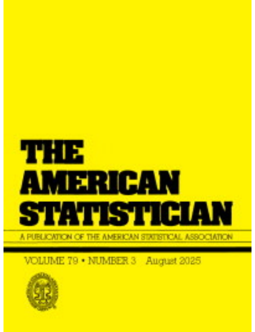

THE AMERICAN STATISTICIAN
A PUBLICATION OF THE AMERICAN STATISTICAL ASSOCIATION
VOLUME 79 • NUMBER 3 August 2025

The American Statistician

ISSN: 0003-1305 (Print) 1537-2731 (Online) Journal homepage: www.tandfonline.com/journals/utas20

A Survey on Large Language Model-based Agents for Statistics and Data Science

Sun Maojun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan & Jian Huang

To cite this article: Sun Maojun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan & Jian Huang (15 Sep 2025): A Survey on Large Language Model-based Agents for Statistics and Data Science, The American Statistician, DOI: 10.1080/00031305.2025.2561140

To link to this article: https://doi.org/10.1080/00031305.2025.2561140

© 2025 The Author(s). Published with license by Taylor & Francis Group, LLC.

View supplementary material

Accepted author version posted online: 15 Sep 2025.

Submit your article to this journal

Article views: 197

View related articles

View Crossmark data

Full Terms & Conditions of access and use can be found at
https://www.tandfonline.com/action/journalInformation?journalCode=utas20

A Survey on Large Language Model-based Agents for Statistics and Data Science

Maojun Sun[a], Ruijian Han[a], Binyan Jiang[a], Houduo Qi[a,b], Defeng Sun[b], Yancheng Yuan[b*] and Jian Huang[a,b]

*Corresponding authors.

[a] Department of Data Science and Artificial Intelligence, The Hong Kong Polytechnic University

[b] Department of Applied Mathematics, The Hong Kong Polytechnic University

Accepted Manuscript

Abstract

In recent years, data science agents powered by Large Language Models (LLMs), known as “data agents,” have shown significant potential to transform the traditional data analysis paradigm. This survey provides an overview of the evolution, capabilities, and applications of LLM-based data agents, highlighting their role in simplifying complex data tasks and lowering the entry barrier for users without related expertise. We explore current trends in the design of LLM-based frameworks, detailing essential features such as planning, reasoning, reflection, multi-agent collaboration, user interface, knowledge integration, and system design, which enable agents to address data-centric problems with minimal human intervention. Furthermore, we analyze several case studies to demonstrate the practical applications of various data agents in real-world scenarios. Finally, we identify key challenges and propose future research directions to advance the development of data agents into intelligent statistical analysis software.

Keywords: data agents; generative AI; data analysis; natural language interaction; statistical software.

1 Introduction

As nearly every aspect of society becomes digitized, data analysis has emerged as an indispensable tool across various industries (Inala et al., 2024). For instance, financial institutions leverage data analysis to make informed decisions about stock trends (Provost and Fawcett, 2013; Institute, 2011), hospitals utilize it to monitor patients’ health conditions (Waller and Fawcett, 2016), and companies employ it to develop strategic plans (Chen et al., 2012). Despite its widespread utility, data analysis is often perceived as a challenging field with a significant “entry barrier” (Cao, 2017; Jordan and Mitchell, 2015), typically requiring knowledge in areas such as statistics, data science, and computer science (Kitchin, 2014). Since the release of SPSS (IBM, 1968) in 1968, followed by SAS (Inc., 1976), Matlab (MathWorks, 1984), Excel (Microsoft, 1985), Python (Foundation, 1991), R (for Statistical Computing, 1995), PowerBI (Microsoft, 2013), and other specialized data analysis tools and programming languages, these advancements have significantly aided professionals in conducting statistical experiments and data analysis. Moreover, they have made data analysis more accessible to a broader range of practitioners (Witten et al., 2016).

The general data analysis process typically involves several key steps. Initially, data is collected from studies or extracted from databases and imported into tools such as Excel. Next, software like Excel or programming languages such as Python and R are employed to clean and analyze the data, aiming to extract valuable insights. Subsequently, data visualization is performed to make these insights more accessible and understandable. For more complex tasks, such as statistical inference and predictive analysis, statistical and machine learning models are often necessary. This involves data processing, feature engineering, modeling, evaluation, and more. Upon completing the analysis, a final report is usually drafted to summarize the findings and insights. However, for individuals without expertise in statistics, data science, and programming, data analysis remains a high-barrier task.

The barriers to data analysis primarily exist in the following areas:

- Lack of systematic statistical training: Individuals without a background in statistics may find it challenging to understand which types of analysis are feasible, even when data is presented to them. As data and models become increasingly complex, gaining a solid understanding of current statistical techniques typically requires at least a Master's level of statistical training.

- Software limitation: Simple data analysis tools like Excel are inadequate for complex scenarios, such as predictive analysis or analyzing data from enterprise databases. Conversely, advanced programming languages for data analysis, such as Python and R, require prior programming knowledge, which can be a barrier for many users.

- Challenges in domain-specific problems: In specialized fields like protein or genetic data analysis, general data scientists may find it difficult to perform effective analysis due to a lack of domain-specific knowledge.

- Difficulty in integrating domain knowledge: Corresponding to the last point, domain experts often lack the data science and programming skills needed to quickly incorporate their expertise into data analysis tools. For example, PSAAM (Steffensen et al., 2016) is software designed for the curation and analysis of metabolic models, yet a biologist researching metabolism might find it challenging to integrate this analytical method into common data analysis tools like Excel or R.

With the rise of generative AI, new opportunities have emerged in statistics and data science. LLM-based data agents are gradually addressing existing challenges while introducing a new paradigm for approaching data analysis tasks.

An "AI agent" (or LLM agent) refers to an autonomous or semi-autonomous software system powered by AI models such as LLMs. These agents can interpret natural language instructions, plan and execute tasks, and interact with users or other systems to complete complex workflows (Cheng et al., 2024).

Specifically, we define an LLM-based data agent as an autonomous or semi-autonomous software system powered by LLMs, capable of understanding natural language instructions, planning and executing data-centric tasks, and interacting with users or external tools to accomplish complex objectives—from exploratory data analysis to machine learning model

Accepted Manuscript

development. In this paper, the terms “LLM-based data science agent,” “LLM-based data agent,” and “data science agent” are collectively referred to as “data agent” for simplicity.

This survey explores recent advancements in data agents and highlights data analysis performed by various agents through a series of case studies. In Section 2, we briefly discuss the opportunities introduced by recent developments in generative AI. Section 3 reviews and categorizes recent work on data science agents. We then present several case studies in Section 4. Section 5 examines the challenges and future directions in this field, followed by our discussion in Section 6. Finally, we present our conclusions in Section 7.

2 Opportunities Brought by Generative AI

The rise and potential of generative AI, particularly Large Language Models (LLMs) or vision language models (VLMs) in the field of data science and analysis have gained increasing recognition in recent years. In addition to understand text, LLMs are also trained to understand tabular data, allowing them to effectively extract insights, identify patterns, and draw meaningful conclusions from tables (Dong and Wang, 2024). Consequently, LLMs have emerged as powerful tools capable of significantly enhancing and transforming a variety of data-driven applications and workflows (Nejjar et al., 2023; Tu et al., 2023; Cheng et al., 2023). Recent research has focused on designing LLM-based data science agents (data agents) to automatically address data science tasks through natural language, as demonstrated by tools like ChatGPT-Advanced Data Analysis (ChatGPT-ADA) (OpenAI, 2023), LAMBDA (Sun et al., 2024) and Colab Data Science Agent (Google, 2025).

The emergence of data agents offers a potential solution to the previously mentioned challenges, as they lower the entry barrier for users who lack programming or statistical knowledge. By providing an intuitive interface that harnesses the capabilities of LLMs, users can request analyses using natural language, and the data agents can interpret these instructions, access relevant data, and autonomously apply appropriate analytical techniques. For example, a user might request, “Calculate the sales growth in different regions from 2021 to 2028, generate a bar chart to visualize the results, and provide key insights.” With this simplified instruction, data agents can automatically extract, analyze, visualize, and report data, reducing the requirement for technical expertise and fostering a more efficient workflow. This significantly lowers the entry barriers for individuals unfamiliar with traditional data analysis tools and methods.

Furthermore, by embedding specialized knowledge into LLMs, data agents can potentially overcome challenges faced by data scientists in fields like genomics, where domain expertise is crucial (Cao, 2017). Simultaneously, domain experts who may lack data science or programming skills can rely on data agents to seamlessly integrate their expertise into data analysis workflows. This ability to bridge the gap between domain expertise and data science has the potential to advance interdisciplinary research and decision-making in complex scenarios.

3 LLM-based Data Science Agents

3.1 Overview

Accepted Manuscript

LLM-based data agents leverage the powerful natural language understanding and generation capabilities of LLMs to autonomously tackle complex data analysis tasks. Figure 3 illustrates a commonly used framework for these agents.

In this framework, the LLM serves as the core of the entire system, driving its performance and reliability. As such, the capabilities of the LLM are critical to the system's effectiveness, with advanced models like GPT-4 often being used. Data analysis typically involves multiple steps, especially when addressing complex tasks. Techniques such as Planning, Reasoning, and Reflection help ensure that the LLM processes these tasks with greater logical coherence and makes optimal use of its knowledge.

In the architecture, the LLM generates the code for a given data analysis task, executes it, and retrieves the corresponding results. This requires an execution environment, represented by the Sandbox, which safely isolates the code execution process. The Sandbox allows users to run programs and access files without risking the underlying system or platform. It includes pre-installed programming environments and software, such as Python, R, Jupyter, and SQL Server.

A user-friendly interface is also essential to improving usability. An intuitive interface not only attracts users but also enables them to quickly engage with and utilize the system effectively.

3.2 Evolution of Data Science Agent

Research on data agents began gaining momentum in 2023. Chandel et al. (2022) trained and evaluated a model within a Jupyter Notebook to predict code based on given commands and results. Soon after, it was discovered that LLMs, such as GPT, could generate accurate code for basic data analysis. With the rise of the LLM-based agent, researchers began designing special data agents for automating data science and analysis tasks by human language. Figure 2 shows some selected works from 2023, while Table 1 illustrates some key characteristics.

3.3 User Interface

The user interface is crucial for attracting users at first glance. Current research on user interface design can be broadly categorized into four types: Integrated Development Environment-based (IDE-based), Independent System, Command line-based (Command-based), and Operation System-based (OS-based).

IDE-based Integrated Development Environments (IDEs) such as Jupyter provide convenient tools for data science and analysis. Recent efforts, including Colab Data Science Agent (Google, 2025), Jupyter-AI (jupyterlab, 2023), Chapyter (chapyter, 2023), and MLCopilot (Zhang et al., 2023a), have incorporated LLMs into Jupyter environments. For example, Colab Data Science Agent enables planning, automatic code cell generation, execution, and result presentation in the notebook. This approach is particularly popular because it allows users to review, edit, and run code directly.

Independent System Some works have focused on developing independent systems equipped with user interfaces. For example, ChatGPT introduced a streamlined, intuitive conversational system—a model of interaction that has been widely adopted in subsequent

Accepted Manuscript

projects. In the context of data analysis tasks, beyond basic text-based input and output, several systems have introduced specialized features, such as visualization, report generation, and file download options, to simplify user interactions. For instance, LAMBDA (Sun et al., 2024) facilitates easy data review by enabling intuitive data display after users upload their data. Data Formulator 2 (Wang et al., 2024a) further enhances the iterative process of creating data visualizations through a multi-modal interface, combining graphical user interface (GUI) elements with natural language inputs, allowing users to specify their visualization intentions with both precision and flexibility. WaitGPT (Xie et al., 2024) addresses the challenge of understanding and verifying LLM-generated code by transforming raw code into an interactive, step-by-step visual representation. This allows users to comprehend, validate, and adjust specific data operations, actively guiding and refining the analysis process.

