PharmaSUG China 2024 - Paper AP-10006

Trial Agent: An Innovative and Intelligent LLM based Agent for Autonomous Clinical Trial Data Analysis

Yu Peng, Michael Wang, SANOFI

ABSTRACT

In this paper, we present the design and implementation of a groundbreaking Trial Agent, tailored specifically for the clinical trial domain, which leverages the LangChain framework and harnesses the power of GPT-4 as its underlying large language model (LLM). This innovative agent is a first-of-its-kind application in the clinical trial domain, conceived from the perspective of a statistical programmer to autonomously tackle complex analytical tasks.

The Trial Agent demonstrates the ability to autonomously decompose tasks, reason, plan, and execute actions by leveraging provided tools. It represents a significant advancement in the clinical trial arena by embedding LLM functionalities into the workflow, which paves the way for a potential autonomous analysis robot. This Agent emerges as a powerful assistant to statisticians, medical professionals, and statistical programmers, collaboratively enhancing their capacity for in-depth data analysis.

For a live demonstration, visit Tiramisu2023/Trial_Agent (github.com)**.**

BACKGROUND

This section serves to supplement some foundational AI knowledge for analysts without prior experience in AI programming. If you are already familiar with these concepts, feel free to skip ahead to the INTRODUCTION section.

Large Language Models (LLMs)

Large language models are neural network architectures specifically designed to understand and generate human language. They are trained on vast amounts of data from diverse sources. During the training process, the models learn the statistical relationships between words, phrases, and sentences. They do this by adjusting the weights of their numerous parameters (which can number in the billions) to minimize prediction errors across the entire dataset. This training phase is computationally intensive and requires significant resources, including powerful GPUs and enormous computational power. Once trained, these models can perform various natural language processing tasks, including but not limited to (*in the scenario of programmers*):

- **Text Generation:** Writing comments and documentations for your codes, improving readability and maintainability.
- **Question Answering:** Providing information, rules, or explanations from PROTOCOL in response to project specific questions.
- **Translation Services:** Converting MedDRA standards from English to Chinese.

- **Text Summarization:** Extracting statistical analysis methods from SAP and summarizing as rules for ADaM variables.
- **Code Generation:** Automatically producing SAS/R code snippets based on metadata.

Prompts

Prompts are the input texts used to guide responses from large language models when interacting with them. They can be a simple question, a complex instruction, or a piece of background information. By crafting prompts carefully, users can steer the output to better align with their expectations. Unlike traditional programming paradigms where explicit instructions are written by coders, prompts in the realm of LLMs enable a more conversational and intuitive method of directing machines to perform complex tasks.

A good prompt provides clear, unambiguous instructions and context, making it easier for the LLM to understand what is required and generate the most appropriate response. When using Application Programming Interfaces (APIs) to communicate with LLMs within code, the importance of precision and clarity becomes even more crucial. Clear prompts ensure that the model understands the request the first time, leading to faster and more efficient responses.

Here's an example of a well-crafted prompt to ask for a code to read SAS dataset in R:

I need an R script that reads a SAS dataset (.sas7bdat file). The script should handle any missing values gracefully and provide a summary of the data after reading it. Please include comments explaining the steps taken in the script.

And a not so good prompt:

Can you write me some code? I want to read a SAS file with R.

Agents

Agents refer to automated systems that can understand, interpret, and execute natural language instructions independently, such as automatically generating analytical pipelines to statistical problems, creating ADaM scripts, or debugging. They are python scripts whose inputs are tasks described in natural language by users. At their cores, they call LLM through API to break down tasks into fundamental requirements and objectives, select the most suitable tools or functions. These tools can range from basic arithmetic operations to complex data manipulation functions, machine learning algorithms, or even external APIs. The selection process is guided by the specifics of the task as interpreted by the LLM and prompts. Once the appropriate tools are identified, the Agents arrange them into a pipeline—a sequence of steps that together form a coherent solution to the task.

Contrast with Traditional Programming

Traditional programming methods typically involve detailed sets of instructions where programmers must define every step of a program's operation explicitly. Programmers write code using strict syntax and commands for statistical analysis, data manipulation, and model building. This often requires deep familiarity with programming languages such as R, or SAS, and statistical packages.

