Data-Copilot: Bridging Billions of Data and Humans with Autonomous Workflow

Anonymous ACL submission

Abstract

Industries such as finance, meteorology, and energy generate vast amounts of heterogeneous data daily. Efficiently managing, processing, and visualizing such data is labor-intensive and frequently necessitates specialized expertise. Leveraging large language models (LLMs) to develop an automated workflow presents a highly promising solution. However, LLMs are not adept at handling complex numerical computations and table manipulations, and they are further constrained by a limited length context. To bridge this, we propose Data-Copilot, a data analysis agent that autonomously performs data querying, processing, and visualization tailored to diverse human requests. The advancements are twofold: First, it is a **code-centric agent** that leverages code as an intermediary to process and visualize massive data based on human requests, achieving automated large-scale data analysis. Second, Data-Copilot involves a **data exploration** phase in advance, which autonomously explores how to design universal and error-free interfaces from data, reducing the error rate in real-time responses. Specifically, It imitates common requests from data sources, abstracts them into universal interfaces (code modules), optimizes their functionality, and validates effectiveness. For real-time requests, Data-Copilot invokes these interfaces to address user intent. Compared to generating code from scratch, invoking these pre-designed and well-validated interfaces can significantly reduce errors during real-time requests. We open-sourced Data-Copilot with massive Chinese financial data, such as stocks, funds, and news. Quantitative evaluations indicate that our exploration-deployment strategy addresses human requests **more accurate** and **efficiently**, with good interpretability.

1 Introduction

In the real world, vast amounts of heterogeneous data are generated every day across various industries, including finance, meteorology, and energy, among others. Humans have an inherent and significant demand for data analysis because these wide and diverse data contain insights that can be applied to numerous applications, from predicting financial trends to monitoring energy consumption. However, these data-related tasks often require tedious manual labor and specialized knowledge.

Recently, the advancement of large language models (LLMs) (Zeng et al., 2023; Touvron et al., 2023; OpenAI, 2022) and techniques (Wei et al., 2022b; Kojima et al., 2022) have demonstrated the capability to handle complex tasks. Given the vast amounts of data generated daily, **can we leverage LLMs to create an automated data processing workflow performing data analysis and visualization** that best matches user expectations?

An intuitive solution is to treat data as a special text, i.e., directly use LLMs to read and process massive data (Wu et al., 2023c; Zha et al., 2023). However, as shown in Figure 1, several challenges must be considered: (1) Due to context limitations of LLMs, it is challenging for LLMs to directly read and process massive data as they do with text. Besides, it also poses the potential risk of data leakage when LLMs directly access private data sources. (2) Data processing is complex, involving many tedious numerical calculations and intricate table manipulations. LLMs are not adept at performing these tasks. (3) Data analysis typically requires visualizing the results of data processing, whereas LLMs are limited to generating text output. These challenges constrain the application of LLMs in data-related tasks.

Recently, many agent-based designs have explored alternative solutions (Wu et al., 2023a; Huang et al., 2023; Chen et al., 2023b; Hong et al., 2023; Wu et al., 2023b; Nejjar et al., 2023; Li et al., 2024b). LiDA (Dibia, 2023) and GPT4-Analyst (Cheng et al., 2023) focus on exploring insight from data. Sheet-Copilot (Li

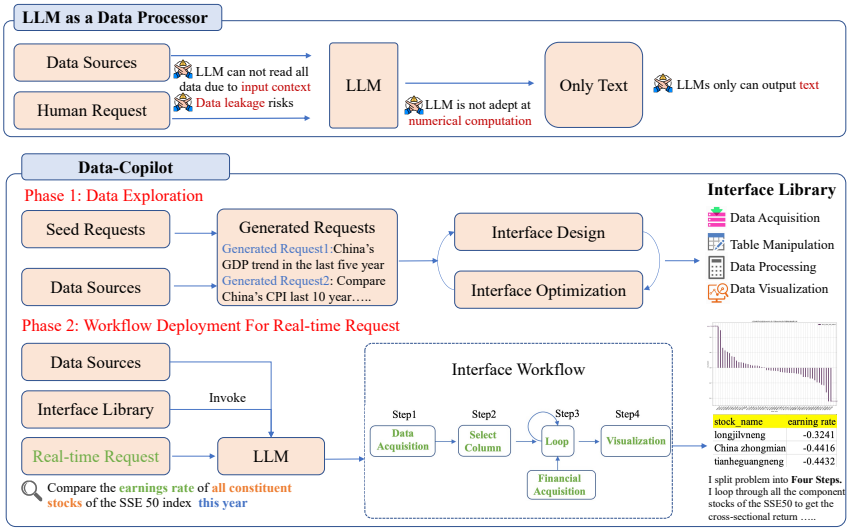


Figure 1: We compare two LLM strategies for automated data analysis. Upper: LLM's capabilities, context length, and output format constraints limit the use of LLMs to process massive data. Bottom: Data-Copilot is a code-centric agent that utilizes code to handle extensive data analysis tasks. It explores how to design more universal and error-free interface modules, improving the success rate of real-time requests. Faced with real-time requests, Data-Copilot invokes self-design interfaces and constructs a workflow for human intent.

et al., 2023b), BIRD (Li et al., 2023c), DS-Agent (Guo et al., 2024b), DB-GPT (Xue et al., 2023) and TAG (Biswal et al., 2024) apply LLMs to data science domain like Text2SQL. Data Interpreter (Hong et al., 2024) proposes a plan-code-verify paradigm for automating machine learning tasks. These methods showcase the potential of LLMs in completing complex daily tasks through agent design paradigms.

Inspired by this, we advocate leveraging the coding capabilities of LLM to build a data analysis agent. Acting like a human data analyst, it receives human requests and generates code as an intermediary to process massive data and visualize its results (e.g., chart, table, text) for humans. However, creating a code agent that can be used in real-world data analysis tasks is far from an easy feat. ① LLMs struggle to generate high-quality, error-free code in a single attempt, often containing format errors, logical inconsistencies, or fabricating non-existent functions. ② Although the inference speed of LLMs has significantly improved, generating lengthy code still consumes a considerable amount of time and tokens. These two challenges—**high code error rate** and **inefficient inference**—must be addressed urgently.

To address this, we observe most human requests are either similar or inherently related. By abstracting common demands into interfaces and validating their functionality in advance, we can significantly improve both the success rate and efficiency of real-time deployment. Therefore, we propose **Data-Copilot**, an LLM-based agent with an innovative exploration phase to achieve more reliable data analysis. First, Data-Copilot is **a code-centric agent** that connects massive data sources and generates code to retrieve, process, and visualize data in a way that best matches user's intent. The code-centric design empowers it to efficiently and securely handle extremely large-scale data and nearly all types of data analysis tasks. Besides, Data-Copilot also incorporates **a data exploration and interface design phase**. It autonomously explores how to design more universal and error-free interfaces (code modules) based on data schemas in advance. In real-world deployment, Data-Copilot flexibly invokes pre-design interface modules for most requests, deploying a well-verified interface workflow for data processing and visualization.

Data-Copilot brings three advantages. Firstly, this exploratory process allows Data-Copilot to analyze and summarize the inherent connections between human requests, design general interfaces for similar requests, and pre-validate their correctness, **reducing errors in real-time responses**. Secondly, when faced with massive requests, our agent only needs to invoke these pre-designed interfaces rather than generate redundant code, significantly **improving inference efficiency**. Lastly, compared to lengthy code, these interfaces provide **greater interpretability**, since it is easier for human reading and interaction. To achieve this, we contains

three steps (Figure 1) when self exploration:

Explore data and Synthesize requests: Data-Copilot discovers potential requests and a broader range of human need from data. It involves a "self-exploration" process to generate massive requests based on all data schemas and seed requests.

Interface Design and Test: It designs modular interface from synthesized requests, with test cases automatically generated for verification.

Interface Optimization: To improve versatility, it merges similar interfaces and also revises erroneous interfaces using compiler's feedback.