Command Line-based Works like Data Interpreter (Hong et al., 2024) and TaskWeaver (Qiao et al., 2023) using command-line interfaces (CLI) in their works. For researchers and experienced users, it provides greater flexibility and control over the system, allowing users to execute a wide range of functions in the command line and customize their actions. Besides, command-based interfaces often require less computational overhead compared to graphical user interfaces, making them more efficient.

OS-based OS-based agents, such as UFO (Zhang et al., 2024), are designed to operate directly within an operating system environment, allowing them to control a wide range of system tasks and resources. Similarly, Spider2-V (Cao et al., 2024) simulates the typical workflow of a data scientist by mimicking actions such as clicking, typing, and writing code, providing an OS-level interactive experience that closely resembles how humans manage data science tasks. However, while OS-based agents like Spider2-V lay a solid foundation for user interaction, achieving full automation of the data science workflow remains an ongoing challenge (Cao et al., 2024).

3.4 Planning, Reasoning, and Reflection

Planning, Reasoning, and Reflection often play crucial roles in guiding the actions of data agents. In particular, planning and reasoning emphasize the generation of a logically structured sequence or roadmap of actions and thought processes to systematically address problems step by step (Huang et al., 2024b; Hong et al., 2024). Complex tasks often require a step-by-step approach to ensure effective resolution, while simpler tasks can be handled without such detailed breakdowns. Recently, GPT-4o (OpenAI, 2024) introduces a planning architecture that integrates external tools and decomposes complex tasks into structured sub-tasks, enabling more accurate and controllable multi-step reasoning.

Some approaches focus on building conversational data agents (Zhang et al., 2023b, a; Sun et al., 2024), where users interact with the agent over multiple rounds to complete a task. In these cases, under human supervision, complex planning is not necessary, as guidance can simplify decision-making and adjust the workflow dynamically. Some of these works operate in a Basic I/O mode. On the other hand, End-to-end data agents (Guo et al., 2024; Qiao et al., 2023; Hong et al., 2024; Chi et al., 2024; Jiang et al., 2024; Li et al., 2024; Trirat et al., 2024; Grosnit et al., 2024) are designed to allow users to issue a single prompt that encompasses all requirements. In these cases, the agent employs planning, reasoning, and reflection to iteratively complete all tasks autonomously.

Accepted Manuscript

Recent research in planning has introduced two main approaches: Linear Structure Planning (or Single Path Planning/Reasoning) and Hierarchical Structure Planning (or Multiple Path Planning/Reasoning). Figure 4 illustrates some recent planning methodologies like Chain-of-Thought (CoT) (Wei et al., 2022), ReAct (Yao et al., 2022), Tree-of-Thoughts (ToT) (Yao et al., 2024), and Graph-of-Thoughts (GoT) (Besta et al., 2024).

Linear Structure Planning In linear structure planning, a task is decomposed into a sequential, step-by-step process. For example, DS-Agent (Guo et al., 2024) utilizes Case-Based Reasoning to retrieve and adapt relevant insights from a knowledge base of past successful Kaggle solutions. This approach allows the agent to learn from previous experiences and continuously improve its performance. Similarly, AutoML-Agent (Trirat et al., 2024) adopts a retrieval-augmented planning (RAP) strategy to generate diverse plans for AutoML tasks. By leveraging the knowledge embedded in LLMs, information retrieved from external APIs, and user requirements, RAP allows the agent to explore a wider range of potential solutions, leading to more optimal plans.

Hierarchical Structure Planning Simple linear planning is often insufficient for complex tasks. Such tasks may require hierarchical and dynamic, adaptable plans that can account for unexpected issues or errors in execution (Hong et al., 2024). For instance, Hong et al. (2024) utilizes a hierarchical graph modeling approach that breaks down intricate data science problems into manageable sub-problems, represented as nodes in a graph, with their dependencies as edges. This structured representation enables dynamic task management and allows for real-time adjustments to evolving data and requirements. Additionally, they further introduce "Programmable Node Generation," to automate the generation, refinement, and verification of nodes within the graph, ensuring accurate and robust code generation. AIDE (Jiang et al., 2024) employs Solution Space Tree Search to iteratively improve solutions through generation, evaluation, and selection components. Similarly, SELA (Chi et al., 2024) combines LLMs with Monte Carlo Tree Search (MCTS) to enhance AutoML performance. It starts by using LLMs to generate insights for various machine learning stages, creating a search space for solutions. MCTS then explores this space by iteratively selecting, simulating, and back-propagating feedback, enabling the discovery of optimal pipelines. Agent K v1.0 (Grosnit et al., 2024) employs a structured reasoning framework with memory modules, operating through multiple phases. The first phase, automation, handles data preparation and task setup, generating actions through structured reasoning. The second phase, optimization, involves solving tasks and enhancing performance using techniques such as Late-Fusion Model Generation and Bayesian optimization. The final phase, generalization, utilizes a memory-driven system for adaptive task selection.

Reflection Reflection enables an agent to evaluate past actions and decisions, adjust strategies, and improve future task performance. This process is essential for self-correction and debugging during task execution. For example, Wang et al. (2024b) employs trajectory filtering to train agents that can learn from interactions and enhance their self-debugging capabilities. This technique involves selecting trajectories in which the model initially makes errors but successfully corrects them through self-reflection in subsequent interactions. Similarly, Data-copilot (Zhang et al., 2023b) and LAMBDA (Sun et al., 2024) use self-reflection based on code execution feedback to address errors. If a compilation error occurs, the agents repeatedly attempt to revise the code until it runs successfully or a maximum retry limit is reached. This iterative process helps ensure code correctness and usability.

3.5 Multi-agent Collaboration

Accepted Manuscript

Multi-agent System (MAS) enable task decomposition through role assignment. In this setup, agents communicate, negotiate, and share information to optimize their collective performance (Xi et al., 2023). It offers several advantages over single-agent setups. First, they reduce redundant and complex context accumulation by isolating responsibilities across agents. Second, each agent instance can be powered by a different language model, opening opportunities to specialize models for domain-specific expertise. For example, in LAMBDA (Sun et al., 2024), a dedicated Programmer Agent is responsible for code generation, while noisy error outputs are handled separately by an Inspector Agent. This separation helps the Programmer Agent avoid context overload, simplifies historical trace management, and ultimately improves response accuracy.

AutoGen introduces a programming framework specifically designed for constructing MAS (Wu et al., 2023). Furthermore, AutoML-Agent (Trirat et al., 2024) involves the Agent Manager, Prompt Agent, Operation Agent, Data Agent, and Model Agent—that together cover the entire pipeline, from data retrieval to model deployment. OpenAgents (Xie et al., 2023) consisted of agents such as the Data Agent, Plugins Agent, and Web Agent. Similarly, AutoKaggle (Li et al., 2024) employs agents like Reader, Planner, Developer, Reviewer, and Summarizer to manage each phase of the process, ensuring comprehensive analysis, effective planning, coding, quality assurance, and detailed reporting. These collaborating mode help decentralized the complicated task, allowing each agent to focus on its specific role, thereby enhancing the overall efficiency and effectiveness of the data analysis process.

3.6 Knowledge Integration

Integrating domain-specific knowledge into data agents presents a challenge (Dash et al., 2022; Sun et al., 2024). For example, when a domain expert has specialized knowledge, such as specific protein analysis code, the agent system are expected able to incorporate and apply this knowledge effectively. One approach is tool-based, where the expert’s analysis code is treated as a tool that is recognizable by the LLM (Xie et al., 2023). When the agent encounters a relevant problem, it can call upon the appropriate tool from its library to execute the specialized analysis. Another method involves the Retrieval-Augmented Generation (RAG) technique (Lewis et al., 2020), where relevant code is first retrieved and then embedded within the context to facilitate in-context learning. LLM-based agents can also access and interact with external knowledge sources, such as databases or knowledge graphs, to augment their reasoning capabilities (Wang et al., 2024b).

Sun et al. (2024) proposes a Knowledge Integration method that builds on this concept. In LAMBDA, analysis codes are parsed into two parts: descriptions and executable code. These are then stored in a knowledge base. When the agent receives a task, it retrieves the relevant knowledge based on the similarity between the task description and the descriptions stored in the knowledge base. The corresponding code is then used for in-context learning (ICL) or back-end execution, depending on the configuration. This approach enables agents to effectively leverage domain-specific knowledge in relevant scenarios.

3.7 Benchmarks for Evaluating Data Agents

Evaluating the performance of data agents is crucial for understanding their effectiveness and reliability. Current benchmarks primarily rely on deterministic output comparisons, where an LLM processes a task, generates code, and is evaluated based on the final execution results.

Accepted Manuscript

For example, DS-1000 (Lai et al., 2022) provides a large-scale benchmark of 1000 realistic problems spanning seven core Python data science libraries, with execution-based multi-criteria evaluation and mechanisms to reduce memorization bias. MLAgentBench (Huang et al., 2024a) introduces a benchmark focused on machine learning research workflows by constructing an LLM-agent pipeline. Furthermore, InfiAgent-DABench (Hu et al., 2024) presents a end-to-end benchmark for evaluating the capabilities of data agents, the tasks require agents to end-to-end solving complex tasks by interacting with an execution environment. However, for tasks such as data visualization, the outputs are often difficult to compare directly. Designing effective evaluation strategies for data visualizations remains an open and important question.

3.8 System Design and Other Related Works

Recent advancements in interactive data science systems highlight a variety of approaches in system design, with LLMs and structured frameworks significantly enhancing the user experience across key areas such as data visualization, task specification, predictive modeling, and data exploration. Notable systems like VIDS (Hassan et al., 2023), Data-Copilot (Zhang et al., 2023b), InsightPilot (Ma et al., 2023), and JarviX (Liu et al., 2023) exemplify diverse design principles tailored to these specific functions. For instance, Data-Copilot adopts a code-centric approach, generating intermediate code to process data and subsequently transforming it into visual outputs, such as charts, tables, and summaries (Zhang et al., 2023b).

Other frameworks emphasize workflow automation. InsightPilot integrates an “insight engine” that guides data exploration, reducing LLM hallucinations and enhancing the accuracy of exploratory tasks (Ma et al., 2023). JarviX, in combination with MLCopilot (Zhang et al., 2023a), contributes to automated machine learning by merging LLM-driven insights with AutoML pipelines. Additionally, in the domain of database management, systems like LLMDB (Zhou et al., 2024) improve efficiency and reduce hallucinations and computational costs during tasks such as query rewriting, database diagnosis, and data analytics. In terms of data visualization, MatPlotAgent (Yang et al., 2024) transforms raw data into clear, informative visualizations by leveraging both code-based and multi-modal LLMs.

Moreover, Data Formulator 2 (Wang et al., 2024a) organizes user interactions into ”data threads” to provide context and facilitate the exploration and revision of prior steps. A similar approach is seen in WaitGPT (Xie et al., 2024), which transforms raw code into an interactive visual representation. This provides a step-by-step visualization of LLM-generated code in real-time, allowing users to understand, verify, and modify individual data operations. SEED (Chen et al., 2024) combines LLMs with methods like code generation and small models to produce domain-specific data curation solutions. HuggingGPT (Shen et al., 2024), on the other hand, uses LLMs to coordinate a variety of expert models from platforms such as Hugging Face, solving a broader range of AI tasks across multiple modalities.

Lastly, in terms of industry applications, lots of companies have used agents in the business analysis. For example FUTU use AI to analyze the stock market and provide investment advice (FUTU, 2024). Julius (Julius, 2025) facilitates data science education by building a bridge that allowing professors to create interactive workflows for lessons, which can be shared with students for a seamless teaching experience through natural language interaction.

Accepted Manuscript

Accepted Manuscript

4 Data Analysis Through Natural Language Interaction: Case Studies

In this section, we present a series of case studies conducted by a diverse range of agents, each illustrating the new data analysis paradigm facilitated through natural language interaction. These case studies demonstrate how this approach enables users to engage with data more intuitively and effectively, breaking down traditional barriers to data accessibility and understanding. By leveraging natural language processing, these agents can interpret and respond to complex queries, providing insights that are both comprehensive and easily digestible. Through these examples, we aim to highlight the transformative potential of natural language interaction in data analysis.

