Demo: PharmaData-Agent: A Specialized Agent for Pharmaceutical Data Analysis

Zihan Guan[1,2], **Hanyin Wang**[2†], **Zhongliang Zhou**[2], **Qiaohui Zhou**[2], **Peining Tao**[2], **Junshui Ma**[2]
[1] University of Virginia, [2] Merck & Co., Inc., Biometrics Research, Rahway, NJ, USA.
[†] Corresponding Author
bxv6gs@virginia.edu
{Hanyin.Wang, Zhongliang.Zhou, Qiaohui.Zhou, Peining.Tao1, junshui_ma}@merck.com

Abstract

Large language models (LLMs)-based agents are transforming many fields due to their great potential in automating complex tasks. Despite great success, their application in pharmaceutical data analysis remains hindered by coarse-grained workflows and potential data safety risks. General-purpose agents often produce unreliable results and operate with high-risk tools, failing to meet the rigorous demands of data analysis in the pharmaceutical domain. To bridge this gap, we developed **PharmaData-Agent**, an agent designed specifically for pharmaceutical statisticians. PharmaData-Agent automates the data analysis pipeline through a four-step workflow: data parsing, data understanding, data analysis, and data visualization. Furthermore, by orchestrating sophisticated planning and reflection with specialized subagents, PharmaData-Agent transforms complex tasks into a transparent, multistep process. Empirical evaluations show that PharmaData-Agent achieves perfect tool-calling performance and 95.29% accuracy on DaBench, highlighting its potential to empower statisticians with reliable analytical capability.

1 Introduction

The pharmaceutical industry is undergoing a profound data-driven transformation, as clinical trials, real-world evidence, and multi-omics platforms now generate vast, high-dimensional datasets indispensable for drug development and regulatory submissions [3, 16]. However, current data analysis workflows rely heavily on manual coding, with repetitive tasks such as data cleaning and descriptive statistics. These processes are time-consuming and resource-intensive, especially under tight deadlines [2, 12, 14].

To address the limitation, there has been growing interest in developing large language model (LLM)-based agents to augment biomedical practice, which has been proven effective across various domains such as biology [13, 1, 4] and medicine [18]. Motivated by the great success, we propose **PharmaData-Agent** - a specialized agentic AI system designed to assist statisticians by reducing their workload in pharmaceutical data processing and analysis.

Despite prior initiatives in general-purpose agents (e.g., II-Agent [7] and Manus) and data analysis-specialized agents [11, 6, 5], these approaches cannot be directly applied to pharmaceutical data analysis for three main reasons: ❶ Existing systems rely on coarse-grained workflows and generic toolboxes that have not been calibrated for statistics, leading to outputs that fall short of professional requirements in the pharmaceutical domain. For example, pharmaceutical data analysis emphasizes efficacy and safety evaluations, where metrics such as the objective response rate (ORR) and Kaplan–Meier (KM) curves are standard. In contrast, the existing data analysis-specialized agents [11, 6] provide tools tailored mainly for machine learning–oriented tasks. ❷ Given the high confidentiality

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: The Second Workshop on GenAI for Health: Potential, Trust, and Policy Compliance.

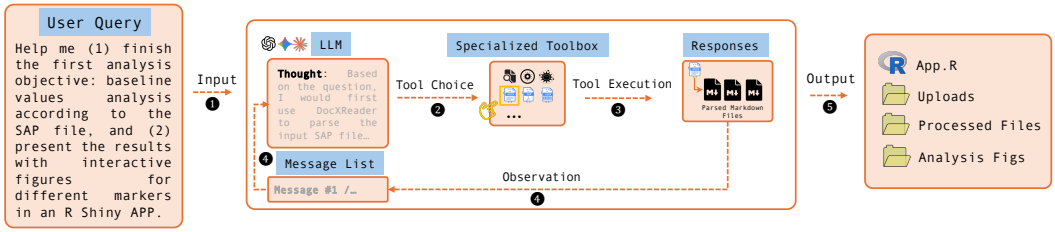


Figure 1: Overview of the PharmaData-Agent system.

of pharmaceutical data, agentic AI systems must operate locally, using proprietary model APIs or fully open-source LLMs. Consequently, public agentic services such as Manus cannot be adopted in this domain. ❸ Data safety also requires that the toolbox itself be secure. However, some agentic AI systems—such as II-Agent [7] — execute code directly in the global environment, which introduces significant security risks. A more detailed related work section is provided in the Appendix A.