After **exploration-design-optimization**, Data-Copilot designs many general and error-free interfaces, e.g., data acquiring, forecasting, and visualizing modules, to accomplish data analysis tasks. When faced with real-time requests, Data-Copilot invokes these predefined interface modules to create a concise interface workflow for user requests. For different requests, Data-Copilot can flexibly deploy various invocation structures, such as step-by-step serial workflows, parallel, or loop. It can even output a hybrid form of interface workflows and raw code for these "unfamiliar" requests. Our contributions are threefold:

- We propose a code-centric agent, Data-Copilot, for automated data analysis and visualization. It leverages LLM's code generation abilities for data querying, processing, and visualization, reducing tedious human labor.
- We decouple code generation into two phases: data exploration and workflow deployment. In exploration phase, Data-Copilot learns to design universal, error-free interface modules tailored to data. In the face of real-time requests, it flexibly invokes these interfaces to address users' diverse requests. This design enhances the success rate and efficiency of real-time responses.
- We open-source Data-Copilot for Chinese financial data analysis, including stocks, funds, and live news. Quantitative evaluations indicate our agent outperforms other strategies, with **higher success rates and lower token consumption**. Besides, interface workflows are more convenient for human inspection and interaction, offering **interpretability**.

2 Data-Copilot

Data-Copilot is a code-centric agent capable of performing data analysis and visualization based on human instructions. It operates in two phases: Data Exploration and Workflow Deployment. In the first phase (Section 2.1), Data-Copilot designs a self-exploration process to discover numerous potential needs based on data schemas (Step 1: exploration). Then based on these synthesized requests, it abstracts many universal interface modules (Step 2: design). After that, it optimizes similar interfaces and tests their correctness, ensuring each interface is correct and universal (Step 3: testing and optimization). The whole process is operated in advance, yielding many generic, error-free interfaces for subsequent use.

When faced with real-time requests (Section 2.2), Data-Copilot can flexibly invoke predesigned interfaces or directly generate raw code based for the user's request. In most cases, existing interfaces can cover the majority of real-world requests, significantly enhancing both success rate and response speed. We provide a detailed prompt for two phases in Appendix E.3 and E.4.

2.1 Data Exploration

Let's review how human data analysts operate. Initially, they need to observe the available data, understand the data formats, and learn how to access them. Subsequently, humans often design generic modules to simplify the code logic and test these modules for usability. Similar to this, Data-Copilot also autonomously explores data and derives insights from vast data sources, including the relations between the data, and the potential requests associated with the data. Then Data-Copilot abstracts these exploratory insights into numerous reusable code components (interfaces), testing their correctness and optimizing their generality. This process of exploring data, identifying common requests, designing general interfaces, and testing and optimizing their performance is conducted in advance on its own.

Self-Exploration Request. To explore data and mine insights, we design a self-exploration process. Beginning with some seed requests collected from humans, LLMs are prompted to read the data and generate a large number of requests, each representing a potential demand scenario. This process is similar to (Wang et al., 2022a; Dibia, 2023), but the LLMs should generate requests specifically based on provided data. As shown in Figure 2, when the LLMs observe that the economic database contains historical GDP and CPI data, they generate multiple related requests, e.g., `Compare the CPI of`

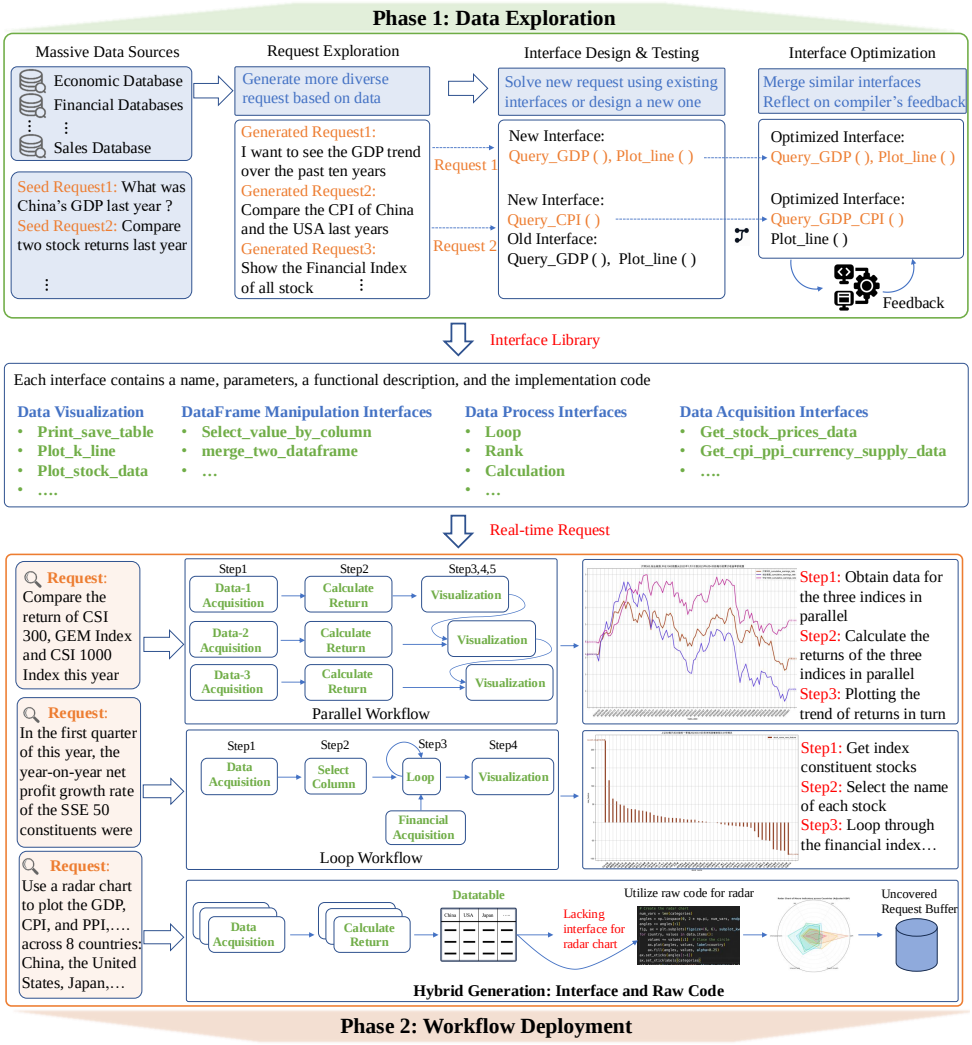


Figure 2: Overview of Data-Copilot. **Data Exploration:** First, it performs a self-exploration process to uncover potential human requests from data sources. Then it abstracts many universal and error-free interfaces from synthesized requests, including interface designing, testing, and optimizing similar interfaces. This exploration-design-optimization process is operated in advance. **Workflow Deployment:** Upon receiving real-time requests, Data-Copilot invokes existing interfaces and deploys a workflow for familiar requests, or flexibly combines interfaces and raw code for "uncovered" requests.

`China and the USA...`, or `I want to see the GDP trend over the past ten years`. Each request involves one or more types of data.

To achieve this, we first generate a parsing file for each data source to help LLM understand the data. Each file includes a description of the data, the access method, the data schema (the name of each column), and a usage example. Then we feed the parsing files and a few seed requests into the LLMs, and prompt LLMs to synthesize more diverse requests based on these data. The brief example is shown in Appendix E.2.

The quality of the generated requests. A common issue is that the LLM often proposes a request about the data that doesn't exist. To address this, we design a backward verification strategy to check these synthesized requests. Specifically, we instruct the LLM to reverse-convert the generated request into the desired data source and other key information, and then we verify the existence of such data, thereby filtering out hallucinatory requests. Besides, when synthesizing, we also use keywords to control the topics of synthesized requests, ensuring they closely align with real-world distribution. Upon manual evaluation of the synthesized requests, we found that the generated requests generally met our expectations. We discuss the quality of synthesized requests in detail in Tables 1 and B2.

Interface Design. After generating massive requests through self-exploration, Data-Copilot ab-

stracts many universal interfaces from these requests. First, we have to clarify what an interface is in our paper. Similar to human-defined functions, an interface is a code module consisting of a name, parameters, a functional description, and an implementation code. It performs specific tasks such as data retrieval, computation, and visualization. Each interface is designed, tested, and optimized iteratively by Data-Copilot.

First, starting from the initial request, we iteratively feed synthesized requests from the self-exploration stage and related data parsing files into LLMs, prompting the LLM to design complete code modules for request solving. Each code module is defined as an interface and stored in the interface library. During each iteration, LLMs are instructed to prioritize utilizing the existing interfaces within the library. If the available interfaces are insufficient for request solving, the LLMs design a new interface.