4.1 Case study 1: Exploratory Data Analysis and Model Building by Conversational Data Agents

In this case study, we utilized ChatGPT and LAMBDA to demonstrate exploratory data analysis (EDA) and a simple model building process. Specifically, we first used ChatGPT to explore the effect of alcohol content on the quality of different types of wine, focusing on both red and white varieties. Then, we used LAMBDA to illustrate an interactive modeling process and automatically generate analysis reports.

We used the Wine Quality dataset, a tabular dataset with dimension 4898×11. The goal is to examine how 10 coviarates in this dataset affect the wine quality rating. We employed ChatGPT-ADA to conduct EDA and visualize the influence of alcohol content on wine quality ratings. Figure 5 illustrates the detailed planning and problem-solving process.

GPT-ADA first analyzed the problems and then outlined a step-by-step plan to solve the tasks. The entire workflow proceeded smoothly, with the code running efficiently to load the data, check for missing values, and generate visualizations, with each step delivering accurate results. Its ability to interpret data and provide insights significantly streamlined the analytical process. Finally, it provided insights into the relationship between quality scores and alcohol content.

Next, we train a set of models to predict wine quality using LAMBDA. LAMBDA facilitates an interactive analysis process, enabling us to perform tasks such as data processing, feature engineering, model training, parameter tuning, and evaluation through a series of guided conversations. Finally, we used LAMBDA's built-in report generation feature to compile a analysis report, which includes details of the tasks completed in the conversation history. The analysis process, including the conversation and the generated report, is presented in Figure 6.

As beginner-level users, we first asked LAMBDA to recommend some models, and it suggested advanced options like XGBoost. Next, we tasked LAMBDA with basic data preprocessing, which it handled correctly. We then trained and evaluated the recommended models using 5-fold cross-validation, a task LAMBDA performed exceptionally well, even providing download links for the resulting models. Finally, we used LAMBDA's report generation feature to create a structured and comprehensive report that effectively captured the key insights.

This example demonstrates the effectiveness of conversational data agents like ChatGPT and LAMBDA in streamlining the data visualization and machine learning workflow, particularly for users without programming experience.

4.2 Case Study 2: Residual Diagnostics and Heteroskedasticity Testing

To examine the ability of LLM-based data agents to perform statistically rigorous regression diagnostics, we prompted LAMBDA and GPT-4o to conduct a linear regression analysis using the Auto MPG dataset, a tabular data with dimension of 398 *times* 7. The goal was to predict `mpg` (miles per gallon) based on vehicle characteristics, notably `horsepower` and `weight`. The prompt and response of LAMBDA are detailed in the figure 7.

LAMBDA correctly loaded the dataset, performed appropriate preprocessing (e.g., handling non-numeric entries), and fit a linear model using `statsmodels`. It then computed and visualized residuals, followed by executing the Breusch–Pagan test for heteroskedasticity. The test output included the LM statistic and associated p-value, indicating a strong violation of the homoskedasticity assumption.

The residual plot visually confirmed increasing residual variance with larger fitted values. LAMBDA also summarized next steps, suggesting robust standard errors or model transformation to address heteroskedasticity. This example demonstrates LAMBDA’s ability to execute, interpret, and communicate statistically meaningful diagnostics in a flexible code-first environment. Besides, GPT-4o was also able to complete the same task successfully; further details and chat transcripts can be found in the supplementary materials.

4.3 Case Study 3: Bootstrap Confidence Intervals

In this case study, we assessed whether LLM-based data agents can perform non-parametric inference through bootstrap resampling. Using the Wine Quality dataset, the task was to estimate the average alcohol content for red wine and construct a 95% confidence interval using 1000 bootstrap resamples. Figure 7 shows the interaction with LAMBDA for completing this task.

LAMBDA successfully filtered the dataset to isolate red wines, extracted the `alcohol` variable, and implemented the bootstrap routine by repeatedly sampling with replacement. It then computed the empirical 2.5th and 97.5th percentiles of the bootstrapped means to form the confidence interval. The agent also produced a histogram showing the bootstrap distribution, overlaid with the CI bounds and sample mean.

This case illustrates that LAMBDA is capable of performing robust uncertainty quantification and generating high-quality visual explanations without relying on strict parametric assumptions. GPT-4o also successfully completed this task; its outputs and detailed interactions are included in the supplementary materials.

We found that different prompting may lead to differences in implementation details, such as the choice of hyperparameters or types of plots.

4.4 Case study 4: Expandability of Data Agents

Accepted Manuscript

In many situations, we encounter tasks that cannot be handled effectively using LLMs because their training data do not include the necessary knowledge for such tasks. In these cases, if a data agent is designed to be extensible, manual tool expansion or knowledge integration can address this limitation. In this case study, we demonstrate how both the Data Interpreter and LAMBDA leverage integration mechanisms to incorporate additional packages or domain-specific knowledge.

Tools Integration in Data Interpreter In this example, our objective is to extract submission deadlines for AI conferences from a public website [1] and save the results. We prompted the agent with the target URL and the desired output format. The agent successfully identified relevant information such as conference names and deadlines and generated structured output. The complete workflow, including prompt, execution, and results, is shown in Figure 8.

In this example, the Data Interpreter began with an initial plan. For each sub-task, it recommended relevant tools with a score indicating their suitability. The system then decided whether to use the suggested tool. For instance, it used `scrape_web_playwright` for a web-scraping task. This iterative recommendation and tool selection process continued until all sub-tasks were completed, addressing limitations in LLMs’ built-in abilities and knowledge.

Knowledge Integration in LAMBDA In this example, we consider the problem of training a Fixed Point Non-Negative Neural Network (FPNNN), which is defined as a neural network that maps nonnegative vectors to nonnegative vectors. We train a FPNNN with MNIST data. First, we integrated the code into the knowledge base. Then, we defined the model as `Core` and delineated the `Core` function, which directly accepts parameters, and the `Runnable` function, which was defined and executed separately. Figure 6 presents the configuration, prompt, and problem-solving process.

LAMBDA first retrieved the relevant code from the knowledge base, and then its `Core` function was presented in the context. By modifying the core code, LAMBDA generated the correct code and completed the task successfully.

5 Challenges and Future Directions

In this section, we highlight some challenges and suggest future directions in using LLMs or LLM-based data agents for statistical analysis.

5.1 Challenges in the Capabilities of LLMs

LLMs function as the “brain” of a data agent, interpreting user intent and generating structured plans to carry out data analysis tasks. For a data agent to be effective, it must possess advanced knowledge in statistics, data science, and programming, enabling it to support users throughout the analytical process.

Advanced Models Current state-of-the-art models like GPT-4 show strong performance on undergraduate-level mathematics and statistics problems, yet struggle with more advanced, graduate-level tasks (Frieder et al., 2023). Additionally, the success rate of fully automating complete data workflows with current agents remains low (Cao et al., 2024). This

Accepted Manuscript

suggests that enhancements in LLMs, particularly in knowledge of statistics and data analysis, are still needed.

Multi-Modality and Reasoning A key challenge for current LLMs lies in processing multi-modal inputs, including charts, tables, and code, which are essential to data analysis workflows (Inala et al., 2024). Future advancements may improve the ability to perform reasoning across mixed modalities, such as generating visualizations by replicating the style of an input visualization.

5.2 Challenges in Statistical Analysis

Intelligent Statistical Analysis Software While established tools such as SPSS and R are highly mature, data agents have the potential to transform statistical analysis through intelligent assistance. To realize this vision, agents must support flexible package integration, facilitate contributions from domain experts, and remain aligned with evolving programming ecosystems. Such a collaborative framework could accelerate innovation in the field. Furthermore, by guiding users and recommending appropriate methods, data agents can enhance research efficiency and expand access to advanced statistical techniques.

Incorporating Other Large Models into Statistical Analysis Statistical analysis of complex data is increasingly leveraging representations generated by large models for research purposes. For example, in predicting the tertiary structure of proteins, LLMs can utilize representations of primary and secondary structures—capabilities that traditional statistical software such as Matlab and R currently lack. Similarly, in the analysis of electronic health records, LLMs are being used to construct meaningful representations that facilitate downstream analysis. If data agents can effectively harness domain-specific knowledge models, they have the potential to significantly advance statistical and data science research, enabling more sophisticated analyses and fostering deeper insights across scientific disciplines.

5.3 Challenges in Real-World Adoption

Although the data agents have shown great potential in improving the accessibility of data analysis, there are still several challenges that need to be addressed for real-world adoption.

Trade-off Between Hardware and Privacy First, deploying large language models often requires high-performance computing resources. Running these models on CPU-only machines results in slow inference. API-based solutions also raise concerns about data privacy and security, as sensitive information may be transmitted to external servers. This is especially critical in fields such as healthcare and finance, where data confidentiality is paramount. Therefore, developing lightweight, expert-level data science models that can run efficiently on local machines without compromising performance is essential.

High-concurrency System High-concurrency environments pose significant scalability issues. In client-server architectures where each user session is associated with an isolated sandbox for secure code execution, the server may experience substantial resource strain under heavy load. Maintaining a large number of concurrent sandboxes can overwhelm system resources, leading to degraded performance or system instability. Therefore, the

Accepted Manuscript

design of efficient scheduling algorithms to manage limited computational resources across multiple sandbox instances becomes critical.

Integration with Existing Workflows While data agents excel in lowering the barrier to entry for non-programmers, they currently lack the flexibility and debugging capabilities of traditional IDEs. This makes them less suitable for complex, customized workflows that require iterative development and fine-grained control. A promising direction is to support the seamless export of an agent's actions (Sun et al., 2024), such as executed code, into IDEs like Jupyter Notebooks, which can serve as a bridge for smoother integration with conventional tools and workflows.

6 Discussion

6.1 Model Level Reproducibility

While data agents are generally robust to variations in prompt phrasing and can reliably complete the intended analytical tasks, we observed notable differences in their reasoning processes and implementation details. For example, when prompted to perform regression diagnostics, different phrasings such as "analyze residuals" versus "check model assumptions" resulted in the same core analysis but with different statistical tests or plotting choices. Similarly, in visualization tasks, one prompt might produce a bar chart while another yields a pie chart, depending on how the goal is described. Even for model training, default hyperparameters, such as learning rate or number of iterations, could vary slightly across prompts, leading to differences in performance metrics. These variations do not typically prevent task completion but can impact result interpretability, especially in rigorous statistical workflows where consistency across runs is critical.

6.2 System Level Reproducibility

Experiment Setting Experiment reproducibility can be enhanced through careful experiment designs. For example, LAMBDA (Sun et al., 2024) incorporates built-in mechanisms to export the full execution history into executable formats such as Jupyter Notebooks. When combined with proper experiment controls, such as setting random seeds, these exports enable end-to-end reproducibility of experimental results. In addition, designing human-in-the-loop mechanisms allows users to inspect, edit, or revise the code generated by LLMs during the problem-solving process. This interactive approach further supports reproducibility by enabling manual correction and verification of intermediate steps.

Version Control and Workflow Management Version control tools such as Git can enhance reproducibility by tracking changes in code, data, and prompts, making it easier to reproduce results and collaborate with others. Furthermore, workflow management systems like Snakemake and Nextflow allow users to define and automate each step of the analysis pipeline, ensuring that processes can be reliably repeated. When used alongside data agents, these tools can greatly improve both reproducibility and transparency. However, most current data agents lack native support for these tools, presenting opportunities for future development.

7 Conclusion

Accepted Manuscript

This survey has explored the recent progress of LLM-based data science agents. These agents have shown great potential in making data analysis more accessible to a wider range of users, even those with limited technical skills. By leveraging the capabilities of LLMs, they are able to handle various data analysis tasks, from data visualization to machine learning, through natural language interaction.

However, as discussed, they also face several challenges. In terms of model capabilities, improvements are needed in domain-specific knowledge and multi-modal handling. For intelligent statistical analysis software, seamless package management and community building are crucial. Additionally, effectively integrating other large models into statistical analysis and addressing data infrastructure and evaluation issues remain important areas for future development.

