
Question: This is the original question.
Error: This is the previously generated error output.
Tool: These are the tools to be selected and the order in which they are
↪  called. Please note to generate a Tool different from the Error.
Query: This is the sub-problem derived from the original question that
↪  needs to be input when calling the tool. Please note to generate a
↪  Query different from the Error.
Result: The final result output by the tool.

Here are some examples of mapping problems to tools:

Question: What is the square of the number of albums by Jolin Tsai?
Error: None
Tool: ["SQL Generator", "Python Generator"]
Query: ["What is the number of albums by Jolin Tsai?", "What is the square
↪  of the number of albums by Jolin Tsai?"]
Result: 100

Question: First, calculate the square of 40, denoted as A, and then find
↪  the names of all the singers whose total number of fans is less than A.
Error: None
Tool: ["Python Generator", "SQL Generator"]
Query: ["A is the square of 40, what is the value of A?", "What are the
↪  names of all the singers whose total number of fans is less than A?"]
Result: ['Jolin Tsai']

Let's get started:

Question: {question}
Error: {error}
Tool: {tools}
Query:
```

Table 25: The evaluation prompt for one-step tool-subtask pair planning.

```
You are a strategy model. Given a problem and a set of tools, you need to
↪  generate a sequence of tools to determine the solution to the problem.

Each tool in the toolset is defined as follows:
SQL Generator: Given an input problem and a database, it creates a
↪  syntactically correct SQLite query statement.
Python Generator: Given an input problem and some information, it generates
↪  a syntactically correct Python code snippet.

Please use the following format:

Question: This is the original question
Error: This is the previously generated error output
Tasks: This is a list in Python. Each item in the list is a dictionary. The
↪  key of the dictionary represents the selected Tool, and the value is
↪  the Query when calling the tool. Please note to generate a Tool and
↪  Query different from the Error.
Answer: The final answer

Here are some examples of mapping problems to tools:

Question: What is the square of the number of albums by Jolin Tsai?
Error: None
Tasks: [{{"SQL Generator": "What is the number of albums by Jolin Tsai?"}},
↪  {{"Python Generator": "What is the square of the number of albums by
↪  Jolin Tsai?"}}]
Answer: The square of the number of albums by Jolin Tsai is 100

Question: First, calculate the square of 40, denoted as A, and then find
↪  the names of all the singers whose total number of fans is less than A.
Error: None
Tasks: [{{"Python Generator": "A is the square of 40, what is the value of
↪  A?"}}, {{"SQL Generator": "What are the names of all the singers whose
↪  total number of fans is less than A?"}}]
Answer: Jolin Tsai

You must note that: The generated Tasks must strictly meet the format
↪  requirements: it must be a list in Python, each item in the list is a
↪  dictionary, the key of the dictionary represents the selected Tool, and
↪  the value is the Query when calling the tool.

Let's get started:

Question: {question}
Error: {error}
Tasks: """
```

Table 26: The prompt added to Table 25 for tool-subtask pair planning with other unrelated tools.

```
Each tool in the toolset is defined as follows:
SQL Generator: Given an input problem and a database, it creates a
↪  syntactically correct SQLite query statement.
Python Generator: Given an input problem and some information, it generates
↪  a syntactically correct Python code snippet.
Weather Query Tool: Given a location, it outputs the real-time weather of
↪  that location.
Image Generator: Given a text description, it generates a related image.
Text Extractor: Given a link to an image, it extracts the corresponding
↪  text and its position coordinates.
Translator: Given a piece of text, it translates it into other languages.
Bing Searcher: Given a piece of text, it conducts a search in the Bing
↪  browser and returns the content.
Shell Generator: Given an input problem and some information, it generates
↪  a syntactically correct Shell script.
Java Generator: Given an input problem and some information, it generates a
↪  syntactically correct Java code snippet.
Wikipedia Searcher: Given a piece of text, it conducts a search in
↪  Wikipedia and returns the content.
Office Suite: Given a text description, it automatically generates the
↪  corresponding long document, table, or PPT.
Movie Player: Given a movie name, it automatically plays the corresponding
↪  movie resource.
```

Table 27: The prompt for the tool-subtask pair generation with TPTU-SA.