As shown in Figure 2, for the first request: `I want to see the GDP trend over ...`, Data-Copilot design two interfaces: `Query-GDP()` and `Plot-Line()`. As for the second request: `Compare the CPI of China and the USA over..`, since the previous two could not solve this request, a new interface, `Query-CPI()`, is designed by LLMs.

Interface Testing. After designing a new interface for a request ($request_i \rightarrow interface_i$), Data-Copilot autonomously tests its correctness based on compiler feedback. First, Data-Copilot uses the $request_i$ as the seed to generate massive similar requests as test cases. Then it tests the new $interface_i$ one by one. In Figure 2, for new $interface_i$: `Query-GDP()`, Data-Copilot mimics $request_i$ to generate many similar requests to test the `Query-GDP()`: `"I want to see USA's GDP over the past 10 years", "I want to see China's GDP for the last year",..` If the interface passes all test cases, it is retained; otherwise, Data-Copilot self-reflects on error feedback of compilers, correcting the erroneous code snippets until it successfully passes the tests.

Interface Optimization. To optimize the generality of designed interfaces, Data-Copilot also merges similar interfaces. Similar to human developers, each time a new interface is designed, this optimization process is triggered: evaluating whether the newly designed interface can be merged with previous ones.

- **Retrieve similar interfaces**. After a new interface is designed, we retrieve *Top-N* interfaces from existing interface library. Specifically, we use gte-Qwen1.5-7B-instruct (Li et al., 2023d) to obtain embeddings of each interface code and then calculate their similarity, identifying top N similar interfaces.
- **Decide whether to merge**. LLMs are prompted to compare new interface with *Top-N* retrieved interfaces in terms of functionality, parameters, and processing logic, autonomously deciding whether to merge it with the existing ones. If LLM deems no merging necessary, new interface is retained. As shown in Figure 2, two interfaces `Query-GDP()` and `Query-CPI()` are merged into `Query-GDP-CPI()`. This process makes each interface more general and unique.
- **Test the optimized interface**. When two interfaces are merged, Data-Copilot also needs to test the newly merged interface. Test cases from two original interfaces are used to validate the merged interface. The output of the new interface must be consistent with two original interfaces in both format and content.

In this phase, Data-Copilot alternates the above three steps: interface design, testing, and optimization until all the requests can be covered by these interfaces. As shown in Figure F3, Data-Copilot designs many interfaces for different task and operation types, e.g., data acquisition, prediction, visualization, and DataFrame manipulation. We provide detailed algorithm processes (Appendix E.1) and examples in Appendix F.1.

2.2 Workflow Deployment

As Figure 2 shows, it accurately and efficiently handles requests by invoking relevant interfaces (interface workflow). For long-tail or uncovered requests, it flexibly generates raw code (interface-code hybrid strategy) while documenting these for future interface library updates.

Interfaces Retrieval. Considering the significant differences between the interfaces involved in different requests, it is unnecessary to load all interfaces every time. We design a simple yet efficient interface retrieval strategy: hierarchical retrieval. Specifically, we organize all designed interfaces in a hierarchical structure. Each interface is grouped into different tasks (stock task, fund task, etc.) and different operation types (data acquisition, processing, visualization, etc.). Upon receiving a real-time

request, Data-Copilot first determines the appropriate task types and required operation types and then loads the interfaces associated with these types for subsequent workflow planning. This can reduce the number of interfaces in the prompt.

Interface Invocation Workflow. After reading interface descriptions, Data-Copilot plans workflows—multiple interfaces in specific order forming chain, parallel, or loop structures. It determines which interfaces to invoke, their sequence, and parameters, outputting in `JSON` format. Prompts and cases appear in Appendices E.4 and F.2.

As Figure 2 shows, Data-Copilot designs sequential, parallel, or loop interface workflows. For request `"Compare the return of CSI-300, GEM and CSI-1000 this year"`, it plans a parallel workflow: `Data Acquisition()`, `Calculate Return()` for three indices simultaneously, then `Visualization()`. The second case implements a loop workflow using `Loop()`.

Interface-Code Hybrid Generation. In the real world, it is inevitable to encounter "uncovered" requests that cannot be addressed by existing interfaces. As a remedy, Data-Copilot adopts an interface-code hybrid generation strategy. Specifically, it prioritizes invoking existing interfaces to resolve user requests. It can solve most common user requests. However, if the deployed workflow continues to fail, or if Data-Copilot proactively determines that current request cannot be resolved by existing interfaces, it would generate raw code directly or generate a combination of raw code and interfaces. This design endows Data-Copilot with the flexibility to handle diverse requests.

Besides, each time Data-Copilot encounters such "uncovered" requests, it also records them in the document. After accumulating sufficient new requests, we re-initiated the Data Exploration and interface design stage (Section 2.1), i.e., in real-world interactions, we periodically develop new interfaces for emerging demands, continuously updating Data-Copilot's interface library.

3 Dataset Synthesis

3.1 Environment and Data Sources

Data-Copilot is developed on Chinese financial market data, encompassing massive stocks, funds, economic data, real-time news, and company financial data. Similar to many works, Data-Copilot utilizes data interfaces provided by Tushare[1] to access vast amounts of financial data, including time-series data spanning over 20 years for more than 4,000 stocks, funds, and futures. In the first phase, we use a strong LLM (e.g., `gpt-4o`) for data exploration and a lightweight LLM (e.g., `gpt-3.5-turbo`) for workflow deployment.

[1] https://tushare.pro/

3.2 The Creation of Dataset

Human-proposed Request as Seed Set. We invite 10 students in economics to submit 50 requests each. These requests cover a wide range of common needs, including stock, fund tasks with different complexity levels. These human-proposed requests can represent a distribution of real-world scenarios, where the ratio of four tasks is: Stock(8), Fund(4), Corporation(4) and Others(1). Then we filter out highly similar requests. Lastly, we retain 173 high-quality requests, which are used as seeds for Data Exploration phase.

Self-Exploration Request Set. As mentioned in Section 2.1, we employ self-exploration to expand the request set. We first feed 173 human-proposed seed requests along with all data descriptions and schema into GPT-4 for data exploration and request synthesis. We use keywords to control the distribution of the synthesized requests, ensuring close alignment with the real world. When exploration, it generates 6480 requests. Then we filter out highly similar requests and retain 3547 requests. Then we adopt a stratified sampling strategy to sample 547 instances as test set from each type. The remaining 3000 requests are used for interface design. The distributions of four task types (stock, fund, corporation, and others) and three complexity levels (single-entity, multi-entities, and multi-entities with complex relations) are shown in Table 1. It shows that synthesized requests closely align with humans. The detailed statistics are shown in Tables 1 and B1 and Figure B1.

Annotate Answer Table for Test Set. Human annotators are instructed to annotate data tables as answers for 547 test requests. Specifically, annotators manually retrieve and process the corresponding data based on testing request, and record the final data table before chart plotting. For example, request: `"I want to see China's GDP over past 5 years"`, the labeled data-table is *[2023: 17.8 trillion, 2022: 17.9 trillion, 2021: 17.8 trillion, 2020: 14.6 trillion, 2019: 14.3 trillion]*.

The quality of self-exploration requests. We manually evaluate the quality of testset. As shown in Appendix B.3, these synthesized requests show

Name	Source	#Cases	Form	Four Types Ratio	Three Complexity Levels Ratio
Seed Set	Human-proposed	173	Query	8.0: 4.0: 4.0: 1	2.0: 1.5: 1
Set for Interface Design	Self-Exploration	3000	Query	8.7: 3.3: 4.2: 1	1.9: 1.5: 1
Test Set	Self-Exploration & Human	547	Query, Label	8.6: 3.3: 4.3: 1	1.8: 1.4: 1

Table 1: Statistics on human-proposed requests and self-exploration dataset. We report the number of each task type and complexity level. The results indicate that our synthesized requests closely align with human distribution.

Model	GPT-4o	GPT-4-turbo	GPT-3.5-turbo	Llama3-70B-Instruct	Llama2-70B-Instruct
Direct-Code	73.6	70.0	28.5	52.2	29.6
Ours	**77.9** ↑4.3	**74.6** ↑4.6	**70.2** ↑41.7	**70.9** ↑18.7	**49.3** ↑19.7
Model	Codellama-13B	Vicuna-13B-v1.5	DeepSeek Coder-V2	Qwen2.5-Coder-32B	Qwen2.5-Coder-14B
Direct-Code	21.0	17.3	63.8	66.5	51.7
Ours	**50.8** ↑29.2	**33.2** ↑15.9	**75.3** ↑11.5	**77.2** ↑10.7	**72.3** ↑20.6

Table 2: `Accuracy` of ours and direct code generation (Direct-Code) using different LLMs for workflow deployment.

a comparable quality to human-proposed requests.