However, with LLM-Powered Agent Systems, users can describe their tasks or problems in natural language. The Agent system, equipped with an LLM, interprets these instructions and generates code accordingly. This makes programming more accessible to those less familiar with coding syntax. These systems can generate code that is optimized for performance and efficiency based on the task description. They may also suggest modifications or improvements to existing code to better fit the statistical analysis requirements. Agent systems can adapt dynamically to changes in data or task requirements. They can adjust the code and analytical methods on the fly, making the statistical analysis more responsive to evolving conditions.

INTRODUCTION

The pursuit of artificial general intelligence (AGI) has long been a primary objective in the field of artificial intelligence [1], with autonomous agents serving as a promising route towards achieving this goal. These agents are designed to accomplish tasks through self-directed planning and actions. In recent years, the emergence of Large Language Models (LLMs) has revolutionized the capabilities of AI assistants, extending their functionalities far beyond simple text generation. These intelligent bots, known as LLM Agents, possess the ability to chat, complete tasks, reason, and even take initiative, thanks to their unique architecture.

The training process of LLM Agents involves the use of carefully crafted prompts that define their personality, areas of expertise, and the guidelines they should adhere to. This ensures that they provide appropriate and accurate responses. One of their key strengths lies in their adaptability, allowing them to range from following commands to proactively assisting with tasks. They can serve as digital partners capable of handling anything from casual conversation to managing intricate workflows. To empower them for independent work, LLM Agents are equipped with vast information access, memory capabilities, and problem-solving skills. Through well-designed prompts, they become adept at planning, task execution, learning from missteps, and continuous improvement, resembling miniature, digital colleagues who require some guidance but are remarkably capable on their own.

In this paper, we introduce a groundbreaking development in the field of AI: an intelligent agent specifically designed for the pharmaceutical domain, leveraging the power of a large language model. This agent, developed using LangChain's implementation of AutoGPT and based on the GPT-4 model, represents a pioneering effort in navigating the complexities of pharmaceutical data and applications. It marks the first instance of an LLM-based agent that comprehends pharmaceutical-specific queries and tasks, and autonomously applies provided tools to execute solutions.

The introduced agent is specifically applied to the field of clinical trials, where it can independently think, understand, reason, plan, and analyze clinical trial-related documents (such as PROTOCOL, SAP, and METADATA) and data (including SAS datasets) to complete tasks proposed by users. By leveraging the vast information access, memory capabilities, and problem-solving skills of LLM Agents, this intelligent agent holds the potential to significantly improve the efficiency and accuracy of data analysis processes in the clinical trial domain.

TRIAL AGENT ARCHITECTURE

Agents usually consist of 5 components,

- Profile (system messages or system prompts), defining their system roles or background field they lie in. For example, if you define your agent in the field of statistical programming, it will recognize ADaM as a CDISC model rather than the Adam (Adaptive Moment Estimation) optimization algorithms famous in the field of NLP. And it will treat CRF as Case Report Form rather than Conditional Random Field, also a well-known probabilistic model in NLP.

- Planner (human-like way of decision making or reasoning), the fundamental to the autonomy and intelligence of an agent. This module helps the agent to create plans, strategies or courses of action when given tasks. The capability owes much to innovative techniques such as Few-Shot learning and Least to Most prompting strategy.

- Toolkits (customized functions), extending an agent's core competencies. These tools, essentially self-defined functions, act as extensions of the agent's capabilities, allowing it to engage with its environment in more sophisticated and nuanced ways. The Agent calling customized functions is like we human beings select proper macro functions to create TLGs. The only difference is that our selections are driven by personal familiarity with the code and the specific requirements of the project. However, the agent's choice of functions is entirely predicted by functional description of the code and the parameters associated with it.

- Memory (historical messages among user, LLM, tool, and profile), controlling the information transferred. LLMs have fixed context window size, which is becoming longer and longer with evolution but is still short compared with huge amount of domain information, long and complex scripts, and multi-round conversations. Therefore, Memory module conditionally stores knowledge, controls long-term and short-term messages, dynamically constructs information into proper prompts. This, in turn, leads to more personalized, relevant, and coherent responses, significantly improving the effectiveness of communication.

- Action (execution within the environment), translating the decisions made by the intelligence of the LLM into executable operations within the environment. LLM's input and output are all strings. They do not literally run the code they generate but give user a character block of code. Action module extracts the code from the response of LLM, prepares the environment that are necessary (like R libraries, Python packages), and finally run the code to perform the analysis.

At the core of agents, they basically work by having an LLM pick what to do step by step. They use the LLM like a smart guide to figure out actions and the order they should happen in. LLM's knowledge is set in the field described in Profile; LLM's capability of resolving problems is activated through Planner; LLM's chat history is controlled by Memory; LLM's function calling ability is extended by Toolkits; and LLM's step to perform by Action.