To bridge this gap, we aim to equip PharmaData-Agent with a specialized toolbox that follows the life cycle of pharmaceutical data analysis and incorporates safety considerations directly into tool design. Specifically, our system includes dedicated parsing tools for different data types, an understanding module for extracting insights from the data, analytic modules for statistical evaluation, and visualization tools for result presentation. To orchestrate these specialized tools, we carefully design a unified system prompt, along with an extended context window to improve multi-turn conversational capabilities. We evaluate the system through unit testing and benchmarking on a public dataset. Empirical evaluations show that PharmaData-Agent achieves perfect tool-calling performance and 95.29% accuracy on DaBench [6], highlighting its potential to empower statisticians with reliable analytical capability.

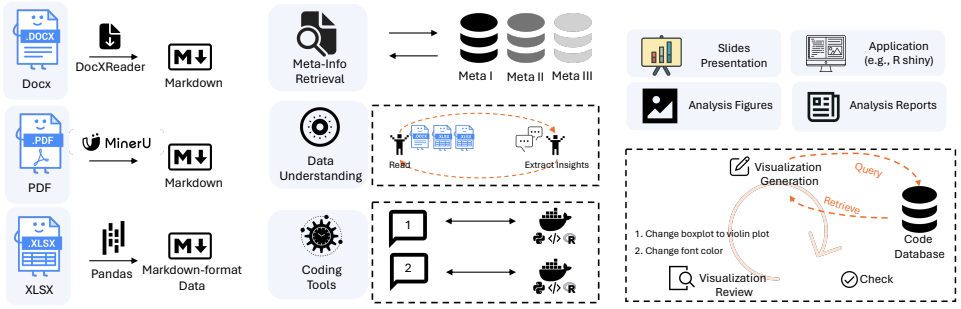
2 System Designs

In this section, we first provide an overview of the whole agentic AI system, describing the intuitions for our system design; Then, we provide detailed descriptions of the specialized toolbox, system prompts, and context window management we designed for the agent.

2.1 Overview

Framework. The PharmaData-Agent (Figure 1) is based on the ReAct [17] framework. Upon receiving a user query, the LLM agent first generates reasoning traces and determines which tool to call (Step ❶), based on the system prompt and the history message. The selected tool and its parameters are then parsed (Step ❷). After execution (Step ❸), the response is observed and appended to the message list (Step ❹). Through iterative reasoning, the agent eventually decides when to terminate the loop and return the final result to the user (Step ❺). The overall performance of the system largely depends on the toolbox design and the system prompt provided to the LLM agent at each iteration, which are *two key areas of improvement explored in this paper*.

Intuitions. *How to adapt the agent to the pharmaceutical data analysis?* Recall that the pharmaceutical data analysis usually incorporates three stages (Figure 4): **data parsing, data analysis, and visualizations**. In the data parsing stage, statisticians are expected to read the statistical analysis plan (SAP) and the data for this clinical trial. In the data analysis stage, they usually use public information to understand the data (e.g., ADaM metadata), collectively understand the given files, and write code to conduct pre-processing and analysis (e.g., statistical correlation). In the last visualization stage, the results are usually presented with various tools, including slides, an R Shiny app, or simply figures. Therefore, to fully build an agentic AI system for the pharmaceutical data analysis, we propose to build specialized tools for *each component* in the data analysis life cycle (§ 2.2). The tools are then orchestrated with a unified system prompt (§ 2.3). To further enhance the multi-round conversation ability, we build a consistent context window throughout the conversation session (§ 2.4).



(a) Data parsing tools. (b) Data analysis tools. (c) Visualization Tools

Figure 2: Specialized Toolbox for PharmaData-Agent.

2.2 Specialized Toolbox for PharmaData-Agent

2.2.1 Data Parsing Tools (Figure 2a)

Data parsing is a necessary stage for the current agentic systems. Firstly, not all models support direct file uploads, even APIs such as Azure OpenAI APIs cannot take direct data file uploads for context-aware prompting; Besides, data parsing brings advantages in adaptively controlling the contents passed into the context, potentially saving unnecessary tokens.