3.3 Qualitative Evaluation

As shown in Figure F9, Data-Copilot constructs the invocation workflow step-by-step (each step corresponds to one or more interfaces) and final results (bar, chart, text) for input request.

Data-Copilot designs versatile interfaces via data exploration. After analyzing Chinese financial data and 3,000 self-exploration requests, Data-Copilot created 73 interfaces across five functionalities (data acquisition, index calculation, table manipulation, visualization, general processing). Figure F3 shows key interfaces.

Data-Copilot adapts to new requests by combining interfaces and raw code. As in Section 2.2, for unsupported requests, Data-Copilot integrates existing interfaces with raw code. In Figure 2 (3rd example), it retrieves data via interface, then generates radar chart code, enhancing flexibility for evolving demands.

Interface Workflow enhances interpretability. After execution, Data-Copilot generates visuals and workflow summaries. As shown in Figures 2, F2 and F4 to F7, outputs are intuitive. Structuring complex code into step-by-step interface calls improves clarity and inspection ease.

4 Quantitative Evaluation

4.1 Experiments Settings

Baselines and Experiments Details We compare Data-Copilot with direct code generation (Direct-Code), ReAct, Reflexion, Multi-agent methods. Detailed prompts are in Appendices B.1 and E.5.

Evaluation Methods. We evaluate all methods on 547 test cases, focusing on three aspects: data-table accuracy, image quality, and inference efficiency. Evaluations are conducted using GPT-4o: **Data-Table Evaluation**: GPT-4o/turbo compares the generated dataframe with a human-labeled answer table (Section 3.2). **Image Evaluation**: To evaluate whether the final image meets human requests, we feed the human request, human-annotated answer table, and the generated image into `GPT-4o`. `GPT-4o` is instructed to check image's visual elements based on human request, including numerical points, lines, axes, image aesthetics, and style, i.e., we design a checklist containing 5 categories with 10 sub-dimensions for GPT-4o scoring. The results of data-table and image evaluation are combined as `Accuracy`. **Efficiency**: Measured by the total token consumption (`#Token`) per method, representing solving efficiency. The evaluation details and comparison between manual evaluation and model evaluation are shown in Appendix B.4.

Methods	Accuracy(%)	#Token
Direct-Code	28.5 ±2.1	823
ReAct (Yao et al., 2022)	44.1 ±3.4	1515
Reflexion (Shinn et al., 2023)	59.0 ±3.9	2463
Multi-Agent (Hong et al., 2023)	57.4 ±1.8	2835
Data-Copilot	**70.2** ±2.5	**561.2**
Data-Copilot + ReAct	71.5 ±1.7	834
Data-Copilot + Reflexion	71.8 ±2.2	978

Table 3: Accuracy and efficiency on `gpt-3.5-turbo`.

4.2 Comparison Results

Data-Copilot Significantly Reduces Deployment Failure Risk. As shown in Table 2, compared to direct code generation, Data-Copilot achieves significant improvements: +41.7 (`GPT-3.5-turbo`),

Methods	Single	Multiple	Complex Rel.	Overall
Direct-Code	42.4	29.0	1.6	28.5
ReAct	56.3	48.6	14.6	44.1
Reflexion	67.8	55.1	48.1	59.0
Multi-agent	63.2	64.7	35.5	57.4
Data-Copilot	**71.8** ↑4	**70.0** ↑5.3	**67.1** ↑19	**70.2** ↑11.2

Table 4: `Accuracy` for three complexity levels samples.

		GPT-3.5	GPT-4
Direct Code Generation		28.3	70
		Workflow Deployment LLM	
		GPT-3.5	GPT-4
Interface Design LLM	GPT-3.5	31.9	49.6
	GPT-4	70.2	74.6

Table 5: We explore different LLM combinations for interface design and workflow deployment.

Methods	`Accuracy`(%)
Direct Code Generation	28.5
Data-Copilot	70.2
w/o Self-Exploration Request	35.9
w/o Interface Optimization	42.1
w/o Interface-Code Hybrid	63.8

Table 6: We ablate three modules from Data-Copilot.

+18.7 (`Llama3-70B`), and +29.2 (`Codellama-13B`). By decoupling into interface design and workflow deployment stages, smaller LLMs (`Llama3-70B`, `GPT-3.5`) perform at `GPT-4` level during deployment. Even `GPT-4` improves by +4%, confirming generalizability. Pre-designed interfaces reduce errors through optimization and validation during design, whereas direct code generation often overlooks details and introduces mistakes.

Data-Copilot Outperforms Advanced Agent Strategies in Both Accuracy and Efficiency. As shown in Table 3, Data-Copilot surpasses all baseline strategies in success rate (`Accuracy`), and efficiency (`#token`). Compared to the best baseline (Reflexion), Data-Copilot achieved an +11.2% improvement in accuracy and a -75% reduction in token consumption. Besides, as shown in the last two rows of Table 3, Data-Copilot can seamlessly integrate with agent strategies such as ReAct and Reflexion into the workflow deployment. For example, when combined with ReAct, Data-Copilot invokes an interface, obtains an intermediate result, and then reasons to invoke the next interface, improving its performance by +1.3.

Data-Copilot Reduces Repetitive Generation in the Real World. In the real world, most requests are similar or even repetitive. As shown in Table 3, Data-Copilot can save 70% of token consumption since its output only contains interface names and arguments. In contrast, baseline strategies have to repetitively generate complete code for each request, consuming many more tokens.

4.3 Why Data-Copilot Brings Improvements?

Data-Copilot exhibits superior performance in complex scenarios. We categorize test requests by entity count: `single entity`, `multiple entities`, and `multiple entities with complex relations` (statistics in Appendix B.2). As Table 4 shows, with single entities, improvement is minimal (4%). However, with multiple entities and complex relations, improvements reach 5.3% and 19%, respectively. Baselines often generate **logically incorrect code** and **omit critical steps**, while Data-Copilot simply invokes versatile interfaces, reducing real-time response complexity through pre-designed interfaces.

The Quality of Interface is the Key Factor. We analyze the effects of using different LLM combinations for two stages: interface design and workflow deployment. As shown in Table 5, we observe when using a weaker LLM for interface design, the effectiveness is significantly diminished. For example, using `GPT-3.5-turbo` (1st stage) and `GPT-4-turbo` (2nd stage) resulted in a score of only 49.6, which is even lower than directly using GPT-4-turbo for direct code generation (70.2). This phenomenon can also be seen in other combinations. After manually checking, we find interfaces designed by GPT-3.5-turbo are prone to failure and exhibit poor generalizability. Therefore, we chose to use `GPT-4-turbo` for interface design and optimization, while employing `GPT-3.5-turbo` for real-time deployment, achieving a balance between accuracy and efficiency.

5 Conclusion

We propose Data-Copilot, a code-centric data analysis agent. It generates code for large-scale data processing and creates interface modules through data exploration, improving real-time request success. It autonomously designs universal interfaces for various data types and invokes them for reliable problem-solving. Experiments show higher success rates with lower token consumption.

Limitations

Data-Copilot proposes a new paradigm for addressing the data-related task, through LLM. But we want to highlight that it still remains some limitations or improvement spaces:

1) **Online Design Interface.** The essence of Data-Copilot lies in effective interface design, a process that directly affects the effectiveness of subsequent interface deployments. Currently, this interface design process is conducted offline. Therefore, it is crucial to explore how to design the interface online and deploy it simultaneously. It greatly broadens the application scenarios of Data-Copilot.

2) **System stability** The interface deployment process can occasionally be unstable. The main source of this instability is because LLM is not fully controllable. Despite their proficiency in generating the text, LLMs occasionally fail to follow the instructions or provide incorrect answers, thus causing anomalies in the interface workflow. Consequently, finding methods to minimize these uncertainties during the interface dispatch process should be a key consideration in the future.