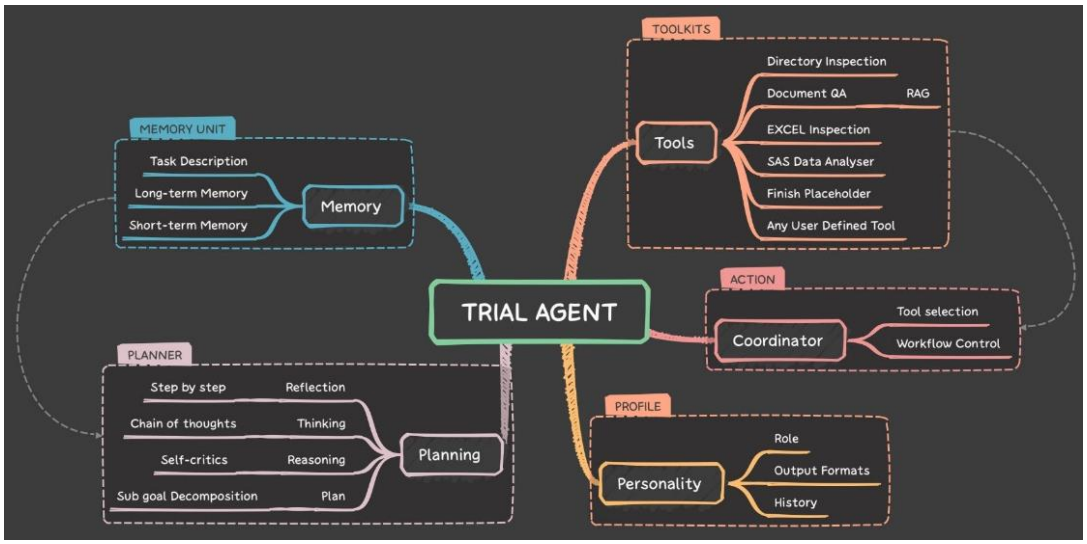

Figure 1 Trial Agent Architecture

PROFILE

As illustrated in the diagram [**Figure 1**], our primary step involves delineating the Trial Agent's role and functions within its designated clinical trial domain. This entails the agent as an expert in its field, equipping it with a distinct persona and establishing guideline generation protocols via an introductory prompt. In this initial version, we maintain a broader scope. We exclude intricate, domain-specific details such as CDISC SDTM/ADaM domain standards or variable nomenclature standards into the prompt. Rather, we provide a general overview of typical documents and datasets the agent might encounter, as shown below [**Figure 2**]. Remarkably, through these basic instructions, the Trial Agent is empowered to plan its workflow well when presented with a task in the domain of clinical trial.

With such system messages in background, Trial Agent knows the potential files it will deal with and the functions of the files in clinical trial. When asked question by user, the agent will calculate the semantical similarity between the question and the description of these files and determine which file to consult to.

```
Description of data and reference files:
  - 1. [Complex Algorithms]/[ADaM Metadata]: Look up which dataset and which
column/parameter contains a clinical test. Should be the first resource you look
up.
  - 2. [Protocol]: Detail background and introduction of a clinical trial.
  - 3. [SAP]: Detail information about analysis methodology.
  - 4. [ADaM datasets]: sas7bdat files, the SAS dataset you will analyze.
```

Figure 2 Trial Agent Profile Prompt

PLANNER

Upon receiving a task, the Trial Agent utilizes an LLM to create a step-by-step action plan. It adopts method known as Chain-of-Thought (CoT) and Least-to-Most (LtM), where the LLM demonstrates its reasoning while tackling complex issues.

CoT breaks down complicated, multi-step tasks into smaller intermediate parts, allowing for deeper analysis and better use of resources [2], providing transparency into the model's behaviour, offering insights into how it arrived at a particular answer and the opportunity to identify and debug errors in the reasoning process. Adding CoT to large language models is straightforward with the right prompts, which is a practical method to amplify LLM reasoning capabilities in complex tasks.