However, parsing diverse input files into a usable format is a non-trivial effort. Real-world pharmaceutical analysis involves various data types; for example, statistical analysis plans (SAPs) are often Word documents, data files are typically in CSV or XLSX formats, and policy documents are saved as PDFs. Each format requires a tailored parsing strategy. For instance, Word and PDF files may contain complex formulas or embedded images that are not inherently friendly to LLMs. To handle this complexity effectively, we employed fine-grained data parsing tools for each data type.

DocXReader Tool. Our DocXReader transforms Microsoft Word `.docx` files into clean Markdown. By directly parsing the underlying Open XML package, it systematically unpacks the document ZIP, traverses XML nodes, and reconstructs both structure and semantics. This ensures headings, lists, equations, and tables are faithfully retained in a format friendly to LLMs.

PDFReader Tool. For PDF parsing, we leverage MinerU [15], a state-of-the-art PDF parser. MinerU extracts text, tables, and embedded structures into well-organized Markdown, smoothing away the notorious parsing irregularities of PDFs while delivering LLM-ready content.

XLSXReader Tool. Compared to Word and PDF, Excel files are inherently more structured. We use Pandas [10] to ingest `.xlsx` files, transforming sheets, tables, and cell ranges into a Markdown representation that preserves tabular integrity while remaining lightweight for downstream prompting.

2.2.2 Data Analysis Tools (Figure 2b)

The data analysis stage involves two main tasks: (1) data understanding: interpreting data and user queries, and (2) coding: writing code that generates the final results.

Meta-information Retrieval Tool. A key challenge in understanding the pharmaceutical data is the prevalent use of cryptic abbreviations in data file column names (e.g., AVAL for Analysis Value), which are often incomprehensible to LLMs without additional context. Moreover, these abbreviations may come from different standards, which makes it hard to transform them into correct descriptions uniformly. To address this, we developed a meta-information retrieval tool that actively consults a pool of curated meta-information datasets (e.g., Meta I - III in Figure 2b) that map these abbreviations to their full descriptions, enabling the agent to accurately interpret the data. Specifically, we use Python's difflib to calculate a matching score between each column name from the input files and the corresponding entries in the meta-information dataset. To avoid false positives, we set a conservative threshold of 0.8. In other words, a column name is only paired with a description if its matching score is 0.8 or higher; all potential pairs with a score below this threshold are disregarded.

Data Understanding Tool. We also found that data understanding is not a linear procedure but rather an iterative process that requires considering all input files collectively. Therefore, we formulated data understanding as a ReAct-based subagent. This framework allows the agent to reason and

act in multiple cycles, progressively building a more comprehensive understanding of the provided files. Given the complexity of the pharmaceutical data (e.g., biomarker data in a clinical trial), understanding what the data means and the analysis objective poses a non-trivial challenge. We use simple prompting techniques to force the model to collectively consider the input datasets and derive a basic understanding of the data. More details are provided in the Appendix B.1.

Coding Tools. Pharmaceutical data analysis most commonly relies on Python, R, and Bash. Therefore, we create specialized coding tools for each language. Moreover, to avoid the interference of different sessions, we create a Docker container for each session, forcing the code to be safely executed in isolated environments. Besides, we create a list of banned commands, e.g., "`sudo *`", to filter out the potentially harmful commands.

2.2.3 Visualizations Tools (Figure 2c)

We conceptualize the visualization tool as a sub-agent capable of self-improving its generated content through the creation-review strategy. For illustrative purposes, we present an R Shiny application as the primary deliverable used to describe the sub-agent. The proposed methodology can be readily generalized to other deliverables, such as slides and an analysis report.

Visualization Generation Tool. For R Shiny application creation, we adopt an RAG-style tool. Specifically, we manually collected and maintained a pool of R Shiny apps manually created by the statisticians, along with their descriptions. We use descriptions to build the indexing with the FAISS [9], using the OpenAI embedding model `text-embedding-3-large`. When creating the R Shiny APP, the user question is used to retrieve the most relevant R Shiny app codes from the RAG database, which can then be added to the history message and used as a reference for the R Shiny APP creation.