3) **Data-Copilot possesses the potential to handle data from other domains effectively.** Currently, our focus is on developing an LLM-based agent for the data domain. Due to the limited access to data, we chose the China Financial Data, which includes stocks, futures, finance, macroeconomics, and financial news. Although these data all belong to the financial domain, the data volume is extremely large, and the data schemas are highly different. The corresponding user's requests are also diverse, which poses a great challenge to the current LLM. However, Data-Copilot has adeptly accomplished this task. Therefore, we believe Data-Copilot also possesses the potential to effectively handle data from other domains.

References

Sangzin Ahn. 2024. Data science through natural language with chatgpt's code interpreter. *Translational and Clinical Pharmacology*, 32(2):73.

Shamma Mubarak Aylan Abdulla Almheiri, Mohammad AlAnsari, Jaber AlHashmi, Noha Abdalmajeed, Muhammed Jalil, and Gurdal Ertek. 2024. Data analytics with large language models (llm): A novel prompting framework. In *International Conference on Business Analytics in Practice*, pages 243–255. Springer.

Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2sql is not enough: Unifying ai and databases with tag. *arXiv preprint arXiv:2408.14717*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.

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

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, and 1 others. 2024. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? *arXiv preprint arXiv:2407.10956*.

Shuaichen Chang and Eric Fosler-Lussier. 2023. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. *arXiv preprint arXiv:2305.11853*.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. 2023a. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*.

Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2024a. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–11.

Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2024b. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, and 1 others. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.

Liying Cheng, Xingxuan Li, and Lidong Bing. 2023. Is gpt-4 a good data analyst? *arXiv preprint arXiv:2305.15038*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and others. 2022. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311.

Victor Dibia. 2023. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. *arXiv preprint arXiv:2303.02927*.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

James Ford, Xingmeng Zhao, Dan Schumacher, and Anthony Rios. 2024. Charting the future: Using chart question-answering for scalable evaluation of llm-driven data visualizations. *arXiv preprint arXiv:2409.18764*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *ArXiv*, abs/2211.10435.

Ken Gu, Madeleine Grunde-McLaughlin, Andrew McNutt, Jeffrey Heer, and Tim Althoff. 2024a. How do data analysts respond to ai assistance? a wizard-of-oz study. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–22.

Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, and 1 others. 2024b. Blade: Benchmarking language model agents for data-driven science. *arXiv preprint arXiv:2408.09667*.

Jiajing Guo, Vikram Mohanty, Jorge H Piazentin Ono, Hongtao Hao, Liang Gou, and Liu Ren. 2024a. Investigating interaction modes and user agency in human-llm collaboration for domain-specific data analysis. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024b. Ds-agent: Automated data science by empowering large language models with case-based reasoning. *arXiv preprint arXiv:2402.17453*.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *ArXiv*, abs/2305.11554.

Sirui Hong, Yizhang Lin, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, and 1 others. 2024. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2023. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*.

Chenxu Hu, Jie Fu, Chenzhuang Du, Simian Luo, Junbo Zhao, and Hang Zhao. 2023. Chatdb: Augmenting llms with databases as their symbolic memory. *arXiv preprint arXiv:2306.03901*.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, and 1 others. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*.

Rongjie Huang, Mingze Li, Dongchao Yang, Jiatong Shi, Xuankai Chang, Zhenhui Ye, Yuning Wu, Zhiqing Hong, Jiawei Huang, Jinglin Liu, and 1 others. 2023. Audiogpt: Understanding and generating speech, music, sound, and talking head. *arXiv preprint arXiv:2304.12995*.

Jeevana Priya Inala, Chenglong Wang, Steven Drucker, Gonzalo Ramos, Victor Dibia, Nathalie Riche, Dave Brown, Dan Marshall, and Jianfeng Gao. 2024. Data analysis in the era of generative ai. *arXiv preprint arXiv:2409.18475*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. Dsbench: How far are data science agents to becoming data science experts? *arXiv preprint arXiv:2409.07703*.

Harshit Joshi, Abishai Ebenezer, José Cambronero, Sumit Gulwani, Aditya Kanade, Vu Le, Ivan Radiček, and Gust Verbruggen. 2023. Flame: A small language model for spreadsheet formulas. *arXiv preprint arXiv:2301.13779*.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards building open-source language models for text-to-sql. *arXiv preprint arXiv:2402.16347*.

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

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, and 1 others. 2023c. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023d. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and 1 others. 2024c. Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency. *arXiv preprint arXiv:2403.09732*.

Jinqing Lian, Xinyi Liu, Yingxia Shao, Yang Dong, Ming Wang, Zhang Wei, Tianqi Wan, Ming Dong, and Hailin Yan. 2024. Chatbi: Towards natural language to complex business intelligence sql. *arXiv preprint arXiv:2405.00527*.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. 2023a. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, Yun Wang, Linjun Shou, Ming Gong, and Nan Duan. 2023b. Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis. *Preprint*, arXiv:2303.16434.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Li Xian Liu, Zhiyue Sun, Kunpeng Xu, and Chao Chen. 2024a. Ai-driven financial analysis: Exploring chatgpt's capabilities and challenges. *International Journal of Financial Studies*, 12(3):60.

Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. 2024b. Are llms capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. *arXiv preprint arXiv:2402.17644*.

Weizheng Lu, Jiaming Zhang, Jing Zhang, and Yueguo Chen. 2024. Large language model for table processing: A survey. *arXiv preprint arXiv:2402.05121*.

Pingchuan Ma, Rui Ding, Shuai Wang, Shi Han, and Dongmei Zhang. 2023. Demonstration of insightpilot: An llm-empowered automated data exploration system. *arXiv preprint arXiv:2304.00477*.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *arXiv preprint arXiv:2406.14991*.

Paula Maddigan and Teo Susnjak. 2023. Chat2vis: generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *Ieee Access*, 11:45181–45193.

Mohamed Nejjar, Luca Zacharias, Fabian Stiehle, and Ingo Weber. 2023. Llms for science: Usage for code generation and data analysis. *Journal of Software: Evolution and Process*, page e2723.

OpenAI. 2022. Chatgpt.

OpenAI. 2023. Gpt-4 technical report.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *CoRR*, abs/2203.02155.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

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

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, and 1 others. 2023a. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023b. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Gaurav Sahu, Abhay Puri, Juan Rodriguez, Alexandre Drouin, Perouz Taslakian, Valentina Zantedeschi, Alexandre Lacoste, David Vazquez, Nicolas Chapados, Christopher Pal, and 1 others. 2024. Insightbench: Evaluating business analytics agents through multi-step insight generation. *arXiv preprint arXiv:2407.06423*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, M. Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *ArXiv*, abs/2302.04761.

Shuyu Shen, Sirong Lu, Leixian Shen, Zhonghua Sheng, Nan Tang, and Yuyu Luo. 2024. Ask humans or ai? exploring their roles in visualization troubleshooting. *arXiv preprint arXiv:2412.07673*.

Yongliang Shen, Kaitao Song, Xu Tan, Dong Sheng Li, Weiming Lu, and Yue Ting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *ArXiv*, abs/2303.17580.

Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce Ho, Carl Yang, and May D Wang. 2024. Ehragent: Code empowers large language models for complex tabular reasoning on electronic health records. *arXiv preprint arXiv:2401.07128*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Ruoxi Sun, Sercan Arik, Rajarishi Sinha, Hootan Nakhost, Hanjun Dai, Pengcheng Yin, and Tomas Pfister. 2023a. SQLPrompt: In-context text-to-SQL with minimal labeled data. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 542–550, Singapore. Association for Computational Linguistics.

Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023b. Sql-palm: Improved large language modeladaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and Efficient Foundation Language Models. *ArXiv*, abs/2302.13971.

Jorge Valverde-Rebaza, Aram González, Octavio Navarro-Hinojosa, and Julieta Noguez. 2024. Advanced large language models and visualization tools for data analytics learning. In *Frontiers in Education*, volume 9, page 1418006. Frontiers Media SA.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023a. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Tianshu Wang, Hongyu Lin, Xianpei Han, Le Sun, Xiaoyang Chen, Hao Wang, and Zhenyu Zeng. 2023b. Dbcopilot: Scaling natural language querying to massive databases. *arXiv preprint arXiv:2312.03463*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022a. Self-instruct: Aligning language model with self generated instructions. *Preprint*, arXiv:2212.10560.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Virendrabhai Purohit, Ishani Mondal, Jacob William Anderson, Kirby C. Kuznia, Krima Doshi, Kuntal Kumar Pal, and 21 others. 2022b. Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and 1 others. 2024. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy

Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. *CoRR*, abs/2206.07682.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022b. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Luoxuan Weng, Yinghao Tang, Yingchaojie Feng, Zhuo Chang, Peng Chen, Ruiqin Chen, Haozhe Feng, Chen Hou, Danqing Huang, Yang Li, and 1 others. 2024. Datalab: A unifed platform for llm-powered business intelligence. *arXiv preprint arXiv:2412.02205*.