Least to Most is inspired by educational methodologies that introduce concepts and tasks in a progressive manner, starting with simpler, foundational elements and gradually moving toward more complex ones. In the context of LLMs, this approach is employed to improve the model's ability to handle complex tasks by breaking them down into more manageable components. It involves 2 steps. The first one is the problem reduction step. This initial phase involves the identification and isolation of simpler aspects of a complex problem. The model is prompted to generate a series of sub-questions or sub-tasks that are derived from the original, overarching query or task. This helps to decompose the complexity of the problem into more digestible pieces. The second one is the sub-tasks solving step. Once the sub-questions or sub-tasks are identified, the LLM is tasked with addressing these in turn. The model attempts to solve each sub-task independently, which often requires less computational effort and reduces the risk of overwhelming the model with too much information at once.

Besides CoT, LtM, and direct prompting, Self-Ask Prompting emerges as a progressive way. It's an advanced method where the LLM not only breaks the main question into smaller ones but also shows its thinking path clearly. The LLM knows when it finds the final answer and moves from intermediary solutions to the ultimate answer. This helps the LLM answer questions it hasn't been directly taught to answer. Even if the model doesn't know the direct answer, it can piece together answers from parts it does know. Self-Ask Prompting serves as a guiding mechanism, assisting the LLM in synthesizing these fragments into a coherent and comprehensive response.

The key prompts that guide the Trial Agent to reason is shown below. [**Figure 3-4**] These are also system messages defined for the agent. Since agents are designed to solve generalized problems, how the issues are decomposed is from case to case. Therefore, to show the LLM how a certain type of problems is split into a series of sub problems will not work for all problems. Thus, the planner module is the key part to stimulate the emergent abilities of the LLM. These emergent abilities include advanced reasoning, creative thinking, nuanced understanding of context, and other sophisticated linguistic skills that weren't explicitly programmed but arise from the model's architecture and training on vast amounts of text data. And finally the agent is able to decompose unmet problems automatically through well-designed prompts.

```
Thinking:
Observe execution records and your self-reflection, and think step by step:
  A. Analyze the dependencies between elements, for example, if you need to
obtain the values of elements X and Y:
    i. Do you need to obtain the value/definition of X first to obtain Y
through X?
    ii. If you obtain X first, can you filter Y through X to reduce the cost of
enumerating each Y?
    iii. Are X and Y present in the same data source, and can you obtain Y at
the same time as X?
    iv. Is there a more efficient or smarter way to query a concept or element?
    v. If the last attempt to query a concept or element failed, can you try to
query it again from another resource?
  B. Based on the above analysis, prioritize the queries between sub-elements.
  C. Identify the sub-elements that need to be valued currently.
Note: Do not make any assumptions about the values/definitions of elements.
Ensure your information comes from the given data source!
```

Figure 3 Trial Agent Planner Prompt - Thinking

```
Plan:
Strictly follow the following rules to plan your current action:
  A. Detail the execution plan for the current action. Plan only one step at a
time. PLAN ONE STEP ONLY!
  B. Ensure project relative paths {work_dir} is added when operating on data
or files.
  C. Analyze step by step, including the data source, the method of operation
on the data source, and the method of data analysis. Determine any known
constants that can be directly inserted into this analysis.
  D. Do not attempt to calculate every element of a file, as this is too costly
and strictly prohibited. You can find a more effective method through analysis,
such as conditional screening.
  E. Whether the above analysis depends on the value/definition of an element
that has not yet been obtained. If so, reconstruct the current action plan to
ensure that all dependent elements values/definitions have been obtained.
  F. Do not assume the value/definition of any element. Ensure your information
comes from the given data source. Do not fabricate information. DO NOT MAKE UP
ANY INFORMATION!!!
  G. Ensure that all elements involved in your action have obtained precise
values/definitions.
  H. If all sub-tasks are completed, use the FINISH action to end the task.
```

Figure 4 Trial Agent Planner Prompt - Planning

TOOLKITS

In the management of decomposed tasks, the Trial Agent relies on a suite of customized tools. This process begins with aligning each subtask with the most fitting tool from a curated inventory. Selection is based on the degree of similarity between the subtask's requirements and the tool's capabilities, as described in their function descriptions. The agent also prepares the settings needed to run the tools, following the instructions provided.

These tool descriptions and setting guidelines become clues for the AI model. Both the tool's description and parameter specifications are converted as prompts for the LLM. Leveraging these prompts, the LLM aligns the objectives of the task and the tool, subsequently selecting the ideal tool and configuring it with the appropriate parameters.

If a tool doesn't work during execution phase, the LLM goes back to choosing another tool or changing parameter settings. The iterative retry strategy continues until the task is successfully

completed or the maximum retry limit is hit. Such a design ensures a robust and adaptive workflow, optimizing the agent's performance in tackling complex tasks.