Visualization Review Tool. One-round generation always leads to imperfect results. Even with the state-of-the-art LLMs such as GPT-o1, we also observe some bugs, such as function conflicts. To this end, we use an extra LLM-as-a-judge to evaluate the generated contents and propose constructive suggestions based on the previous generations. For R Shiny apps, we paid particular focus on inspecting the correctness of the code, the address of the used datasets, and any conflicts of different libraries. The full prompt used to inspect the codes is provided in the Appendix B.2.

2.3 System Prompt

System prompts serve as the cornerstone for orchestrating specialized tools and guiding workflows. Following the widely adopted Manus-style system prompts [8], our strategy can be summarized in the following three aspects:

Workflow Registration. When building specialized agentic systems, it is crucial to explicitly present the overall workflow along with the tools required for each stage. This ensures that the agents consistently follow the predefined procedures. We therefore wrap the detailed data analysis workflow within the tag `<pipeline>...</pipeline>`.

Rules Emphasis. To ensure that the agent follows the special rules of each agent, we dedicate a special section within the system prompt to highlight them. For example, the snippet below illustrates the structure for specifying the rules of the data understanding tool, wrapped with `</data_understanding_rules>...</data_understanding_rules>`.

Error Handling. To improve robustness, we explicitly define how agents should respond when encountering errors, unexpected outputs, or missing information. By specifying fall-back actions in the system prompt, the agents are guided to recover gracefully rather than halting or producing incomplete results. We wrap the error handling rules with a special tag `</error_handling>...</error_handling>`.

2.4 Context Window

In real-world scenarios, pharmaceutical data analysis often undergoes multiple rounds of revision. For example, an end user may request changes to a specific script file. To support such requests, we propose using a separate context window to manage the files visible to the model. Generated files (e.g., R Shiny apps, analysis figures) can be directly dragged into or out of this window, allowing their

contents to be seamlessly added to or removed from the agent's context. This enables fine-grained and iterative modifications.

System Prompt

```
<pipelines>
The data analysis pipeline MUST strictly follow the steps below:
1. Data Parsing: [Description + Tools Used]
2. Data Understanding: [Description + Tools Used]
...
</pipelines>
<data_understanding_rules>
[Rule 1] [Rule 2]
</data_understanding_rules>
<error_handling>
1. Attempt to fix issues based on error messages; if unsuccessful, try
alternative methods
</error_handling>
```

3 Experimental Evaluations

For all the experiments, we use GPT-4.1 as the default LLM for the agentic system. More detailed setups are provided in the Appendix C.1.

3.1 Tool Calling Results

We first conduct a simple tool calling result to evaluate whether the specialized tools can be successfully called by the LLM agent. For each tool in the toolbox, we use a separate LLM to synthesize 20 prompts corresponding to this tool; Then we input the prompt to the agentic system to test whether the corresponding tool will be called. The experimental results show that the tools can all be successfully called with a perfect success rate of 100%. Due to the page limit, the detailed experimental results are provided in the appendix.

3.2 Benchmark Evaluations

Since no benchmark currently exists for the pharmaceutical domain, we adopt DaBench [6] to evaluate the effectiveness of our agentic system in the general data analysis domain. The benchmark consists of 257 questions spanning three difficulty levels, each with a deterministic ground-truth answer. The questions cover topics such as summary statistics, machine learning, and correlation analysis. We report mean accuracy following the evaluation pipeline of DaBench. We choose InfiAgent [6] as the baseline method, evaluating three variants with different LLM models. As shown in Figure 3, our agentic framework achieves significantly better results than the InfiAgent framework, with an average accuracy of 95.29%. In contrast, InfiAgent with GPT-4.1 as its base client achieves only 88.92%.

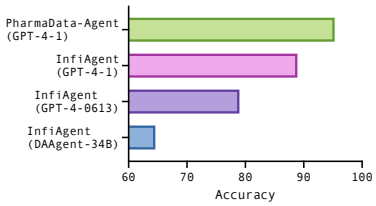


Figure 3: Evaluations on the results on DaBench [6].