Chenfei Wu, Sheng-Kai Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023a. Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models. *arXiv*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023b. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023c. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Xueqing Wu, Rui Zheng, Jingzhen Sha, Te-Lin Wu, Hanyu Zhou, Mohan Tang, Kai-Wei Chang, Nanyun Peng, and Haoran Huang. 2024. Daco: Towards application-driven and comprehensive data analysis via code generation. *arXiv preprint arXiv:2403.02528*.

Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. Waitgpt: Monitoring and steering conversational llm agent in data analysis with on-the-fly code visualization. *arXiv preprint arXiv:2408.01703*.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, and 1 others. 2023. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634*.

Siqiao Xue, Caigao Jiang, Wenhui Shi, Fangyin Cheng, Keting Chen, Hongjun Yang, Zhiping Zhang, Jianshan He, Hongyang Zhang, Ganglin Wei, Wang Zhao, Fan Zhou, Danrui Qi, Hong Yi, Shaodong Liu, and Faqiang Chen. 2023. Db-gpt: Empowering database interactions with private large language models. *arXiv preprint arXiv:2312.17449*.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, and 1 others. 2024. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453*.

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

Junyi Ye, Mengnan Du, and Guiling Wang. 2024. Dataframe qa: A universal llm framework on dataframe question answering without data exposure. *arXiv preprint arXiv:2401.15463*.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. Glm-130b: An Open Bilingual Pre-trained Model. *ICLR 2023 poster*.

Liangyu Zha, Junlin Zhou, Liyao Li, Rui Wang, Qingyi Huang, Saisai Yang, Jing Yuan, Changbao Su, Xiang Li, Aofeng Su, and 1 others. 2023. Tablegpt: Towards unifying tables, nature language and commands into one gpt. *arXiv preprint arXiv:2307.08674*.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023a. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. *arXiv preprint arXiv:2310.17342*.

Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023b. Large language models as data preprocessors. *arXiv preprint arXiv:2308.16361*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open Pre-trained Transformer Language Models. *ArXiv*, abs/2205.01068.

Xiaokang Zhang, Jing Zhang, Zeyao Ma, Yang Li, Bohan Zhang, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, and 1 others. 2024a. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*.

Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. 2024b. Benchmarking data science agents. *arXiv preprint arXiv:2402.17168*.

Appendix

A Preprocessing of Workflow Deployment

Intent Analysis To accurately comprehend user requests, Data-Copilot first parses the time, location, data object, and output format of user requests, which are critical to data-related tasks. For example, the request is: `"I want to compare the GDP and CPI trend in our area over the past five years"`, Data-Copilot parses it as: `"Draw a line chart of China's national GDP and CPI per quarter from May 2019 to May 2024 for comparison"`. To achieve this, we first invoke an external API to obtain the local time and network IP address, then feed this supportive information into LLMs along with the original request to generate the parsed result.

Multi-form Output Upon execution of the workflow, Data-Copilot yields the desired results in the form of graphics, tables, and descriptive text. Additionally, it also provides a comprehensive summary of the entire workflow. It greatly enhances the interpretability of the whole process, as the interface workflow is easy for humans to read and inspect. As the example shown in Figure F2, the request is `"Forecasting China's GDP growth rate..."`. Data-Copilot first interprets the user's intent. Then it deploys a three-step workflow: 1) Invoking `get-GDP-data()` interface to acquire historical GDP data. 2) Invoking `predict-next-value()` interface for forecasting. 3) Visualizing the output.

B Experiments Details

B.1 Baselines and Experiments Details

We compare Data-Copilot with ② direct code generation (Direct-Code) and various agent strategies: ③ ReAct (Yao et al., 2022): LLMs iteratively combine reasoning and code execution. ④ Reflexion (Shinn et al., 2023): LLMs refine responses based on compiler feedback, limited to two iterations. ⑤ Multi-agent collaboration (Wu et al., 2023b; Hong et al., 2023; Liang et al., 2023a): Three LLM agents (two coders, one manager) collaborate to generate solutions. Data-Copilot uses the same LLM as all baselines when invoking predesigned interfaces. Detailed prompts are in Appendix E.5.

B.2 The Definitions of Three Complexity Levels for Testset

We categorize our test set into three types based on the number of entities involved: single entity, multiple entities, and multiple entities with complex relations (e.g., loop calculations). The ststics of three subset are shown in Table B1.

- Single entity: Requests involve a single entity and can be resolved step-by-step. E.g., "query based on a specific condition".
- Multiple entities: Requests require processing multiple entities simultaneously. E.g., "compare a certain metric across multiple entities".
- Multiple entities with complex relations: Involving multiple entities and containing loops, nesting, and other intricate logic. E.g., "List the top 10 stocks by yesterday's price increase that also in the internet industry."

B.3 The Quality of Self-exploration Requests.

To assess the quality of self-exploration requests, we invited four additional graduate students to manually evaluate our test set (546 requests) and human-proposed requests (173 seed set) on four criteria: `task difficulty`, `request rationality`, `expression ambiguity`, and `answer accuracy`. We provide detailed guidance for each criterion in Appendix B.5. The results of two sets are shown in Table B2. We observed that the synthesized requests exhibit a comparable quality to human-proposed requests, with slightly higher difficulty, ambiguity, and similar rationality. It ensures that our test set can reflect most of the real-world demands. Besides, the evaluators gave a high score of 4.8 on the label (answer table) of our test set, which also ensures the accuracy of our dataset.

B.4 The detail of GPT-4o Evaluation

As described in Section 4.1, the evaluation contains three aspects: *Data-table*, *Image*, and *Efficiency*.

Data-table Evaluation: For each request, we use GPT-4o to compare predicted data table with human-annotated table. If GPT-4o identifies any inconsistencies, the judgment is False. For example, a request: *"Please show me the China's GDP ...."* with its labeled data-table: *[2023: 17.8 , 2022: 17.9 , 2021: 17.8 , 2020: 14.6]*. Predicted data:

Table B1: Statistics of four task types and three complexity levels on our test set.

Task Types		Request Complexity			
		Single Entity	Multiple Entities	Multi-entities with Complex Relation	Overall
	Stock	79	106	87	272
	Fund	55	36	22	113
	Corporation	97	30	4	131
	Other	6	12	13	31
	Total	237	184	126	547

Table B2: Human evaluation on the human-proposed requests and synthesized requests across four dimensions.

	Task Difficulty	Request Rationality	Expression Ambiguity	Answer Accuracy
Seed Set	3.5	4.3	4.4	-
Test Set	3.9	4.2	4.0	4.8

Table B3: Comparison of Human and GPT-4o Evaluations

	Human Evaluation	GPT-4o Evaluation
Average Score	65.8	67.2
Correlation Coefficient	0.894	0.894
N(score $\geq$ 60)	35 cases	38 cases

[2023: 17.8 , 2022: 17.9 , 2021: 17.8]. Based on them, judgment of GPT-4o is ***False***.

Image Evaluation: We design a comprehensive evaluation checklist for GPT-4o-based image scoring, comprising 5 main categories with 10 sub-dimensions. It includes (1) numerical points, (2) lines, (3) axes, (4) aesthetics of image layout, and (5) chart design, e.g., *Chart type, Color usage, Proportion and scale, Labels and legends, Readability, Completeness, Relevance, Distinctiveness, Data accuracy, ...* If the total score exceeds 60 points, the image is considered to meet expectations (True); otherwise, it is judged as false.

Manual Evaluation Vs. GPT-4o Evaluation. For data-table evaluation, GPT-4o's assessment results are highly accurate as it only needs to compare differences in text modality. For image evaluation, we randomly sampled 50 examples for both human and GPT-4o-based evaluation. As shown in Table B3, we calculate the average score, correlation coefficient, and the #samples >= 60 of the two methods. The results show that the average scores of the two evaluation methods are close (GPT-based: 67.2, Human-based: 65.8), and the two score sequences also have a strong correlation (0.894).