For the Trial Agent, we have prepared the following four tools:

1. **Directory Inspection Tool**: This tool enables the agent to list the contents of the database directories. By matching descriptions of files and data with the relevance to the task at hand, the agent will be able to select a proper file to the task. For example, when user ask about clinical trial designs, schedule of visit, or treatment information, the tool will identify the “PROTOCOL” file as the most pertinent.

2. **Document-Q&A Tool**: Essentially functioning as a Retrieval-Augmented Generation (RAG) system, this tool locates sections within documents that closely relate to the query. It combines these sections with the question to prompt an LLM, enabling the agent to address clinical trial or statistical programming inquiries, even without specialized training**.**

3. **SAS Dataset Analysis Tool**: Central to the role of a statistical programmer assistant, this tool leverages the agent's ability to analyze clinical trial datasets in response to user requirements. Given the diverse and dynamic nature of SDTM/ADaM datasets that vary with the specific clinical trial and therapeutic area, conventional programming method or rule-based systems are hard to offer a generalized solution. Instead, by crafting prompts, we guide the LLM to harness its code generation capabilities, thereby automatically generating Python code specific to the task description. This innovative approach ensures a versatile SAS dataset analysis tool. Given the diversity and variability of datasets across clinical trials and therapeutic areas, traditional coding methods prove too rigid. Our approach ensures the tool remains broadly applicable, not confined to specific SAS dataset structures like ADAE's occurrence structure or ADLB's BDS layout, nor does it presuppose knowledge of variable formats or meanings. Its versatility extends to potentially analyzing datasets generated by R applications as well.

4. **FINISH Tool**: Finally, a FINISH placeholder tool is necessarily available to the agent when it completes the task.

These tools collectively equip the Trial Agent to navigate, comprehend, and respond effectively to inquiries within the intricate realm of clinical trials, backed by adaptable data analysis capabilities.

MEMORY UNIT

Memory modules are vital components in the Trial Agent, functioning akin to a repository for the agent's internal thought processes, plans, reflections, and interactions with users.

Short-term Memory: This serves as an immediate holding space where we can temporarily store each step of the agent's reasoning, ideas, plans, and reflections during the execution of subtasks. It operates within the constraints imposed by the context window limit of the LLM.

Long-term Memory: Stored typically in databases such as VectorDB, long-term memory is reserved for archiving the agent's strategic plans, conclusions drawn from completing primary objectives, and selectively preserving only those short-term memories that are pertinent to the main

task. Additionally, it keeps historical conversations between the user and the agent, facilitating multi-turn dialogue capabilities and ensuring a continuous thread of interaction across sessions.

ACTION

Action module emerges as a pivotal component that bridges the gap between conceptual decision-making and tangible operational execution. It interprets instructions from the LLM. The Action module receives instructions or directives generated by the LLM. These instructions could be in natural language form, specifying actions to take based on the current context or goal. Once it has received these instructions, the Action module must map them to specific, actionable commands that can be executed in the real world or virtual environment. This often involves understanding the semantics of the LLM's output and converting it into a format that other systems can interpret and then executing commands.

DEMONSTRATION

Here, we illustrate through a common example how the Trial Agent received a request from user, breaks down tasks, utilizes tools, and progressively finishes minor objectives to accomplish an assignment in 4 rounds.

Our primary task is to examine if a specific subject's lab test ALT reading exceeds 90 (IU/L) [**Figure 5**]. You may see that the messages from both the agent and the user are displayed in white, the agent's reasoning process is shown in green, the results of the agent's reasoning are indicated by yellow, and the code autonomously written by the agent is also presented in white. The Agent possesses no additional contextual knowledge of clinical trials. It divides this task into two subtasks:

A. Locate the dataset and column name containing ALT data,

B. Evaluate the specified subject's ALT result against the threshold of 90.

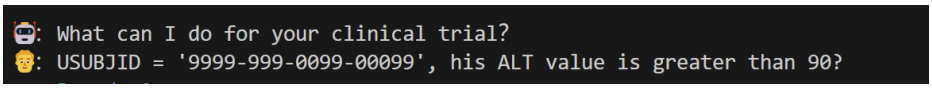

Figure 5 Trial Agent Demo Task

ROUND 0

The Agent, through self-questioning deduction, realizes that to accomplish the initial subtask of locating the dataset and column name for ALT, it first needs to identify available datasets and files. To enumerate the directory contents, the tool "Directory Inspection Tool" can be utilized—it's basically a Python script designed to list files in a given directory, accepting one parameter: the directory path. Drawing from its profile prompt, the Agent knows the analysis directory is "./data", which it feeds into "Directory Inspection Tool". Thus the Agent successfully inspects the contents of the "./data" folder. This marks the completion of the initial exploration phase [**Figure 6**].