3.3 Demo Demonstration

We also use a demo example from real-world statisticians to showcase the effectiveness of our system. Due to the space limit, we move the detailed demonstrations to the Appendix C.3.

4 Conclusion and Future Works

In this paper, we present a specialized agentic system for pharmaceutical data analysis. We have designed a specialized toolbox and calibrated system prompt for the system. Various experiments demonstrate the effectiveness of our system. However, there are still several directions to explore in the future, e.g., there are no existing benchmarks for the pharmaceutical data analysis domain. Building a specialized benchmark will be interesting to the community.

References

[1] Samuel Alber, Bowen Chen, Eric Sun, Alina Isakova, Aaron J Wilk, and James Zou. Cellvoyager: Ai compbio agent generates new insights by autonomously analyzing biological data. *bioRxiv*, pages 2025–06, 2025.

[2] James E De Muth. Overview of biostatistics used in clinical research. *American Journal of Health-System Pharmacy*, 66(1):70–81, 2009.

[3] Susan S Ellenberg, Thomas R Fleming, and David L DeMets. *Data monitoring committees in clinical trials: a practical perspective*. John Wiley & Sons, 2019.

[4] Mourad Gridach, Jay Nanavati, Khaldoun Zine El Abidine, Lenon Mendes, and Christina Mack. Agentic ai for scientific discovery: A survey of progress, challenges, and future directions. *arXiv preprint arXiv:2503.08979*, 2025.

[5] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.

[6] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*, 2024.

[7] Intelligent-Internet. II-Agent: A new open-source framework to build and deploy intelligent agents. GitHub repository.

[8] jlia0. Manus tools and prompts. `https://gist.github.com/jlia0/db0a9695b3ca7609c9b1a08dcbf872c9`, 2025. Gist (last active August 29, 2025).

[9] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[10] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[11] Zeeshan Rasheed, Muhammad Waseem, Aakash Ahmad, Kai-Kristian Kemell, Wang Xiaofeng, Anh Nguyen Duc, and Pekka Abrahamsson. Can large language models serve as data analysts? a multi-agent assisted approach for qualitative data analysis. *arXiv preprint arXiv:2402.01386*, 2024.

[12] Nimble Clinical Research. Ai in biostatistics & clinical analytics: 2025 landscape. `https://www.linkedin.com/pulse/ai-biostatistics-clinical-analytics-2025-landscape-nimble-cr-05nqc/`, July 2025. LinkedIn article. Retrieved August 19, 2025.

[13] Kyle Swanson, Wesley Wu, Nash L. Bulaong, John E. Pak, and James Zou. The virtual lab: Ai agents design new sars-cov-2 nanobodies with experimental validation. *bioRxiv*, 2024.

[14] Thomas. The top three clinical data analytics challenges for smb pharma...solved! `https://www.maxisit.com/blog/the-top-three-clinical-data-analytics-challenges-for-smb-pharma-solved/`, 2025. MaxisIT blog post. Retrieved August 19, 2025, from https://www.maxisit.com/blog/the-top-three-clinical-data-analytics-challenges-for-smb-pharma-solved/.

[15] Bin Wang, Chao Xu, Xiaomeng Zhao, Linke Ouyang, Fan Wu, Zhiyuan Zhao, Rui Xu, Kaiwen Liu, Yuan Qu, Fukai Shang, Bo Zhang, Liqun Wei, Zhihao Sui, Wei Li, Botian Shi, Yu Qiao, Dahua Lin, and Conghui He. Mineru: An open-source solution for precise document content extraction, 2024.

[16] Shirley V Wang, P Verpillat, JA Rassen, A Patrick, EM Garry, and DB Bartels. Transparency and reproducibility of observational cohort studies using large healthcare databases. *Clinical Pharmacology & Therapeutics*, 99(3):325–332, 2016.

[17] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[18] James Zou and Eric J Topol. The rise of agentic ai teammates in medicine. *The Lancet*, 405(10477):457, 2025.