Overall, while LLM-based data science agents have made significant strides, continuous research and innovation are required to overcome the existing challenges and fully realize their potential in revolutionizing the field of data analysis.

Acknowledgments

The authors are grateful to the Editor, Associate Editor and two anonymous reviewers for their valuable comments and suggestions, which significantly improved the quality of the paper.

Funding

This work was funded by the Centre for the Mathematical Foundations of Generative AI and the research grants from The Hong Kong Polytechnic University (P0046811). The research of Ruijian Han was partially supported by The Hong Kong RGC grant (14301821) and The Hong Kong Polytechnic University (P0044617, P0045351, P0050935). The research of Binyan Jiang was partially supported by The Hong Kong RGC grant (15302722). The research of Houduo Qi was partially supported by the Hong Kong RGC grant (15309223) and The Hong Kong Polytechnic University (P0045347). The research of Defeng Sun and Yancheng Yuan was partially supported by the Research Center for Intelligent Operations Research at The Hong Kong Polytechnic University (P0051214). The research of Jian Huang was partially supported by The Hong Kong Polytechnic University (P0042888, P0045417, P0045931).

Disclosure Statement

The authors report there are no competing interests to declare.

References

Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., et al. (2024). Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Accepted Manuscript

Cao, L. (2017). Data science: Challenges and directions. *Communications of the ACM*, 60(8):59–68.

Cao, R., Lei, F., Wu, H., Chen, J., Fu, Y., Gao, H., Xiong, X., Zhang, H., Mao, Y., Hu, W., Xie, T., Xu, H., Zhang, D., Wang, S., Sun, R., Yin, P., Xiong, C., Ni, A., Liu, Q., Zhong, V., Chen, L., Yu, K., and Yu, T. (2024). Spider2-v: How far are multimodal agents from automating data science and engineering workflows?

Chandel, S., Clement, C. B., Serrato, G., and Sundaresan, N. (2022). Training and evaluating a jupyter notebook data science assistant. *arXiv preprint arXiv:2201.12901*.

chapyter (2023). Chapyter. https://github.com/chapyter/chapyter.

Chen, H., Chiang, R. H., and Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188.

Chen, Z., Cao, L., Madden, S., Kraska, T., Shang, Z., Fan, J., Tang, N., Gu, Z., Liu, C., and Cafarella, M. (2024). Seed: Domain-specific data curation with large language models. arxiv 2023. *arXiv preprint arXiv:2310.00749*.

Cheng, L., Li, X., and Bing, L. (2023). Is gpt-4 a good data analyst? *arXiv preprint arXiv:2305.15038*.

Cheng, Y., Zhang, C., Zhang, Z., Meng, X., Hong, S., Li, W., Wang, Z., Wang, Z., Yin, F., Zhao, J., et al. (2024). Exploring large language model based intelligent agents: Definitions, methods, and prospects. *arXiv preprint arXiv:2401.03428*.

Chi, Y., Lin, Y., Hong, S., Pan, D., Fei, Y., Mei, G., Liu, B., Pang, T., Kwok, J., Zhang, C., et al. (2024). Sela: Tree-search enhanced llm agents for automated machine learning. *arXiv preprint arXiv:2410.17238*.

Dash, T., Chitlangia, S., Ahuja, A., and Srinivasan, A. (2022). A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1040.

Dong, H. and Wang, Z. (2024). Large language models for tabular data: Progresses and future directions. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’24, page 2997–3000, New York, NY, USA. Association for Computing Machinery.

for Statistical Computing, R. F. (1995). *R: A Language and Environment for Statistical Computing*.

Foundation, P. S. (1991). *Python Programming Language*.

Frieder, S., Pinchetti, L., Chevalier, A., Griffiths, R.-R., Salvatori, T., Lukasiewicz, T., Petersen, P. C., and Berner, J. (2023). Mathematical capabilities of chatgpt.

FUTU (2024). Futubull ai.

Accepted Manuscript

GLM, T. (2024). Chatglm: A family of large language models from glm-130b to glm-4 all tools.

Google (2025). Data science agent in colab with gemini. https://developers.googleblog.com/en/data-science-agent-in-colab-with-gemini/. Accessed: 2025.

Grosnit, A., Maraval, A., Doran, J., Paolo, G., Thomas, A., Beevi, R. S. H. N., Gonzalez, J., Khandelwal, K., Iacobacci, I., Benechehab, A., et al. (2024). Large language models orchestrating structured reasoning achieve kaggle grandmaster level. *arXiv preprint arXiv:2411.03562*.

Guo, S., Deng, C., Wen, Y., Chen, H., Chang, Y., and Wang, J. (2024). Ds-agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*.

Hassan, M. M., Knipper, A., and Santu, S. K. K. (2023). Chatgpt as your personal data scientist. *arXiv preprint arXiv:2305.13657*.

Hong, S., Lin, Y., Liu, B., Wu, B., Li, D., Chen, J., Zhang, J., Wang, J., Zhang, L., Zhuge, M., et al. (2024). Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.

Hu, X., Zhao, Z., Wei, S., Chai, Z., Ma, Q., Wang, G., Wang, X., Su, J., Xu, J., Zhu, M., et al. (2024). Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*.

Huang, Q., Vora, J., Liang, P., and Leskovec, J. (2024a). Mlagentbench: Evaluating language agents on machine learning experimentation.

Huang, X., Liu, W., Chen, X., Wang, X., Wang, H., Lian, D., Wang, Y., Tang, R., and Chen, E. (2024b). Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.

IBM (1968). *SPSS Statistics*.

Inala, J. P., Wang, C., Drucker, S., Ramos, G., Dibia, V., Riche, N., Brown, D., Marshall, D., and Gao, J. (2024). Data analysis in the era of generative ai. *arXiv preprint arXiv:2409.18475*.

Inc., S. I. (1976). *SAS Software*.

Institute, M. G. (2011). *Big data: The next frontier for innovation, competition, and productivity*. McKinsey & Company.

Jiang, Z. et al. (2024). AIDE: the Machine Learning CodeGen Agent. https://github.com/WecoAI/aideml. Accessed: 2024-08-29.

Jordan, M. I. and Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260.

Accepted Manuscript

Julius (2025). Julius ai.

jupyterlab (2023). Jupyter-ai. https://github.com/jupyterlab/jupyter-ai.

Kitchin, R. (2014). *The data revolution: Big data, open data, data infrastructures and their consequences*. Sage.

Lai, Y., Li, C., Wang, Y., Zhang, T., Zhong, R., Zettlemoyer, L., tau Yih, S. W., Fried, D., Wang, S., and Yu, T. (2022). Ds-1000: A natural and reliable benchmark for data science code generation.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Li, Z., Zang, Q., Ma, D., Guo, J., Zheng, T., Niu, X., Yue, X., Wang, Y., Yang, J., Liu, J., et al. (2024). Autokaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424*.

Liu, S.-C., Wang, S., Chang, T., Lin, W., Hsiung, C.-W., Hsieh, Y.-C., Cheng, Y.-P., Luo, S.-H., and Zhang, J. (2023). Jarvix: A llm no code platform for tabular data analysis and optimization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 622–630.

Luo, D., Feng, C., Nong, Y., and Shen, Y. (2024). Autom3l: An automated multimodal machine learning framework with large language models. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 8586–8594.

Ma, P., Ding, R., Wang, S., Han, S., and Zhang, D. (2023). Insightpilot: An llm-empowered automated data exploration system. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 346–352.

MathWorks (1984). *MATLAB*.

Microsoft (1985). *Microsoft Excel*.

Microsoft (2013). *Power BI*.

Nejjar, M., Zacharias, L., Stiehle, F., and Weber, I. (2023). Llms for science: Usage for code generation and data analysis. *Journal of Software: Evolution and Process*, page e2723.

OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI (2024). Gpt-4o system card.

Provost, F. and Fawcett, T. (2013). Data science and its relationship to big data and data-driven decision making. *Big data*, 1(1):51–59.

Accepted Manuscript

Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., Yang, F., Dong, H., Zhang, J., Wang, L., et al. (2023). Taskweaver: A code-first agent framework. *arXiv preprint arXiv:2311.17541*.

Shen, Y., Song, K., Tan, X., Li, D., Lu, W., and Zhuang, Y. (2024). Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Steffensen, J. L., Dufault-Thompson, K., and Zhang, Y. (2016). Psamm: A portable system for the analysis of metabolic models. *PLOS Computational Biology*, 12(2):1–29.

Sun, M., Han, R., Jiang, B., Qi, H., Sun, D., Yuan, Y., and Huang, J. (2024). Lambda: A large model based data agent. *arXiv preprint arXiv:2407.17535*.

Trirat, P., Jeong, W., and Hwang, S. J. (2024). Automl-agent: A multi-agent llm framework for full-pipeline automl. *arXiv preprint arXiv:2410.02958*.

Tu, X., Zou, J., Su, W. J., and Zhang, L. (2023). What should data science education do with large language models?

Waller, M. A. and Fawcett, S. E. (2016). Data science, predictive analytics, and big data: A revolution that will transform supply chain design and management. *Journal of Business Logistics*, 37(1):55–62.

Wang, C., Lee, B., Drucker, S., Marshall, D., and Gao, J. (2024a). Data formulator 2: Iteratively creating rich visualizations with ai. *arXiv preprint arXiv:2408.16119*.

Wang, X., Chen, Y., Yuan, L., Zhang, Y., Li, Y., Peng, H., and Ji, H. (2024b). Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Witten, I. H., Frank, E., and Hall, M. A. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B., Zhang, M., Wang, J., Jin, S., Zhou, E., et al. (2023). The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Xie, L., Zheng, C., Xia, H., Qu, H., and Zhu-Tian, C. (2024). Waitgpt: Monitoring and steering conversational llm agent in data analysis with on-the-fly code visualization. *arXiv preprint arXiv:2408.01703*.

Accepted Manuscript

Xie, T., Zhou, F., Cheng, Z., Shi, P., Weng, L., Liu, Y., Hua, T. J., Zhao, J., Liu, Q., Liu, C., et al. (2023). Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*.

Yang, Z., Zhou, Z., Wang, S., Cong, X., Han, X., Yan, Y., Liu, Z., Tan, Z., Liu, P., Yu, D., et al. (2024). Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453*.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. (2024). Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

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

Zhang, C., Li, L., He, S., Zhang, X., Qiao, B., Qin, S., Ma, M., Kang, Y., Lin, Q., Rajmohan, S., et al. (2024). Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*.

Zhang, L., Zhang, Y., Ren, K., Li, D., and Yang, Y. (2023a). Mlcopilot: Unleashing the power of large language models in solving machine learning tasks. *arXiv preprint arXiv:2304.14979*.

Zhang, W., Shen, Y., Lu, W., and Zhuang, Y. (2023b). Data-copilot: Bridging billions of data and humans with autonomous workflow. *arXiv preprint arXiv:2306.07209*.