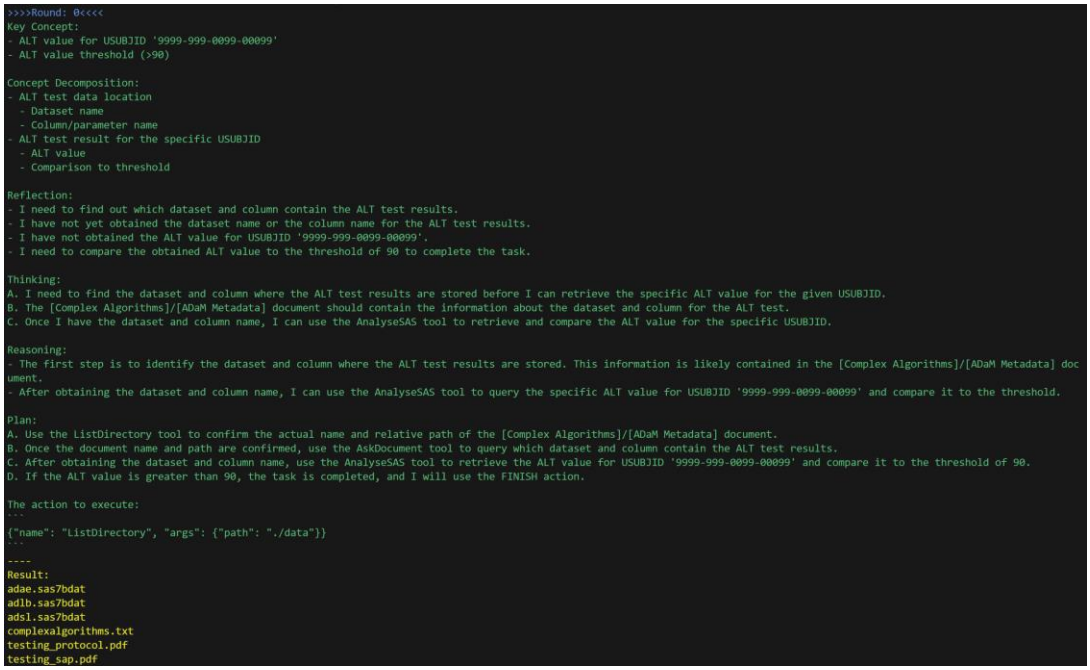

Figure 6 Trial Agent uses Directory Inspection Tool to find available resources

ROUND 1

Leveraging its extensive training corpus, the Agent, as a large language model, recognizes ALT as a LAB TEST. From the profile prompt, it deduces that metadata is necessary for acquiring clinical trial test mappings. To obtain ALT related dataset names and filenames, it infers that "complexalgorithms.txt" is the most pertinent file regarding metadata. To extract information from this document, the "Document-Q&A Tool"—a RAG-based document question-answering script that takes two parameters (filename and query)—is employed. The Agent provides the path to the metadata document and formulates a query to locate the dataset and column for ALT. Upon execution, the tool manages to identify ALT in the ADLB dataset at "./data/adlb.sas7bdat", with PARAMCD/PARAM as the relevant column names. Thus, the first subtask is achieved [**Figure 7**].