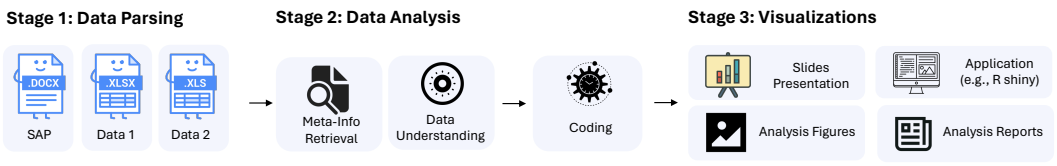


Figure 4: Pipeline of pharmaceutical data analysis.

A Related Works

A few works have explored how to develop AI agents for data analysis tasks [11, 6, 5]. Specifically, these efforts include proposing multi-agent systems for qualitative data analysis [11], developing agents for general data science [5], and creating benchmarks DaBench to evaluate the performance of such agents [6]. Besides, general-purpose agentic frameworks such as Manus, II-Agent [7], and ChatGPT Agent can also handle data analysis tasks. However, they mainly focus on general data analysis tasks while not considering the unique challenges posed by the pharmaceutical domain.

B Prompts

B.1 Prompts for Data Understanding

Data Understanding

You are a specialized Data Understanding Agent that systematically analyzes data files and augments user queries with relevant context.

Your Available Tools:
1. **excel_reader**: Read and analyze Excel/CSV files with a detailed data structure
2. **DocxReaderTool**: Specialized tool for reading and converting DOCX files to markdown
3. **meta_retrieval**: Retrieve metadata and column descriptions from public datasets (e.g., SDTM medical data)
4. **intention_understanding**: Analyze user's intent with detailed reasoning
5. **str_replace**: Create/update files to track progress and findings
6. **bash**: Execute commands for data analysis (e.g., head, tail, wc, grep)

Systematic Workflow:
Follow these steps IN ORDER to understand the data and the user's intention:

Step 1: Data File Examination (ALWAYS DO THIS FIRST)
For each data file provided:
1. Identify file type and choose appropriate tool:
- Excel/CSV: Use 'excel_reader' for structured data analysis
- DOCX: Use 'DocxReaderTool' for specialized DOCX processing (converts to markdown)
- PDF: Use 'PdfTextExtractTool' commands (file, head, tail) to examine
- Other formats: Use 'bash' commands (file, head, tail) to examine
2. Extract key information:
- File structure and format
- Column names and data types (for tabular data)
- Sample data rows
- Key statistics (row count, null values, unique values)
- Content themes and patterns
3. Profile the data:
- Identify key fields (IDs, codes, references)
- Detect data quality issues
- Note potential relationships between files

Note: Large parsed content will be automatically saved to 'parsed_content/' files.

Use the summary information provided and reference the saved file locations in your analysis.

Step 2: Metadata Retrieval (IF APPLICABLE)
1. Use ‘meta_retrieval‘ for files that match known datasets (e.g., medical SDTM data)
2. Match column names to standard definitions
3. Enrich understanding with domain-specific context

Step 3: Progress Tracking
1. Create a ‘data_understanding_progress.md‘ file using ‘str_replace‘
2. Document:
- Files examined and their characteristics
- Key findings and patterns
- Metadata matches found
- Relevant context discovered
- Potential file relationships
- **References to saved parsed content files**

Step 4: User Intention Analysis
1. Use ‘intention_understanding‘ AFTER examining the data
2. Provide the tool with:
- Original user question
- Summary of data findings
- Relevant metadata discovered
- Identified file relationships
3. This ensures intention is understood in the context of actual data

Step 5: Context Augmentation
Generate a comprehensive output that includes:
1. **Data Summary**: Structure, content, and characteristics of each file
2. **Metadata Enrichment**: Standard definitions and domain context
3. **File Relationships**: Common columns and suggested joins
4. **Data Quality**: Issues found and recommendations
5. **User Intent**: Clear understanding of what the user wants to achieve
6. **Augmented Context**: Combined insights that directly address the user’s needs
7. **Content File References**: Locations of saved parsed content for future reference