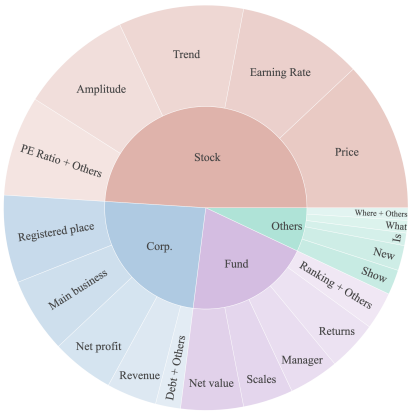


Figure B1: We count the keywords for each request type (Stock, Corp., Fund, Others) in the test set.

B.5 Human Evaluation on Test Benchmark

We invite four more graduate students to manually evaluate our benchmark according to four criteria: task difficulty, request rationality, expression ambiguity, and answer accuracy. The belief guidance for human evaluation is as follows:

- **Task Difficulty**: The difficulty of the task, whether it requires multiple steps... Scoring Criteria: 5: very difficult,..., 1: easy.
- **Request Rationality**: Whether the request is reasonable, or if it is strange and does not align with human habits... Scoring Criteria: 5: Reasonable, aligns with humans..
- **Expression Ambiguity**: Whether the phrasing of the request is ambiguous, or if the entities involved are unclear... Scoring Criteria: 5: Clear without any ambiguity...
- **Answer Accuracy**: Whether the answer table is correct, detailed, and comprehensive, totally meeting user expectations... Scoring Criteria: 5: Data is totally correct ... 3: Partial...

B.6 Ablation Study

As shown in Section 4.3, we ablate Data-Copilot from three aspects: ① **We ablate the self-**

exploration request process. Instead, we directly used seed requests for interface design and optimization. It leads to a 34.3 performance drop. We observe that too few seed requests are insufficient to design universe interfaces. ② **We ablate the interface optimization**. It means every successfully designed interface was retained in the interface library. We observe that performance is also significantly affected (-28 points). Without interface optimization, there are many similar interfaces, which hurt the effect of workflow invocation. ③ **Interface-Code Hybrid Generation**: Data-Copilot can only invoke interfaces during workflow deployment. Without a hybrid generation manner, it shows a 6.4-point performance drop, which is caused by "uncovered" requests. It highlights our flexibility in addressing different types of requests.

B.7 Expanding to Other Programming Languages

In addition to the Python language, Data-Copilot exhibits excellent scalability, allowing for easy switching to other programming languages by simply regenerating the corresponding interfaces. We tested three programming languages: Python, C++, and Matlab. The results indicate that Python performed the best (`Accuracy`: 70.2), followed by C++ (54.2), with Matlab (36.5) yielding the poorest results. Upon examination, we found that Matlab code often suffers from formatting errors or dimensional discrepancies in the data, rendering the program non-executable. We speculate this may be related to a lack of sufficient Matlab code in the pre-training corpus. So ultimately, we opted for Python for Data-Copilot.

C Related Works

In the recent past, breakthroughs in large language models (LLMs) such as GPT-3, GPT-4, PaLM, and LLaMa (Brown et al., 2020; Chowdhery et al., 2022; Zhang et al., 2022; Zeng et al., 2023; Touvron et al., 2023; Ouyang et al., 2022; OpenAI, 2023; Wei et al., 2022a; Zhang et al., 2024a) have revolutionized the field of natural language processing (NLP). These models have showcased remarkable competencies in handling zero-shot and few-shot tasks along with complex tasks like mathematical and commonsense reasoning. The impressive capabilities of these LLMs can be attributed to their extensive training corpus, intensive computation, and alignment mechanism (Ouyang et al., 2022; Wang et al., 2022b,a).

LLM-based Agent Recent studies have begun to explore the synergy between external tools and large language models (LLMs). Tool-enhanced studies (Schick et al., 2023; Gao et al., 2022; Qin et al., 2023a; Hao et al., 2023; Qin et al., 2023b; Hou et al., 2023) integrate external tools into LLM, thus augmenting the capability of LLMs to employ external tools. Several researchers have extended the scope of LLMs to include the other modality (Wu et al., 2023a; Surís et al., 2023; Shen et al., 2023; Liang et al., 2023b; Huang et al., 2023). In addition, there are many LLM-based agent applications (Xie et al., 2023), such as CAMEL (Li et al., 2023a), AutoGPT[2], AgentGPT[3], BabyAGI[4], BMTools[5], LangChain[6], Agentverse (Chen et al., 2023b), Autoagent (Chen et al., 2023a), MetaGPT (Hong et al., 2023), AutoGEN (Wu et al., 2023b), etc. Most of them are focused on daily tools or code generation and do not consider the specificity of data-related tasks. Except for learning to operate the tools, several contemporaneous studies (Cai et al., 2023; Qian et al., 2023) have proposed to empower LLMs to create new tools for specific scenarios like mathematical solving and reasoning. These impressive studies have revealed the great potential of LLM to handle specialized domain tasks.

Applying LLM To Data Science Apart from these studies, the application of large models in the field of data science has garnered significant interest among researchers (Maddigan and Susnjak, 2023; Valverde-Rebaza et al., 2024; Gu et al., 2024a; Liu et al., 2024b; Chen et al., 2024a; Zhang et al., 2023b; Ahn, 2024; Inala et al., 2024; Liu et al., 2024a; Xie et al., 2024; Wu et al., 2024; Guo et al., 2024a; Cao et al., 2024; Lu et al., 2024; Ye et al., 2024; Sui et al., 2024; Ford et al., 2024; Chen et al., 2024b; Weng et al., 2024; Shen et al., 2024). FLAME (Joshi et al., 2023) investigates the feasibility of using NLP methods to manipulate Excel sheets. StructGPT (Jiang et al., 2023) explore reasoning abilities of LLM over structured data. LiDA (Dibia, 2023) and GPT4-Analyst (Cheng et al., 2023; Ma et al., 2023) focus on automated

[2] https://github.com/Significant-Gravitas/Auto-GPT
[3] https://github.com/reworkd/AgentGPT
[4] https://github.com/yoheinakajima/babyagi
[5] https://github.com/OpenBMB/BMTools
[6] https://github.com/hwchase17/langchain

data exploration. Besides, many reseraches (Liu et al., 2023; Chang and Fosler-Lussier, 2023; Dong et al., 2023; Almheiri et al., 2024), like Sheet-Copilot (Li et al., 2023b), BIRD (Li et al., 2024b), DAIL-SQL (Gao et al., 2023), DIN-SQL (Pourreza and Rafiei, 2023), PET-SQL (Li et al., 2024c), DB-Copilot (Wang et al., 2023b), MAC-SQL (Wang et al., 2023a), ACT-SQL (Zhang et al., 2023a), ChatBI (Lian et al., 2024), CodeS (Li et al., 2024a), SQLPrompt (Sun et al., 2023a), ChatDB (Hu et al., 2023), SQL-PaLM (Sun et al., 2023b), and DB-GPT (Xue et al., 2023), EHRAgent (Shi et al., 2024) apply LLMs to Text2SQL and table rasoning. Chain-of-Table (Wang et al., 2024) proposes a step-by-step reasoning strategy based on the table. Some researchers also focus on designing various benchmarks and evaluation methods (Lai et al., 2023; Zhang et al., 2024b; Sahu et al., 2024; Yang et al., 2024; Ma et al., 2024; Gu et al., 2024b; Jing et al., 2024; Hu et al., 2024) for LLMs in data science.

D Visualization

We provide several cases in this section to visualize workflow deployed by Data-Copilot, which includes queries about diverse sources (stocks, company finance, funds, etc.) using different structures (parallel, serial, and loop Structure).

Different Structures As shown in Figure F4,F5, Data-Copilot deploys different structural workflows based on user requirements. In Figure F4, the user proposes a complex request, and Data-Copilot deploys a loop structure to implement the financial data query of each stock, and finally outputs a graph and table in parallel. In Figure F5, Data-Copilot proposes a parallel structure workflow to meet user request (the demand for comparison in user request) and finally draws two stock indicators on the same canvas. These concise workflows can cope with such complex requests well, which suggests that the data exploration and workflow deployment process of Data-Copilot are rational and effective.