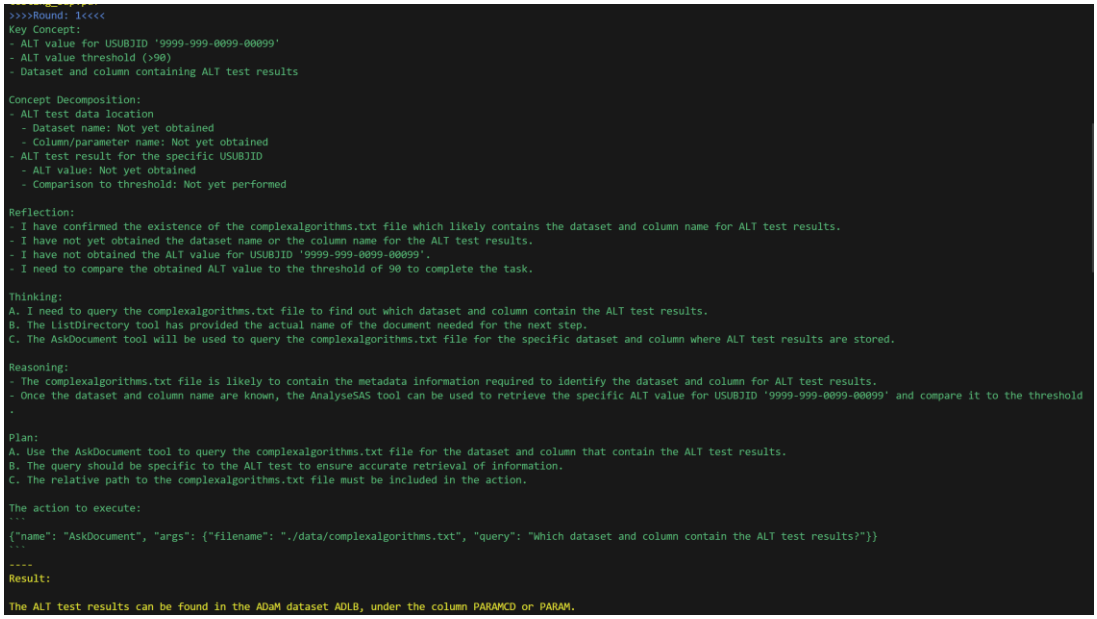

Figure 7 Trial Agent uses Document-Q&A Tool to figure out ALT is in ADLB.PARAMCD

ROUND 2

The Agent proceeds to the second subtask: obtaining ALT values and comparing them to the threshold. This necessitates analyzing the actual dataset. Recognizing the need, the Agent selects the "SAS Dataset Analysis Tool", which is a prompt guiding the LLM to generate Python code without explicit inputs. Using the description of the subtask and prior findings on variable names, the Agent crafts a Python script that filters adlb.sas7bdat to retrieve the AVAL values for records where PARAMCD equals "ALT", uncovering records where the value surpasses 90. With this, the second subtask is fulfilled. [**Figure 8-9**]