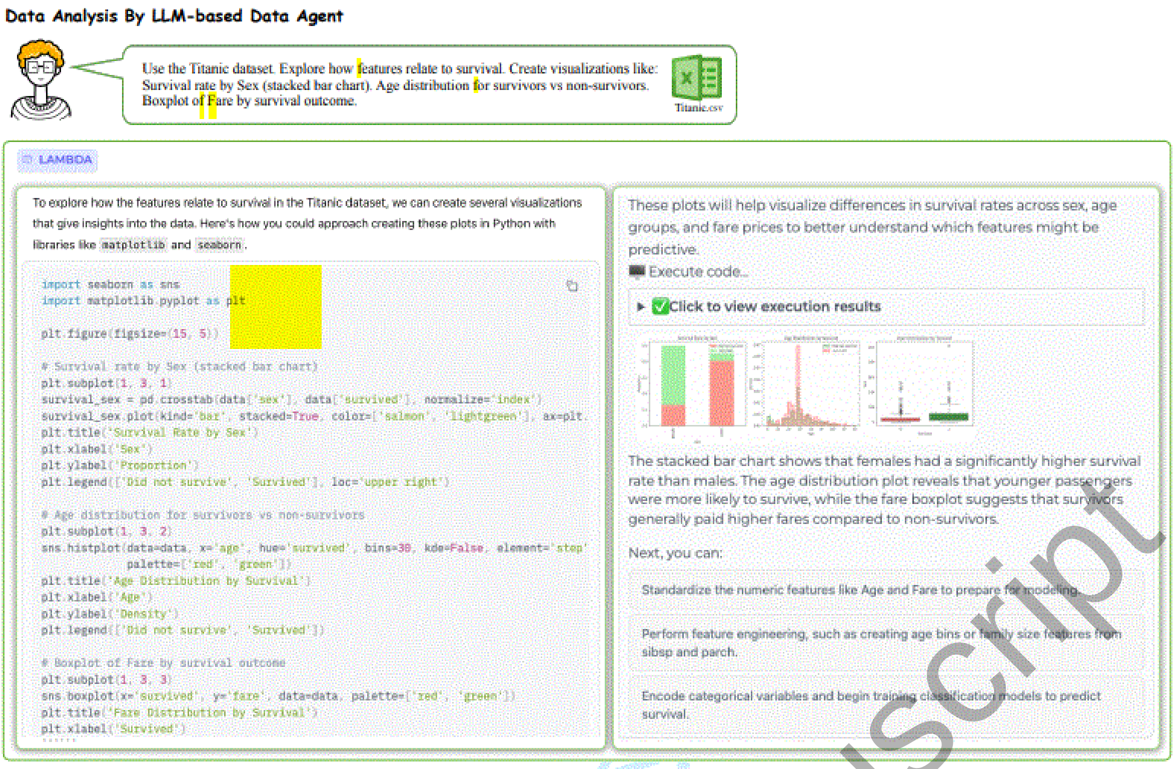
Zhou, X., Zhao, X., and Li, G. (2024). Llm-enhanced data management. *arXiv preprint arXiv:2402.02643*.

Notes

[1] https://aideadlin.es

Figure 1: New paradigm of data analysis brought by generative AI.

Accepted Manuscript


Data Analysis By LLM-based Data Agent
Titanic.csv
LAMBDA
Execute code...
▸ ✅Click to view execution results

Use the Titanic dataset. Explore how features relate to survival. Create visualizations like: Survival rate by Sex (stacked bar chart). Age distribution for survivors vs non-survivors. Boxplot of Fare by survival outcome.

To explore how the features relate to survival in the Titanic dataset, we can create several visualizations that give insights into the data. Here's how you could approach creating these plots in Python with libraries like `matplotlib` and `seaborn`.

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 5))

# Survival rate by Sex (stacked bar chart)
plt.subplot(1, 3, 1)
survival_sex = pd.crosstab(data['sex'], data['survived'], normalize='index')
survival_sex.plot(kind='bar', stacked=True, color=['salmon', 'lightgreen'], ax=plt.
plt.title('Survival Rate by Sex')
plt.xlabel('Sex')
plt.ylabel('Proportion')
plt.legend(['Did not survive', 'Survived'], loc='upper right')

# Age distribution for survivors vs non-survivors
plt.subplot(1, 3, 2)
sns.histplot(data=data, x='age', hue='survived', bins=30, kde=False, element='step'
             palette=['red', 'green'])
plt.title('Age Distribution by Survival')
plt.xlabel('Age')
plt.ylabel('Density')
plt.legend(['Did not survive', 'Survived'])

# Boxplot of Fare by survival outcome
plt.subplot(1, 3, 3)
sns.boxplot(x='survived', y='fare', data=data, palette=['red', 'green'])
plt.title('Fare Distribution by Survival')
plt.xlabel('Survived')
```

These plots will help visualize differences in survival rates across sex, age groups, and fare prices to better understand which features might be predictive.

The stacked bar chart shows that females had a significantly higher survival rate than males. The age distribution plot reveals that younger passengers were more likely to survive, while the fare boxplot suggests that survivors generally paid higher fares compared to non-survivors.

Next, you can:

- Standardize the numeric features like Age and Fare to prepare for modeling.
- Perform feature engineering, such as creating age bins or family size features from sibsp and parch.
- Encode categorical variables and begin training classification models to predict survival.

Figure 2: Timeline of selected related works from 2023.

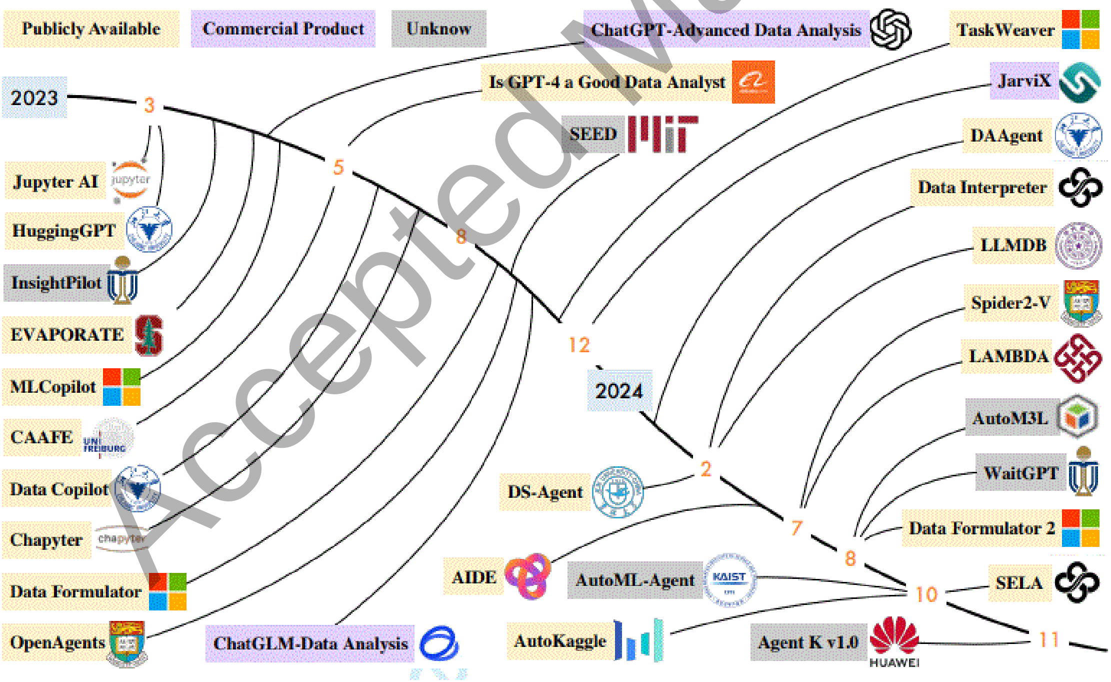


Figure 3: An architecture of an LLM-based data agent. The diagram illustrates the interaction between LLMs and a sandbox environment. On the left, key components of LLMs are highlighted, including User Interface, Planning, Reasoning, Reflection, and Error Handling. The sandbox, positioned centrally, serves as a controlled environment for executing task codes and generating results. On the right, various tools and software that can be pre-installed

Accepted Manuscript

in the sandbox, such as Python, SQL, Jupyter, and R, indicate the diverse ecosystems where LLM-powered agents can operate.

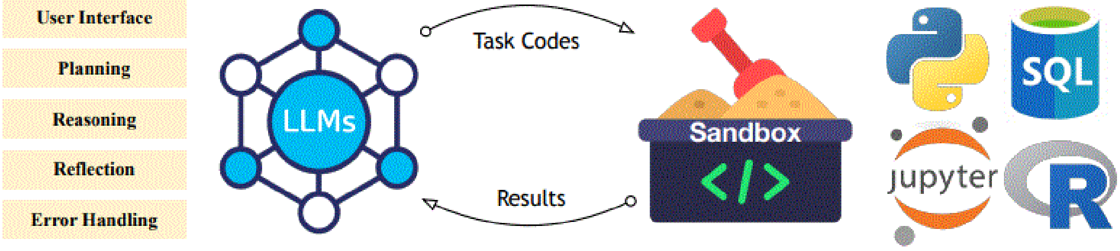


Figure 4: Commonly used planning and reasoning strategies in LLM-based data agents for organizing tasks or solving problems. Each node represents a sub-task in the roadmap.

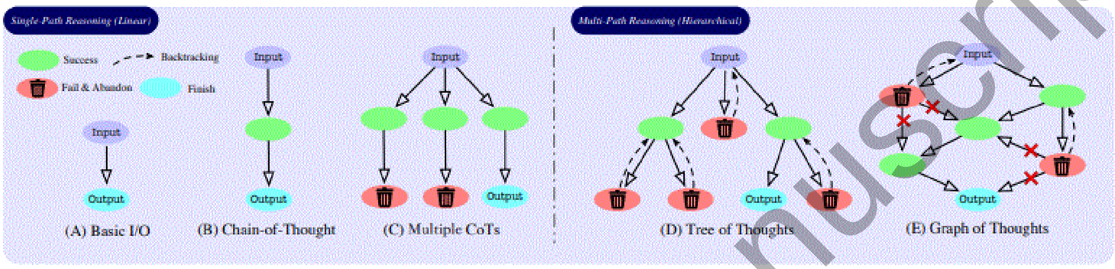
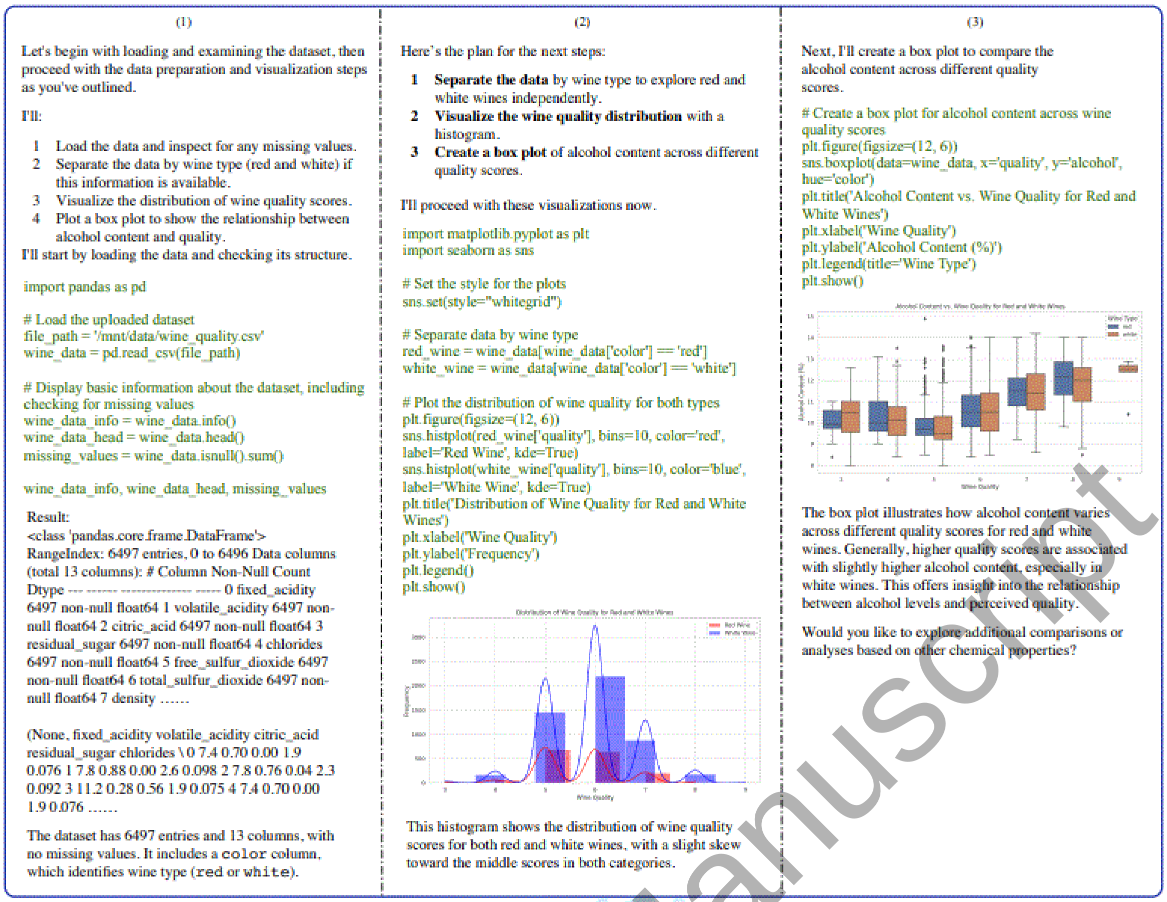


Figure 5: Partial dialogue from the ChatGPT-Advanced Data Analysis in Case Study 1. Items 1-4 list the work done by ChatGPT in each step.

Accepted Manuscript



(1)

Let's begin with loading and examining the dataset, then proceed with the data preparation and visualization steps as you've outlined.

I'll:

1. Load the data and inspect for any missing values.
2. Separate the data by wine type (red and white) if this information is available.
3. Visualize the distribution of wine quality scores.
4. Plot a box plot to show the relationship between alcohol content and quality.

I'll start by loading the data and checking its structure.