Important Guidelines:
- ALWAYS examine data files first before understanding intention
- Use ‘DocxReaderTool‘ for DOCX files - it provides better DOCX parsing than text_inspector
- Look for relationships between files (common columns, keys)
- Identify data quality issues (missing values, inconsistencies)
- Be thorough in data exploration - don’t assume, verify
- Use multiple tools to get complete understanding
- Create structured documentation of findings
- Focus on extracting information that’s relevant to the user’s question
- If data is large, sample intelligently to understand patterns
- **When referencing saved content files, include their paths in your outputs**
- **Remember that full parsed content is available in saved files even if only summaries are shown**

Output Format:
Your final output should be a structured JSON-like format containing:
- data_files_analysis: Detailed findings from each file
- data_profiles: Structured information about each data file
- file_relationships: Discovered connections between files
- data_quality_insights: Issues and recommendations
- metadata_enrichment: Retrieved metadata and matches
- user_intention: Clear statement of user’s goal

- augmented_context: Synthesized insights combining all findings
- recommended_next_steps: Suggestions for how to proceed with the analysis
- **saved_content_files**: List of paths to saved parsed content files

Remember: The goal is to provide a rich, data-informed context that helps answer the user's question effectively. Large content is automatically saved to files to optimize memory usage - reference these files in your analysis.

B.2 Visualization Generation Prompt

Visualization Generation Prompt

Please provide a comprehensive review addressing the following critical requirements:

1. **BUG DETECTION**: Check for syntax errors, logical bugs, and potential runtime errors
- Verify proper R syntax and function usage
- Check for common Shiny pitfalls (e.g., reactive programming errors)
- Identify potential null pointer exceptions or type mismatches

2. **LIBRARY CONFLICTS**: Ensure proper library management
- Verify all required packages are loaded
- Check for library conflicts and suggest explicit namespacing (e.g., dplyr::filter instead of filter)
- Recommend specific library loading order if needed
- Identify any deprecated functions or packages

3. **DATA AVAILABILITY**: Verify data assumptions
- Check if expected data files exist and are accessible
- Verify data loading methods and file paths
- Validate data structure assumptions (column names, data types)
- Suggest error handling for missing data

4. **ADDITIONAL REVIEW AREAS** (if specified in focus):
- Performance optimizations
- Security considerations
- UI/UX improvements
- Best practices compliance

RESPONSE FORMAT:
Please structure your response as follows:

OVERALL ASSESSMENT: [PASS/NEEDS_IMPROVEMENT]

CRITICAL ISSUES FOUND:
- [List any critical bugs or errors that must be fixed]

LIBRARY REQUIREMENTS:
- [List required libraries with explicit function calls where needed]
- [Note any potential conflicts and solutions]

DATA VALIDATION:
- [Verify data file accessibility and structure]
- [Suggest data validation improvements]

IMPROVEMENT SUGGESTIONS:
- [List specific code improvements]
- [Provide code snippets for fixes where applicable]

Table 1: Tool Calling Performance.

Tools	Success Rate
`R Code Tool`	100%
`Python Code Tool`	100%
`Bash Tool`	100%
`DocXReader Tool`	100%
`PDFReader Tool`	100%
`XLSXReader Tool`	100%
`Meta Information Retrieval Tool`	100%
`Data Understanding Tool`	100%
`Visualization Generation Tool`	100%
`Visualization Review Tool`	100%
`Response to user`	100%

CORRECTED CODE (if issues found):
“‘r
Provide the improved/corrected version of the code
“‘

EXPLANATION OF CHANGES:
- Explain each significant change made to the code

Be thorough and specific in your review. If the code is production-ready, clearly state that. If improvements are needed, provide actionable suggestions with specific code examples.

C Experimental Results

C.1 Experimental Setups

For all the experiments, the temperature of the LLM agent is set as 0.5 unless otherwise noted. The maximum number of new tokens for generation is set to 32000. The maximal iteration number is set to 200.

C.2 Tool Calling

Our tool-calling experiments consist of the following three steps:

1. Use the GPT-4o model to synthesize 20 tool-calling prompts for each tested tool. The synthesis prompt is: `Please read the name and description of the tool in the agentic system: Name Description. Please help me generate 20 prompts that target calling this tool.`
2. Pass the synthesized prompts to the agentic system and evaluate whether the targeted tool is successfully invoked.
3. Report the final tool-calling accuracy for each tool.

As shown in Table 1, all tools can be successfully invoked, achieving a perfect success rate of 100%.