```
You are a strategic model. Given a problem and a set of tools, you need to generate
↪  the next tool to be called and the corresponding subtask.

Each tool in the toolset is defined as follows:
SQL Generator: Given an input question and a database, it creates a syntactically
↪  correct SQLite query statement.
PythonREPL: Given an input question and some information, it generates a segment of
↪  syntactically correct Python code.

Please use the following format:

Question: This is the question
History: This is the history of previously generated sub-problems; if it's empty, it
↪  means there are no historical information currently
Tool_Query: This is a dictionary in Python, where the key represents the chosen
↪  Tool, and the value is the query input when invoking the Tool.
Result: This is the output result of the current Tool_Query Tool
...
History: This is the history of all previously generated sub-problems
Tool_Query: 'None' signifies that the Final_Answer can be derived
Result: 'None' signifies that the Final_Answer can be derived
Final_Answer: This is the final answer; when the history is sufficient to reason out
↪  the answer, provide the Final_Answer directly

In the above format, ... signifies that (History/Tool_Query/Result) can be repeated
↪  N times.
When you can get the Final_Answer, you can generate an empty Tool_Query and Result,
↪  and provide the Final_Answer
Please stop after generating the Result line or the Final_Answer line.

Below are some examples:

Question: First calculate the square of 40 as A, and find the names of all singers
↪  whose total fan count is less than A.
History:
Tool_Query:{{"PythonREPL": "A is the square of 40, what is the value of A?"}}
Result:1600
History: The Tool_Query for the first tool execution was:{{"PythonREPL": "A is the
↪  square of 40, what is the value of A?"}}, Result:1600
Tool_Query:{{"SQL Generator": "Find the names of all singers whose total fan count
↪  is less than A"}}
Result: Jolin Tsai

History: The Tool_Query for the first tool execution was: {{"PythonREPL": "A is the
↪  square of 40, what is the value of A?"}}, Result: 1600
        The Tool_Query for the second tool execution was: {{"SQL Generator": "Find
        ↪  the names of all singers whose total fan count is less than A"}},
        ↪  Result: Jolin Tsai
Tool_Query:None
Result:None
Final_Answer: Jolin Tsai


Note: The generated Tool_Query must strictly comply with the format requirements,
↪  and only one Tool_Query can be generated each time. Do not perform additional
↪  problem analysis, strictly adhere to the format of the problem, and generate
↪  output similar to the examples.

Now let's get started:
Question: {question}
History: {history}
Tool_Query:
```

Table 28: The evaluation prompt for simple SQL questions.

```
You are an SQLite expert. Given an input question, first, generate a
↪  grammatically correct SQLite query to execute. Then examine the query
↪  results and provide an answer to the input question.
Unless a specific number of examples to retrieve is specified in the
↪  question, use the LIMIT clause to query for a maximum of 5 results.
Do not query all columns in the table. You must only query the columns
↪  necessary to answer the question.
Please only use the column names you can see in the table below. Be careful
↪  not to query columns that do not exist. Additionally, be aware of which
↪  column is in which table.

Please use the following format:

Question: This is the question.
SQLQuery: The SQL query to be executed.
SQLResult: The result of the SQL query execution.
Answer: The final answer.

Note to only use the tables below:

CREATE TABLE Person (\n\t id TEXT, \n\t name TEXT, \n\t age INTEGER, \n\t
↪  sex TEXT, \n\t school TEXT, \n\t phone TEXT, \n\t qualifications TEXT,
↪  \n\t ability TEXT\n)\n\n/*\n 3 rows from person table:\n id \t name \t
↪  age \t sex \t school \t phone \t qualifications \t ability \n 01 \t
↪  Wang Min \t 32 \t Female \t Beijing University of Technology \t
↪  13938493271 \t Undergraduate \t Tourism Industry-related Work \n 02 \t
↪  Li Liang \t 27 \t Male \t Beijing University of Technology \t
↪  13812764851 \t Master \t Internet Company Operations \n 03 \t Zhang
↪  Jing \t 50 \t Female \t Wuhan University of Technology \t 13764592384
↪  \t Master \t Editor of Publishing House \n*/\n\n\n
CREATE TABLE School (\n\t id TEXT, \n\t name TEXT, \n\t info\_985 TEXT,
↪  \n\t info\_211 TEXT\n)\n\n/*\n 3 rows from school table:\n id \t name
↪  \t info\_985 \t info\_211 \n 01 \t Central South University \t yes \t
↪  yes \n 02 \t Shandong University \t yes \t yes \n 03 \t Tsinghua
↪  University \t yes \t yes\n*/

Question: What is the average age of the people?
SQLQuery:
```

Table 29: The evaluation prompt for complex nested SQL questions.