```
import pandas as pd

# Load the uploaded dataset
file_path = '/mnt/data/wine_quality.csv'
wine_data = pd.read_csv(file_path)

# Display basic information about the dataset, including checking for missing values
wine_data_info = wine_data.info()
wine_data_head = wine_data.head()
missing_values = wine_data.isnull().sum()

wine_data_info, wine_data_head, missing_values
```

```
Result:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496 Data columns
(total 13 columns): # Column Non-Null Count
Dtype --- ------ -------------- ----- 0 fixed_acidity
6497 non-null float64 1 volatile_acidity 6497 non-
null float64 2 citric_acid 6497 non-null float64 3
residual_sugar 6497 non-null float64 4 chlorides
6497 non-null float64 5 free_sulfur_dioxide 6497
non-null float64 6 total_sulfur_dioxide 6497 non-
null float64 7 density ……

(None, fixed_acidity volatile_acidity citric_acid
residual_sugar chlorides \ 0 7.4 0.70 0.00 1.9
0.076 1 7.8 0.88 0.00 2.6 0.098 2 7.8 0.76 0.04 2.3
0.092 3 11.2 0.28 0.56 1.9 0.075 4 7.4 0.70 0.00
1.9 0.076 ……
```

The dataset has 6497 entries and 13 columns, with no missing values. It includes a `color` column, which identifies wine type (`red` or `white`).

(2)

Here's the plan for the next steps:

1. **Separate the data** by wine type to explore red and white wines independently.
2. **Visualize the wine quality distribution** with a histogram.
3. **Create a box plot** of alcohol content across different quality scores.

I'll proceed with these visualizations now.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style for the plots
sns.set(style="whitegrid")

# Separate data by wine type
red_wine = wine_data[wine_data['color'] == 'red']
white_wine = wine_data[wine_data['color'] == 'white']

# Plot the distribution of wine quality for both types
plt.figure(figsize=(12, 6))
sns.histplot(red_wine['quality'], bins=10, color='red',
label='Red Wine', kde=True)
sns.histplot(white_wine['quality'], bins=10, color='blue',
label='White Wine', kde=True)
plt.title('Distribution of Wine Quality for Red and White
Wines')
plt.xlabel('Wine Quality')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

This histogram shows the distribution of wine quality scores for both red and white wines, with a slight skew toward the middle scores in both categories.

(3)

Next, I'll create a box plot to compare the alcohol content across different quality scores.

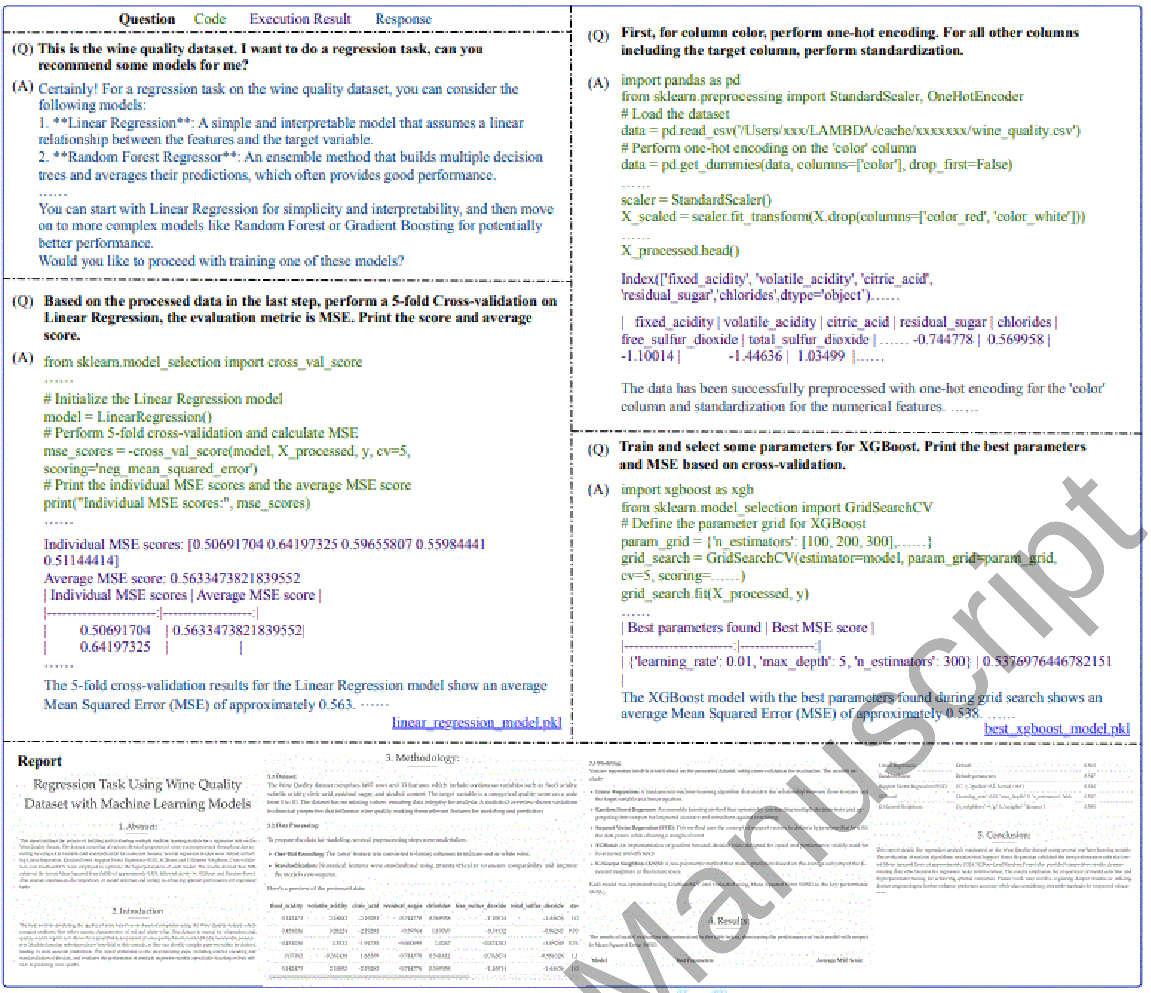
```
# Create a box plot for alcohol content across wine
quality scores
plt.figure(figsize=(12, 6))
sns.boxplot(data=wine_data, x='quality', y='alcohol',
hue='color')
plt.title('Alcohol Content vs. Wine Quality for Red and
White Wines')
plt.xlabel('Wine Quality')
plt.ylabel('Alcohol Content (%)')
plt.legend(title='Wine Type')
plt.show()
```

The box plot illustrates how alcohol content varies across different quality scores for red and white wines. Generally, higher quality scores are associated with slightly higher alcohol content, especially in white wines. This offers insight into the relationship between alcohol levels and perceived quality.

Would you like to explore additional comparisons or analyses based on other chemical properties?

Figure 6: Conversational machine learning and report generation by LAMBDA. Excerpt from a partial dialogue.

Accepted Manuscript



(Q) **This is the wine quality dataset. I want to do a regression task, can you recommend some models for me?**

(A) Certainly! For a regression task on the wine quality dataset, you can consider the following models:
1. **Linear Regression**: A simple and interpretable model that assumes a linear relationship between the features and the target variable.
2. **Random Forest Regressor**: An ensemble method that builds multiple decision trees and averages their predictions, which often provides good performance.
……
You can start with Linear Regression for simplicity and interpretability, and then move on to more complex models like Random Forest or Gradient Boosting for potentially better performance.
Would you like to proceed with training one of these models?

(Q) **Based on the processed data in the last step, perform a 5-fold Cross-validation on Linear Regression, the evaluation metric is MSE. Print the score and average score.**

(A)
```
from sklearn.model_selection import cross_val_score
……
# Initialize the Linear Regression model
model = LinearRegression()
# Perform 5-fold cross-validation and calculate MSE
mse_scores = -cross_val_score(model, X_processed, y, cv=5,
scoring='neg_mean_squared_error')
# Print the individual MSE scores and the average MSE score
print("Individual MSE scores:", mse_scores)
……
```

```
Individual MSE scores: [0.50691704 0.64197325 0.59655807 0.55984441
0.51144414]
Average MSE score: 0.5633473821839552
| Individual MSE scores | Average MSE score |
|----------------------:|------------------:|
|       0.50691704    | 0.5633473821839552|
|       0.64197325    |                   |
……
```

The 5-fold cross-validation results for the Linear Regression model show an average Mean Squared Error (MSE) of approximately 0.563. ……

linear_regression_model.pkl

(Q) **First, for column color, perform one-hot encoding. For all other columns including the target column, perform standardization.**

(A)
```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
# Load the dataset
data = pd.read_csv('/Users/xxx/LAMBDA/cache/xxxxxxx/wine_quality.csv')
# Perform one-hot encoding on the 'color' column
data = pd.get_dummies(data, columns=['color'], drop_first=False)
……
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X.drop(columns=['color_red', 'color_white']))
……
X_processed.head()
```

```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid',
'residual_sugar','chlorides',dtype='object')……

|   fixed_acidity | volatile_acidity | citric_acid | residual_sugar | chlorides |
free_sulfur_dioxide | total_sulfur_dioxide | …… -0.744778 |  0.569958 |
-1.10014 |          -1.44636 |  1.03499  |……
```

The data has been successfully preprocessed with one-hot encoding for the 'color' column and standardization for the numerical features. ……

(Q) **Train and select some parameters for XGBoost. Print the best parameters and MSE based on cross-validation.**

(A)
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
# Define the parameter grid for XGBoost
param_grid = {'n_estimators': [100, 200, 300],……}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
cv=5, scoring=……)
grid_search.fit(X_processed, y)
……
```

```
| Best parameters found | Best MSE score |
|----------------------:|---------------:|
| {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 300} | 0.5376976446782151
|
```