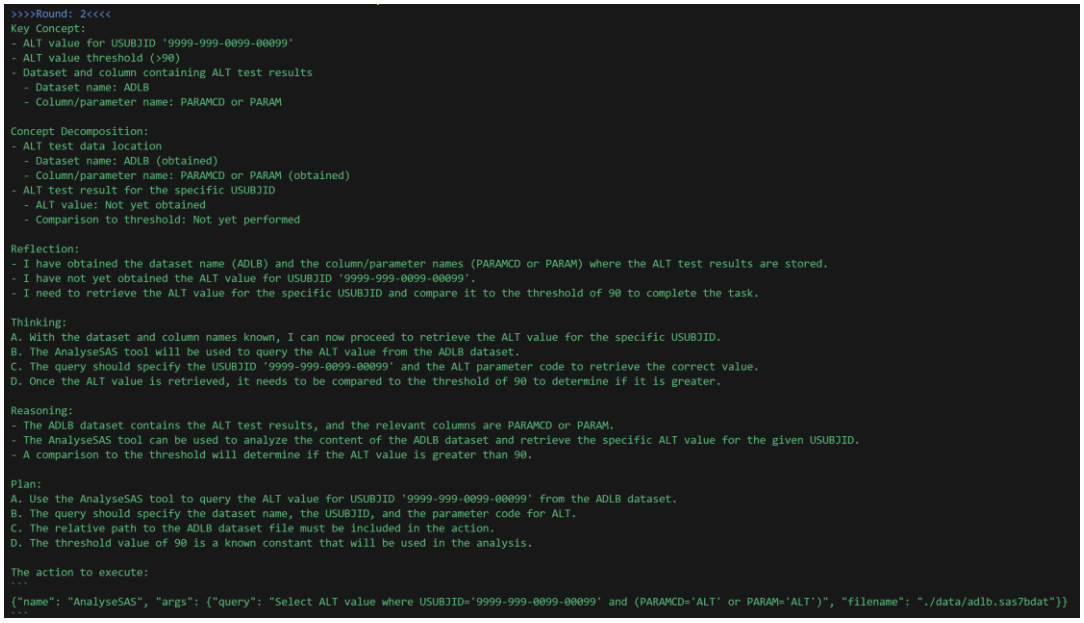

Figure 8 Trial Agent uses SAS Dataset Analysis Tool to analyze SAS datasets

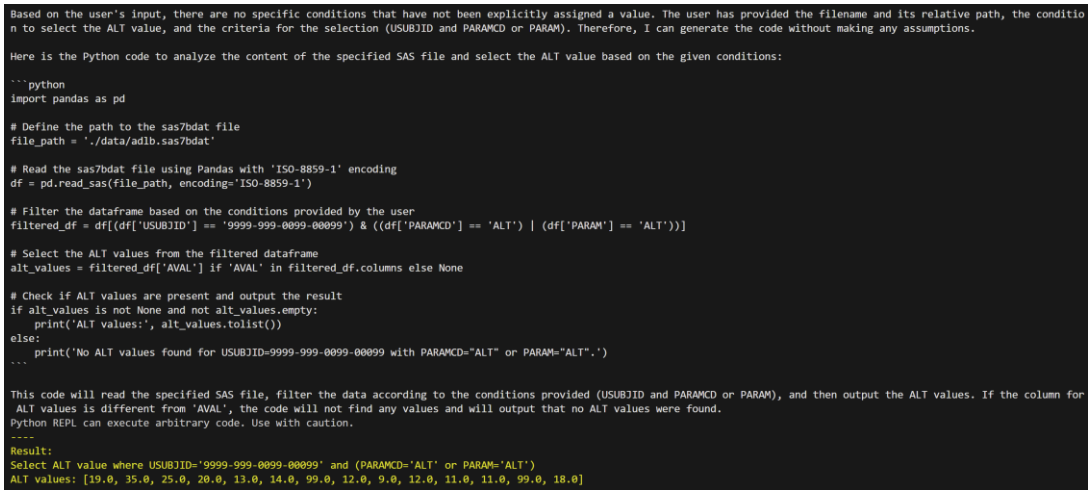

Figure 9 Trial Agent autonomously writes Python codes to analyze subject's ALT result

ROUND 3

In its final analytical step, the Agent confirms that all subtasks have been successfully executed and achieves the conclusion: evidence exists of the subject's ALT readings exceeding 90. As designed, the Agent utilizes the placeholder tool "FINISH" to stop reasoning and conclude the task [**Figure 10**].

Figure 10 Trial Agent completes main task

CONCLUSION

The TRIAL AGENT, developed through AUTO-GPT leveraging the LangChain framework and the LLM GPT-4, has demonstrated the capability to accurately utilize user-provided tools for task

decomposition, thinking, planning, and reasoning—effectively accomplishing assigned tasks without preliminary domain knowledge integration.

Future work includes several key aspects:

1. **Refining Prompt**: Incorporate precise prompts adhering to CDISC standards, detailing SDTM/ADaM structures and variable descriptions. This enhancement aims to shorten the agent's reasoning process, enabling it to complete more intricate tasks with increased efficiency and accuracy.

2. **Expanding the Toolkit**: Augment the agent's capabilities by broadening toolkit, which will facilitate the execution of a wider spectrum of tasks, further enhancing its versatility and utility.

3. **Interactive Feedback:** The incorporation of interactive feedback mechanisms enhance control and adaptation. Letting LLMs operate totally autonomously without oversight often leads to unpredictable outcomes due to the model's lack of context or misinterpretation of intent. By prioritizing interactive forms of engagement, developers can create more refined and user-centric agent tools.

4. **Domain-Specific LLM Training**: Fine-tune the LLM (such as the open source LLMs LLAMA3, QWEN2, GLM4-9B) specifically for the clinical trials domain. By doing so, the model's intrinsic adaptability to the nuances and complexities inherent in clinical trial contexts will be significantly enhanced, thereby optimizing its performance within this specialized field.

REFERENCES

[1] Wang, Lei, et al. "A survey on large language model based autonomous agents." Frontiers of Computer Science 18.6 (2024): 1-26.

[2] Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." Advances in neural information processing systems 35 (2022): 24824-24837.

[3] Yuan, Siyu, et al. "Easytool: Enhancing llm-based agents with concise tool instruction." arXiv preprint arXiv:2401.06201 (2024).

ACKNOWLEDGMENTS

Heartful thanks to I&I programming team from Sanofi, China, for their supports on the project and this paper.

CONTACT INFORMATION

Warmly welcome your comments and questions are valued and encouraged.

Contact the author at:

Yu Peng

SANOFI

Renxiang (Michael) Wang

SANOFI

Yu.Peng@sanofi.com Michael3.Wang@sanofi.com