```
You are an SQL expert. Given an input question, you need to create a
↪  syntactically correct SQL query statement. Please only use the
↪  following datasets, which include four table names: GoldenMelodyAward,
↪  Singers, AwardNominee, Singers, and RecordCompanies. The column names
↪  and types of each table can be obtained from the create commands in the
↪  table below:

CREATE TABLE GoldenMelodyAward (\n\t Nominated\_Count INTEGER, \n\t
↪  Competing\_Count INTEGER, \n\t Awards\_Count INTEGER,  \n\t Award\_Name
↪  TEXT, \n\t Host TEXT, \n\t Year TIME \n) \n\n\n
CREATE TABLE AwardNominees (\n\t Singer_ID INTEGER, \n\t Nominated\_Work
↪  TEXT, \n\t Award\_Name TEXT, \n\t Award_Edition_ID INTEGER \n) \n\n\n
CREATE TABLE Singers(\n\t Name TEXT, \n\t Song\_Count INTEGER, \n\t
↪  Album\_Count INTEGER, \n\t Fan\_Count INTEGER, \n\t Singer\_ID INTEGER,
↪  \n\t Gender TEXT \n) \n\n\n
CREATE TABLE RecordCompanies (\n\t Record\_Company TEXT, \n\t Singer\_Date
↪  TIME, \n\t Singer_ID INTEGER \n) \n\n\n

You can query one or more tables at the same time. Be careful not to query
↪  non-existent table names or column names. Also, please note which
↪  column is in which table.

Please use the following format when answering:

Question: This is the question
Answer: The SQL query statement to be executed
```

Table 30: The evaluation CoT-based prompt for complex nested SQL questions.

```
You are an SQL expert. Given an input question, you need to create a
↪  syntactically correct SQL query statement. Please only use the
↪  following datasets, which include four table names: GoldenMelodyAward,
↪  Singers, AwardNominee, Singers, and RecordCompanie. The column names
↪  and types of each table can be obtained from the create commands in the
↪  table below:


CREATE TABLE GoldenMelodyAward (\n\t Nominated\_Count INTEGER, \n\t
↪  Competing\_Count INTEGER, \n\t Awards\_Count INTEGER,  \n\t Award\_Name
↪  TEXT, \n\t Host TEXT, \n\t Year TIME \n) \n\n\n
CREATE TABLE AwardNominees (\n\t Singer_ID INTEGER, \n\t Nominated\_Work
↪  TEXT, \n\t Award\_Name TEXT, \n\t Award_Edition_ID INTEGER \n) \n\n\n
CREATE TABLE Singers(\n\t Name TEXT, \n\t Song\_Count INTEGER, \n\t
↪  Album\_Count INTEGER, \n\t Fan\_Count INTEGER, \n\t Singer\_ID INTEGER,
↪  \n\t Gender TEXT \n) \n\n\n
CREATE TABLE RecordCompanies (\n\t Record\_Company TEXT, \n\t Singer\_Date
↪  TIME, \n\t Singer_ID INTEGER \n) \n\n\n


You can query one or more tables at the same time. Be careful not to query
↪  non-existent table names or column names. Also, please note which
↪  column is in which table.
Please note that you are not proficient in nested SQL, when encountering
↪  complex problems, you can think step by step to generate multiple
↪  non-nested SQL statements. For example:

Question: Some minor languages are used by no more than 3 countries, what
↪  are the source countries of these languages?
Thought: First generate the 1st SQL ``select Official_Language from Country
↪  group by Official_Language having count(*) > 3'', and assume that the
↪  result of this SQL is result1, then generate the 2nd SQL ``select Name
↪  from Country where Official_Language not in result1''.
Answer: select Name from Country where Official_Language not in ( select
↪  Official_Language from Country group by Official_Language having
↪  count(*) > 3 )

Please use the following format when answering:

Question: This is the question
Thought: This is the thought process
Answer: This is the final SQL query statement
```

Table 31: The evaluation prompt for mathematical questions.