The XGBoost model with the best parameters found during grid search shows an average Mean Squared Error (MSE) of approximately 0.538. ……

best_xgboost_model.pkl

Report

Regression Task Using Wine Quality Dataset with Machine Learning Models

1. Abstract:

2. Introduction

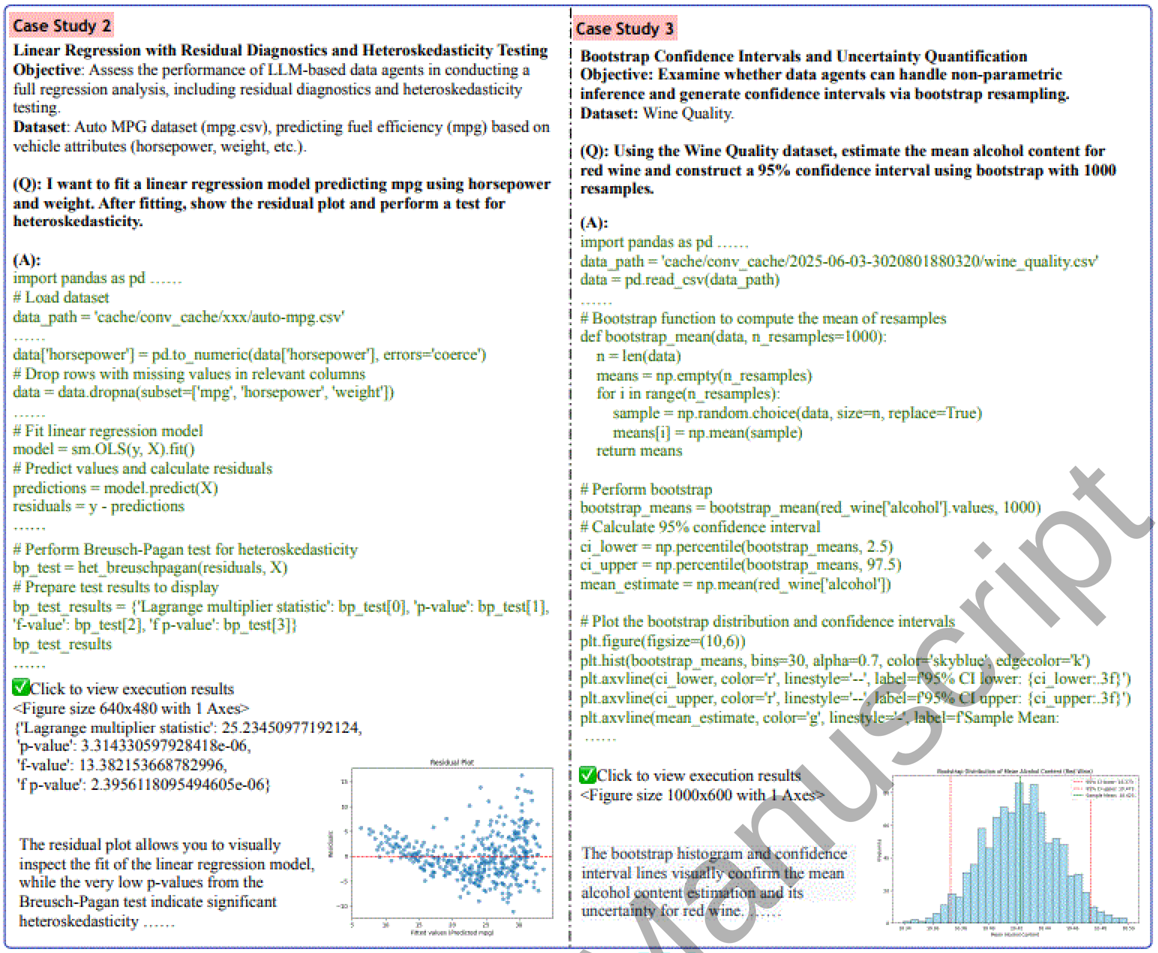
3. Methodology:

4. Results

5. Conclusion:

Figure 7: Partial dialogue from residual diagnostics and heteroskedasticity testing, and bootstrap confidence interval estimation.

Accepted Manuscript



Case Study 2

Linear Regression with Residual Diagnostics and Heteroskedasticity Testing
Objective: Assess the performance of LLM-based data agents in conducting a full regression analysis, including residual diagnostics and heteroskedasticity testing.
Dataset: Auto MPG dataset (mpg.csv), predicting fuel efficiency (mpg) based on vehicle attributes (horsepower, weight, etc.).

(Q): I want to fit a linear regression model predicting mpg using horsepower and weight. After fitting, show the residual plot and perform a test for heteroskedasticity.

(A):

```
import pandas as pd ……
# Load dataset
data_path = 'cache/conv_cache/xxx/auto-mpg.csv'
……
data['horsepower'] = pd.to_numeric(data['horsepower'], errors='coerce')
# Drop rows with missing values in relevant columns
data = data.dropna(subset=['mpg', 'horsepower', 'weight'])
……
# Fit linear regression model
model = sm.OLS(y, X).fit()
# Predict values and calculate residuals
predictions = model.predict(X)
residuals = y - predictions
……
# Perform Breusch-Pagan test for heteroskedasticity
bp_test = het_breuschpagan(residuals, X)
# Prepare test results to display
bp_test_results = {'Lagrange multiplier statistic': bp_test[0], 'p-value': bp_test[1],
'f-value': bp_test[2], 'f p-value': bp_test[3]}
bp_test_results
……
```

✅Click to view execution results

```
<Figure size 640x480 with 1 Axes>
{'Lagrange multiplier statistic': 25.23450977192124,
 'p-value': 3.314330597928418e-06,
 'f-value': 13.382153668782996,
 'f p-value': 2.3956118095494605e-06}
```

The residual plot allows you to visually inspect the fit of the linear regression model, while the very low p-values from the Breusch-Pagan test indicate significant heteroskedasticity ……

Case Study 3

Bootstrap Confidence Intervals and Uncertainty Quantification
Objective: Examine whether data agents can handle non-parametric inference and generate confidence intervals via bootstrap resampling.
Dataset: Wine Quality.

(Q): Using the Wine Quality dataset, estimate the mean alcohol content for red wine and construct a 95% confidence interval using bootstrap with 1000 resamples.

(A):

```
import pandas as pd ……
data_path = 'cache/conv_cache/2025-06-03-3020801880320/wine_quality.csv'
data = pd.read_csv(data_path)
……
# Bootstrap function to compute the mean of resamples
def bootstrap_mean(data, n_resamples=1000):
    n = len(data)
    means = np.empty(n_resamples)
    for i in range(n_resamples):
        sample = np.random.choice(data, size=n, replace=True)
        means[i] = np.mean(sample)
    return means

# Perform bootstrap
bootstrap_means = bootstrap_mean(red_wine['alcohol'].values, 1000)
# Calculate 95% confidence interval
ci_lower = np.percentile(bootstrap_means, 2.5)
ci_upper = np.percentile(bootstrap_means, 97.5)
mean_estimate = np.mean(red_wine['alcohol'])

# Plot the bootstrap distribution and confidence intervals
plt.figure(figsize=(10,6))
plt.hist(bootstrap_means, bins=30, alpha=0.7, color='skyblue', edgecolor='k')
plt.axvline(ci_lower, color='r', linestyle='--', label=f'95% CI lower: {ci_lower:.3f}')
plt.axvline(ci_upper, color='r', linestyle='--', label=f'95% CI upper: {ci_upper:.3f}')
plt.axvline(mean_estimate, color='g', linestyle='-', label=f'Sample Mean:
 ……
```

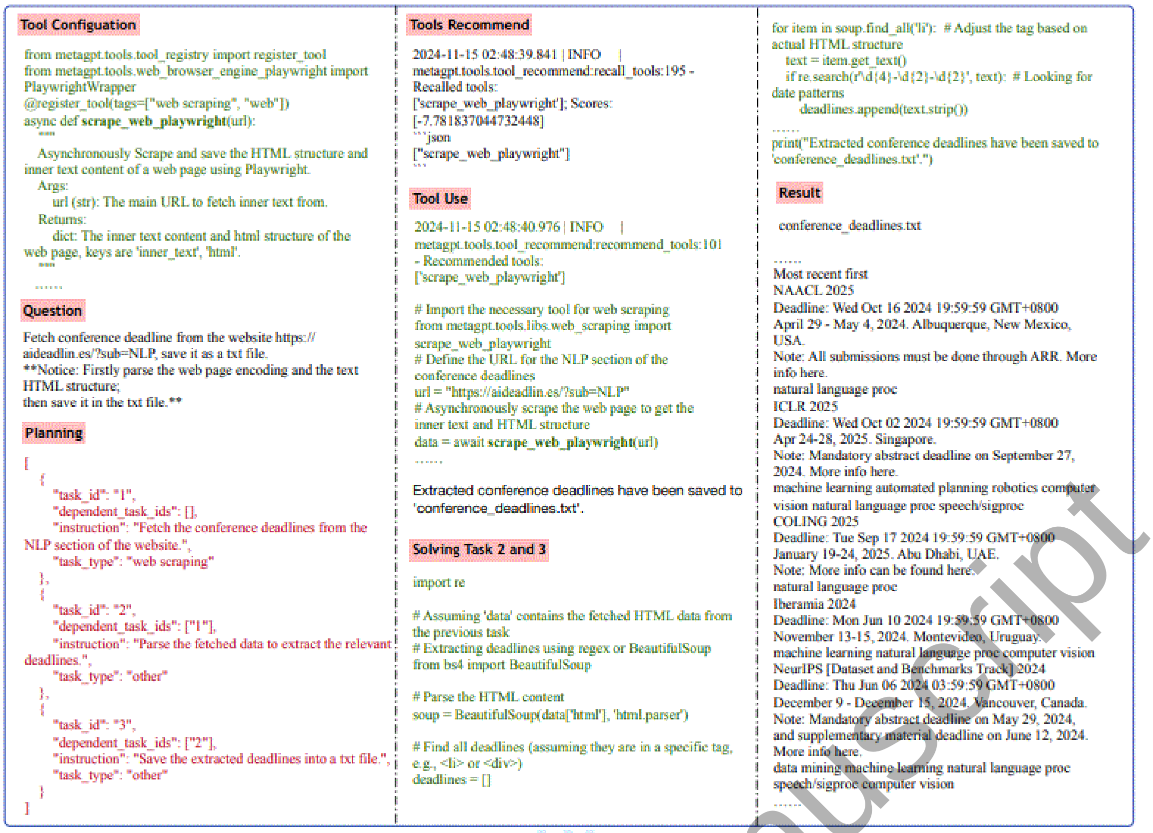
✅Click to view execution results

```
<Figure size 1000x600 with 1 Axes>
```

The bootstrap histogram and confidence interval lines visually confirm the mean alcohol content estimation and its uncertainty for red wine. ……

Figure 8: Creating and using the customized tool in the Data Interpreter. Excerpt from a partial dialogue.

Accepted Manuscript



Tool Configuation

```
from metagpt.tools.tool_registry import register_tool
from metagpt.tools.web_browser_engine_playwright import
PlaywrightWrapper
@register_tool(tags=["web scraping", "web"])
async def scrape_web_playwright(url):
    """
    Asynchronously Scrape and save the HTML structure and
inner text content of a web page using Playwright.
    Args:
        url (str): The main URL to fetch inner text from.
    Returns:
        dict: The inner text content and html structure of the
web page, keys are 'inner_text', 'html'.
    """
   ……
```

Question

Fetch conference deadline from the website https://aideadlin.es/?sub=NLP, save it as a txt file.
**Notice: Firstly parse the web page encoding and the text HTML structure;
then save it in the txt file.**

Planning

```
[
    {
        "task_id": "1",
        "dependent_task_ids": [],
        "instruction": "Fetch the conference deadlines from the
NLP section of the website.",
        "task_type": "web scraping"
    },
    {
        "task_id": "2",
        "dependent_task_ids": ["1"],
        "instruction": "Parse the fetched data to extract the relevant
deadlines.",
        "task_type": "other"
    },
    {
        "task_id": "3",
        "dependent_task_ids": ["2"],
        "instruction": "Save the extracted deadlines into a txt file.",
        "task_type": "other"
    }
]
```

Tools Recommend

~~~
2024-11-15 02:48:39.841 | INFO     |
metagpt.tools.tool_recommend:recall_tools:195 -
Recalled tools:
['scrape_web_playwright']; Scores:
[-7.781837044732448]
```json
["scrape_web_playwright"]
```
~~~

Tool Use

```
2024-11-15 02:48:40.976 | INFO     |
metagpt.tools.tool_recommend:recommend_tools:101
- Recommended tools:
['scrape_web_playwright']

# Import the necessary tool for web scraping
from metagpt.tools.libs.web_scraping import
scrape_web_playwright
# Define the URL for the NLP section of the
conference deadlines
url = "https://aideadlin.es/?sub=NLP"
# Asynchronously scrape the web page to get the
inner text and HTML structure
data = await scrape_web_playwright(url)
……

Extracted conference deadlines have been saved to
'conference_deadlines.txt'.
```