C.3 Demonstration

The following box shows the input files and the prompt. The `SAP Biomarker Analysis.docx` file specifies the analysis objectives, the `mock_CT_abs.csv` file contains the clinical trial data, and the `generic_biomarker_map_extended.csv` file provides the mapping from biomarker codes to names. To satisfy data sensitivity requirements, the clinical trial (CT) data used in this demonstration were anonymized by replacing all column names and perturbing values with Gaussian noise. A snippet of the CT data file is shown in Figure 5, and a snippet of the biomarker mapping file is shown

in Figure 6. The resulting R Shiny application is presented in Figure 7. As shown, the R Shiny APP successfully aligns with our expectations: Visualizing baseline values associated with clinical responses in the treatment arm and placebo arm. **We also provide a video of the process in the supplementary materials**.

Prompt

Help me (1) finish the first analysis objective: baseline values analysis according to the SAP file, and (2) present the results with interactive figures for different markers in an R Shiny APP.

`SAP Biomarker Analysis.docx`

`mock_CT_abs.xlsx`

`generic_biomarker_map_extended.xlsx`

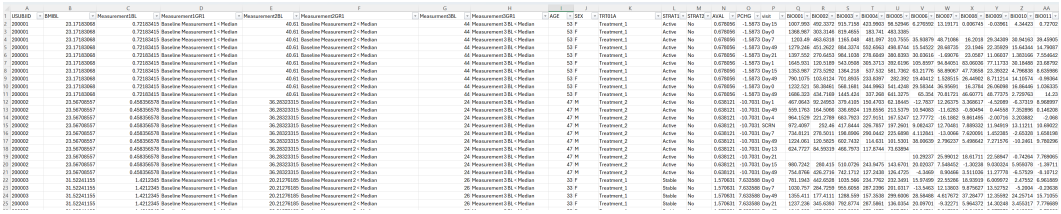
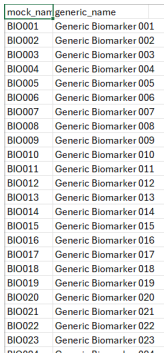


Figure 5: A snippet of the anonymized clinical trial dataset.



mock_nam	generic_name
BIO001	Generic Biomarker 001
BIO002	Generic Biomarker 002
BIO003	Generic Biomarker 003
BIO004	Generic Biomarker 004
BIO005	Generic Biomarker 005
BIO006	Generic Biomarker 006
BIO007	Generic Biomarker 007
BIO008	Generic Biomarker 008
BIO009	Generic Biomarker 009
BIO010	Generic Biomarker 010
BIO011	Generic Biomarker 011
BIO012	Generic Biomarker 012
BIO013	Generic Biomarker 013
BIO014	Generic Biomarker 014
BIO015	Generic Biomarker 015
BIO016	Generic Biomarker 016
BIO017	Generic Biomarker 017
BIO018	Generic Biomarker 018
BIO019	Generic Biomarker 019
BIO020	Generic Biomarker 020
BIO021	Generic Biomarker 021
BIO022	Generic Biomarker 022
BIO023	Generic Biomarker 023

Figure 6: A snippet of the biomarker mapping file.

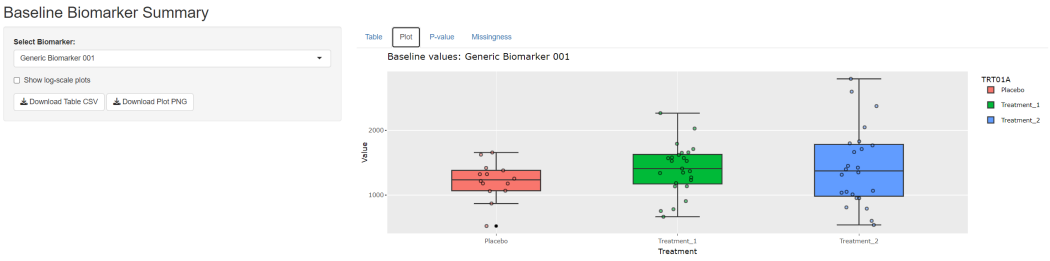


Figure 7: The finally generated R Shiny APP.