```
Transform a math problem into a solution function that can be executed
↪  using Python`s math library. Use the output of running this code to
↪  answer the question.

Please use the following format:

History: Information output from previous tool invocation
Question: A question about mathematics
Error: This is the error output previously generated
PythonSolution: A Python solution, make sure to generate a PythonSolution
↪  different from the one in Error, for example,
## Python Solution
def solution():
  Python statement
Answer: The final answer


Below are some demonstrations of mapping math problems to PythonSolution:

History: The original question was: What is 37593 * 67?
Question: What is 37593 * 67?
Error: None
PythonSolution:
## Python Solution
def solution():
  import math
  return 37593 * 67
Answer: 2518731


History: The original question was: What is the 1/5th power of 37593?
Question: What is the 1/5th power of 37593?
Error: None
PythonSolution:
## Python Solution
def solution():
  import math
  return 37593 ** 1/5
Answer: 8.222831614237718


History: The original question was: What is the logarithm of 5 with base
↪  10?
Question: What is the logarithm of 5 with base 10?
Error: None
PythonSolution:
## Python Solution
def solution():
  import math
  return math.log(5, 10)
Answer: 0.69897

Now let's get started:

History:{history}
Question: {question}
Error:{error}
PythonSolution:
```

Table 32: The system prompt for one-step agent.

```
You are a strategy model and given a problem and a set of tools, you need
↪  to generate a sequence of executable tools to determine the solution to
↪  the problem.

Each tool in the toolset is defined as follows:
SQL Generator: Given an input problem and a database, create a
↪  syntactically correct SQLite query statement.
PythonREPL: Given an input problem and some information, generate a
↪  syntactically correct Python code.

Please use the following format:

Question: Here is the question
Error: Here is the previously generated error output
Tasks: Here is a Python List type, where each item in the List is a
↪  dictionary. The key of the dictionary represents the selected tool, and
↪  the value is the query input when calling the tool. Please note that
↪  the generated Tool and Query should be different from those in the
↪  Error.
Answer: The final answer

Here are some examples mapping the question to the tools:

Question: What is the square of the number of albums by Jolin Tsai?
Error: None
Tasks: [{{SQL Generator: "What is the number of albums by Jolin Tsai?"}},
↪  {{PythonREPL: "What is the square of the number of albums by Jolin
↪  Tsai?"}}]
Answer: The square of the number of albums by Jolin Tsai is 100

Question: First, calculate the square of 40 and denote it as A. Then, find
↪  the names of all artists with a total number of fans less than A.
Error: None
Tasks: [{{PythonREPL: "Let A be the square of 40. What is the value of
↪  A?"}}, {{SQL Generator: "Find the names of all artists with a total
↪  number of fans less than A"}}]
Answer: Jolin Tsai

Note that you must ensure that the generated Tasks strictly adhere to the
↪  format requirements: they must be in Python List type, where each item
↪  is a dictionary. The key of the dictionary represents the selected tool,
↪  and the value is the query input when calling the tool.

Now, let's proceed:

Question: {question}
Error: {error}
Tasks:
```

Table 33: The system prompt for the sequential agent.

```
Answer the following questions as best you can. You have access to the
↪  following tools:
Use the following format:


Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
ActionInput: the input to the action
Observation: the result of the action which should not be generate
...
Thought: I now know the final answer
Final Answer: the final answer to the original input question


... in the above format means that this
↪  Thought/Action/ActionInput/Observation can repeat N times.
The line of Observation will be given through the input.
Please stop to chat after you generate the line ActionInput or the line of
↪  Final Ansewer.

For example, when I ask what is the 0.4 power of 24, you should use the
↪  following format:

<bot>:
Question: What is the 0.4 power of 24?
Thoutht: I need to calculate the 0.4 power of 24
Action: Python REPL
ActionInput: print(24**0.4)
Observation: 3.565204915932007
Thought: I now know the final answer
Final Answer: 3.565204915932007


Begin!

<bot>:
Question: {input}
Thought:{agent_scratchpad}
```