Solving Task 2 and 3

```
import re

# Assuming 'data' contains the fetched HTML data from
the previous task
# Extracting deadlines using regex or BeautifulSoup
from bs4 import BeautifulSoup

# Parse the HTML content
soup = BeautifulSoup(data['html'], 'html.parser')

# Find all deadlines (assuming they are in a specific tag,
e.g., <li> or <div>)
deadlines = []
for item in soup.find_all('li'):  # Adjust the tag based on
actual HTML structure
    text = item.get_text()
    if re.search(r'\d{4}-\d{2}-\d{2}', text):  # Looking for
date patterns
        deadlines.append(text.strip())
……
print("Extracted conference deadlines have been saved to
'conference_deadlines.txt'.")
```

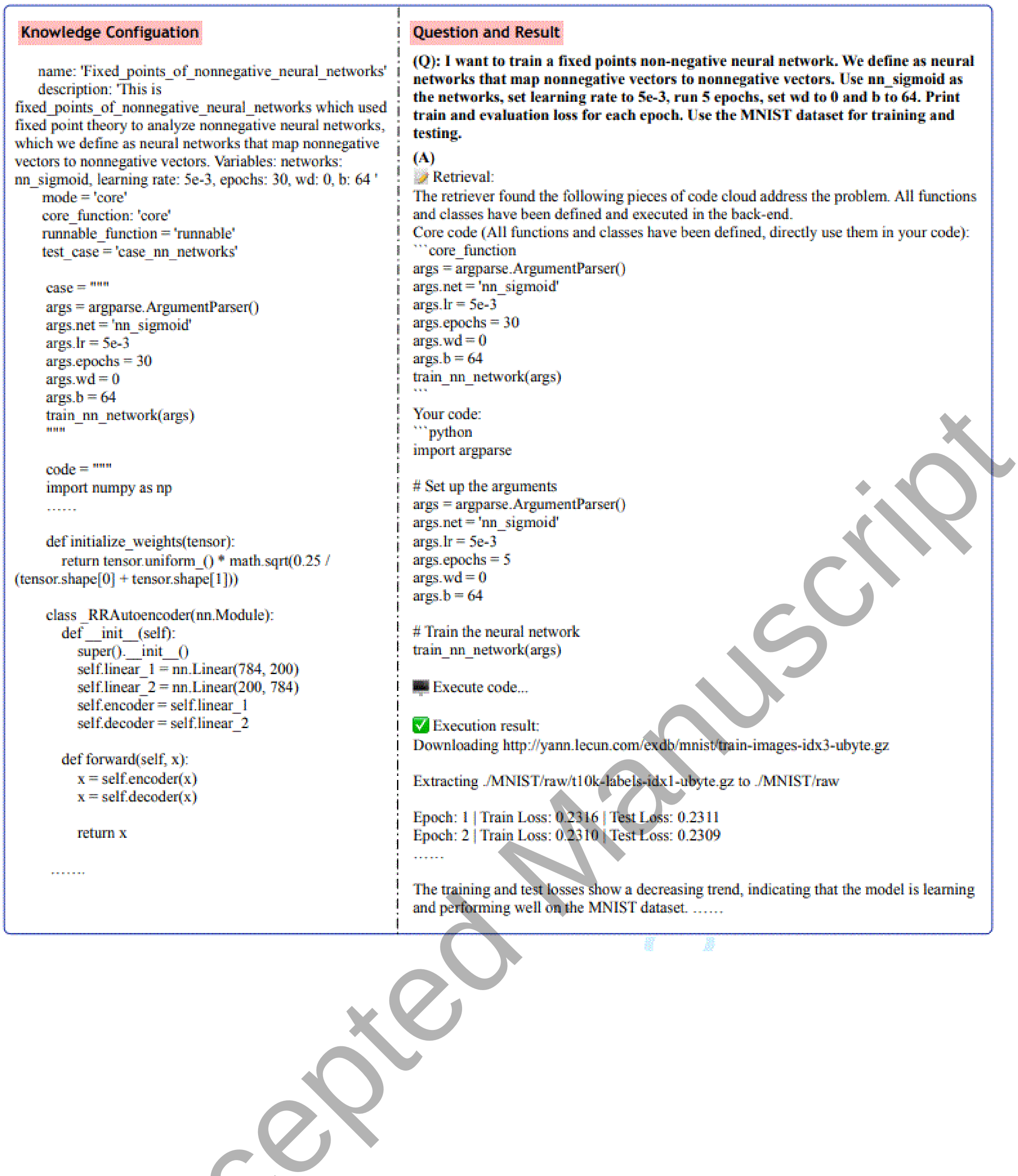
Result

```
 conference_deadlines.txt


……
Most recent first
NAACL 2025
Deadline: Wed Oct 16 2024 19:59:59 GMT+0800
April 29 - May 4, 2024. Albuquerque, New Mexico,
USA.
Note: All submissions must be done through ARR. More
info here.
natural language proc
ICLR 2025
Deadline: Wed Oct 02 2024 19:59:59 GMT+0800
Apr 24-28, 2025. Singapore.
Note: Mandatory abstract deadline on September 27,
2024. More info here.
machine learning automated planning robotics computer
vision natural language proc speech/sigproc
COLING 2025
Deadline: Tue Sep 17 2024 19:59:59 GMT+0800
January 19-24, 2025. Abu Dhabi, UAE.
Note: More info can be found here.
natural language proc
Iberamia 2024
Deadline: Mon Jun 10 2024 19:59:59 GMT+0800
November 13-15, 2024. Montevideo, Uruguay.
machine learning natural language proc computer vision
NeurIPS [Dataset and Benchmarks Track] 2024
Deadline: Thu Jun 06 2024 03:59:59 GMT+0800
December 9 - December 15, 2024. Vancouver, Canada.
Note: Mandatory abstract deadline on May 29, 2024,
and supplementary material deadline on June 12, 2024.
More info here.
data mining machine learning natural language proc
speech/sigproc computer vision
……
```

Figure 9: Integrating knowledge of FPNNNs in LAMBDA. Excerpt from a partial dialogue.

Accepted Manuscript



Knowledge Configuation

~~~
    name: 'Fixed_points_of_nonnegative_neural_networks'
    description: 'This is
fixed_points_of_nonnegative_neural_networks which used
fixed point theory to analyze nonnegative neural networks,
which we define as neural networks that map nonnegative
vectors to nonnegative vectors. Variables: networks:
nn_sigmoid, learning rate: 5e-3, epochs: 30, wd: 0, b: 64 '
     mode = 'core'
     core_function: 'core'
     runnable_function = 'runnable'
     test_case = 'case_nn_networks'

      case = """
      args = argparse.ArgumentParser()
      args.net = 'nn_sigmoid'
      args.lr = 5e-3
      args.epochs = 30
      args.wd = 0
      args.b = 64
      train_nn_network(args)
      """

      code = """
      import numpy as np
      ……

      def initialize_weights(tensor):
         return tensor.uniform_() * math.sqrt(0.25 /
(tensor.shape[0] + tensor.shape[1]))

      class _RRAutoencoder(nn.Module):
         def __init__(self):
            super().__init__()
            self.linear_1 = nn.Linear(784, 200)
            self.linear_2 = nn.Linear(200, 784)
            self.encoder = self.linear_1
            self.decoder = self.linear_2

         def forward(self, x):
            x = self.encoder(x)
            x = self.decoder(x)

            return x

       …….
~~~

Question and Result

(Q): I want to train a fixed points non-negative neural network. We define as neural networks that map nonnegative vectors to nonnegative vectors. Use nn_sigmoid as the networks, set learning rate to 5e-3, run 5 epochs, set wd to 0 and b to 64. Print train and evaluation loss for each epoch. Use the MNIST dataset for training and testing.

(A)

📝 Retrieval:

The retriever found the following pieces of code cloud address the problem. All functions and classes have been defined and executed in the back-end.

Core code (All functions and classes have been defined, directly use them in your code):

~~~
```core_function
args = argparse.ArgumentParser()
args.net = 'nn_sigmoid'
args.lr = 5e-3
args.epochs = 30
args.wd = 0
args.b = 64
train_nn_network(args)
```
~~~

Your code:

~~~
```python
import argparse

# Set up the arguments
args = argparse.ArgumentParser()
args.net = 'nn_sigmoid'
args.lr = 5e-3
args.epochs = 5
args.wd = 0
args.b = 64

# Train the neural network
train_nn_network(args)
~~~

💻 Execute code...

✅ Execution result:

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz

Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw

Epoch: 1 | Train Loss: 0.2316 | Test Loss: 0.2311
Epoch: 2 | Train Loss: 0.2310 | Test Loss: 0.2309
……
```

The training and test losses show a decreasing trend, indicating that the model is learning and performing well on the MNIST dataset. ……

Accepted Manuscript

Table 1: Characteristics of selected data agents. Methods can be categorized into Conversational and End-to-End approaches. Conversational methods support interactive dialogue with iterative user feedback, whereas End-to-End approaches rely on a single prompt, with the agent autonomously planning and solving the problem. The user interface can be categorized into IDE-based, Systems, CLI, and OS-based. The term “Human-in-the-Loop” indicates that humans can intervene in the data agent’s workflow, such as modifying code in situations where automatic processes are inadequate. “Self-Correcting” refers to the agent’s ability to automatically identify and correct errors within the workflow through reflection. Finally, “Expandable” denotes the data agent’s capacity to incorporate customized tools or knowledge. “-” indicates that the attribute is either not mentioned in the paper or could not be observed from the provided resources.

Data Agents	Methods	User Interface	Planning	Human in the Loop	Self-correcting	Expandable
ChatGPT-ADA (OpenAI, 2023)	Conversational	System	Linear	✗	X	✗
Data Copilot (Zhang et al., 2023b)	End-to-end	System	Linear	✗	X	✗
Jupyter AI (jupyterlab, 2023)	Conversational	IDE-based	Basic IO	X	✗	✗
MLCopilot (Zhang et al., 2023a)	Conversational	IDE-based	Basic IO	X	✗	✗
Chapyter (chapyter, 2023)	Conversational	IDE-based	Basic IO	X	✗	✗
Openagents (Xie et al., 2023)	Conversational	System	Linear	✗	✗	X
JarviX (Chen et al., 2024)	End-to-end	-	-	-	-	-
DS-Agent (Guo et al., 2024)	End-to-end	CLI	Linear	✗	X	-
Spider2-V (Cao et al., 2024)	End-to-end	OS-Based	-	✗	X	-
ChatGLM-DA (GLM, 2024)	Conversational	System	Linear	✗	X	✗
TaskWeaver (Qiao et al., 2023)	End-to-end	CLI & System	Linear	✗	X	X
Data Interpreter (Hong et al., 2024)	End-to-end	CLI	Hierarchical	X	X	X
LAMBDA (Sun et al., 2024)	Conversational	System	Basic IO	X	X	X
Data Formulator 2 (Wang et al., 2024a)	Conversational	System	Basic IO	✗	X	-
AutoM3L (Luo et al., 2024)	End-to-end	-	-	✗	-	X
SELA (Chi et al., 2024)	End-to-end	CLI	Hierarchical	✗	X	-
AIDE (Jiang et al., 2024)	End-to-end	CLI	Hierarchical	✗	X	-
AutoKagle (Li et al.,	End-to-end	CLI	Linear	X	X	X

Accepted Manuscript

2024)						
AutoML-Agent (Trirat et al., 2024)	End-to-end	-	Linear	-	X	-
Agent K v1.0 (Grosnit et al., 2024)	End-to-end	-	Linear	-	X	✗
GPT-4o (OpenAI, 2024)	End-to-end	System	-	✗	X	X
AutoGen Studio (Wu et al., 2023)	End-to-end	System	Linear	✗	X	X
Colab Data Science Agent (Google, 2025)	End-to-end	IDE-based	Linear	X	X	✗

Accepted Manuscript