Diverse Sources Figure F6, F7, F8 demonstrate that Data-Copilot is capable of handling a large number of data sources, including stocks, funds, news, financial data, etc. Although the formats and access methods of these data types are quite different, our system efficiently manages and displays the data through its self-designed versatile interface, requiring minimal human intervention.

E Detailed Prompts

We provide detailed prompts for our Data-Copilot and baselines. Specifically, for the data exploration, we provide detailed prompts in Appendix E.3 for four phases: self-exploration, interface design, and interface optimization. For the workflow deployment phase, we also provide prompts for three key procedures in Appendix E.4: Intent Analysis, Task Selection, and Planning Workflow. Additionally, we outline prompts for all baselines in Appendix E.5: Direct-Code, ReAct, Reflection, and Multi-Agent Collaboration strategies.

E.2 Example for Data Exploration

```
###Instruction: Given some data and its
description, please mimic these seed
requests and generate more requests. The
 requests you generate should be as
diverse as possible, covering more data
types and common needs.
###Seed Request: {request1, request2,...}
###Parsing file for GDP_Data:{
    Description: This data records China's
    annual and quarterly GDP...,
    Access Method: pro.cn-gdp(start-time,
    end-time, frequency,...),
    Output Schema: Return 9 columns,
    including quarter, gdp, gdp-yoy...,
    Usage:{
        Example: pro.cn-gdp(start-q='2018Q1',
        end-q='2019Q3',..),
        First Row: {2019Q4, 990,
        ..}, Last Rows: {2018Q4, 900, ..}}}
###Parsing file for Stock_data:{...},
....
```

F Case Study

Detailed cases of data exploration and workflow deployment are presented in Appendices F.1 and F.2.

E.1 Algorithm Flow For Interface Design and Optimization

Step 1: Interface Design and Testing

1. **One-to-One Interface Design**
   * Design interfaces one-by-one using synthesized requests (request 1----> interface 1)
   * Generate a series of initial interfaces with single functionalities
2. **Test Case Generation**
   * For each designed interface:
      > Sample a sub data-table from relevant data sources
      > Generate test requests based on data schemas and sampled sub-tables
      > For example: For an interface: "GDP_retrieves(Year, Country)"
         - Sample a data point from the GDP table: China, 2023, $13 trillion
         - Generate test request: "What is China's GDP in 2023?"
         - Use the sampled sub-table as the expected answer
      > Repeat this process K times for each interface, generating K test cases
3. **Interface Evaluation**
   * Use the generated test cases to evaluate the usability of each interface
   * Record interfaces that pass the tests and their testing results

Step 2: Interface Optimization and Compiler Feedback Evaluation

1. **Similar Interface Retrieval**
   * Analyze the functionality and parameters of all interfaces
   * Identify pairs of interfaces with similar or overlapping functionalities
2. **Interface Merging Decision**
   * For each pair of similar interfaces, assess the necessity and feasibility of merging
   * Consider functional coverage, usage scenarios, and complexity of the interfaces
3. **Interface Merging Execution**
   * Design a new merged interface ensuring:
      > The new interface covers all functionalities of the original two interfaces
4. **Merged Interface Evaluation**
   * Evaluate the merged interface using test cases generated in the previous steps
   * Collect compiler feedback on the merged interface
   * Verify:
      > Whether the new interface can correctly handle all test cases of the original interfaces
      > Whether the output is consistent with the original interfaces in both format and content
5. **Feedback-Based Self-Optimization**
   * If the merged interface fails evaluation:
      > Guide the LLMs to analyze compiler feedback
      > Perform self-reflection based on the feedback and two old interfaces
      > Optimize the interface design to address issues
   * Repeat the evaluation-reflection-optimization cycle multiple times
   * Abandon the merging attempt after multiple fails

E.3 Prompts For Data Exploration

```
Explore Data by self-exploration Phase
###Instruction: Given some data and its description, please mimic these seed
requests and generate more requests. The requests you generate should be as diverse
as possible, covering more data types and common needs.

###Seed Request: {request1, request2,...}
###parsing file for GDP-Data:{
    Description: This data records China's annual and quarterly GDP...,
    Access Method: You can access the data by pro.cn-gdp(start-time, end-time, frequency,...),
    start-time means....,
    Output Schema: The data return 9 columns, including quarter: quarter, gdp: cumulative GDP,
    gdp-yoy: quarterly Year-on-Year growth rate, pi,...,
    Usage:{
            Example: pro.cn-gdp(start-q='2018Q1', end-q='2019Q3', frequency='quarter'),
            First Row: {2019Q4 990865.1 6.10, ...},
            Last Rows: {2018Q4 900309.5 6.60, ...}}}
###parsing file for stock_data: {...}


Interface Design:
###Instruction: You are an experienced program coder. Given a request and some
existing interfaces, you should use these interfaces to solve the request or design
new interfaces to resolve my request.

(1) You should define the name, function, inputs, and outputs of the interface.
Please describe the functionality of the interface as accurately as possible and
write complete implementation code in the new interface.
(2) Finally please explain how to resolve my request using your newly designed
interfaces or existing interfaces in the neural language.

###Output Format:
Your newly designed interfaces:
Interface1={Interface Name: {name},
            Function description: {This interface is to ...},
            Input: {argument1: type, argument2: type, ...},
            Output: {pd.DataFrame} }
Interface2=....

The solving process for request: {To fullfil this request, I design a interface ...}
###The user request: {input request}
###Data Files: {All data parsing files}
###The existing interfaces: {all interfaces in library}

Interface Optimization
###Instruction: Please check that the interface you have designed can be merged with
 any existing interfaces in the library.
(1) You should merge interfaces with similar functionality and similar input and
output formats into a new interface.
(2) You can use parameters to control the different inputs, if you want to merge two
 interfaces.
(3) Please explain your reason for merging and output all interfaces in the library
after merging.
(4) If you don't think a merge is necessary, then just add new interfaces into the
existing interface library and output them all.

###New interfaces: {interfaces}
###Existing interfaces: {interface1, interface2, interface3, ..}
###Output Format:
The reasons for merging: {reason}
Interfaces after merging: {interface1, interface2, optimized interface3, ..}
```

E.4 Prompts For Workflow Deployment

Intent Analysis Phase

```
### Analysis Prompt: Please parse the input request for time, place, object, and
output format. You should rewrite the instruction according to today's date. The
rewritten new instruction must be semantically consistent and contain a specific
time and specific indicators.

### Output Format: Rewritten Request. (Time:%s, Location:%s, Object:%s, Format:%s).

### User Request: Today is {Timestamp}. The user request is {Input Request}.
Please output a Rewritten Request.
```

Task Selection

```
###Select Prompt: Please select the most suitable task according to the given
Request and generate its task_instruction in the format of task={task_name:
task_instruction}. There are four types of optional tasks. [fund_task]: used to
extract and process tasks about all public funds. [stock_task]: for extracting and
processing tasks about all stock prices, index information, company financials, etc
., [economic_task]: for extracting and processing tasks about all Chinese
macroeconomic and monetary policies, as well as querying companies and northbound
funds, [visualization_task]: for drawing one or more K-line charts, trend charts, or
 outputting statistical results.

###Output Format: task1={%s: %s}, task2={%s: %s}

###User Request: {Rewritten Request}.
Please output a task plan for this request.
```

Planning Workflow

```
###Planning prompt: Please use the given interface (function) to complete the
Instruction step by step. At each step you can only choose one or more interfaces
from the following interface library without dependencies, and generate the
corresponding arguments for the interface, the arguments format should be strictly
in accordance with the interface description. The interface in the later steps can
use results generated by previous interfaces.

###Output Format:
Please generate as json format for each step:step1={"arg1": [arg1,arg2...], "
function1": "%s", "output1": "%s", "description1": "%s"}, step2={"arg1": [arg1,arg2
..], "function1": "%s", "output1": "%s", "description1": "%s"}, ending with ###.

###User Request: {Task Instruction}.
Please output an interface invocation for this instruction.
```

E.5 Prompts For Baselines

Direct-Code LLM

```
###Instruction: You are an artificial intelligence assistant. Given some data access
 methods and a user request, you should write a complete Python code to fulfill the
user's request. Your code must completely fulfill all the user's requirements
without syntax errors!

###User Request: {User request}
###Data files: {All data files}
Please solve the request by Python Code.
```

Step-by-Step ReAct

```
###Instruction: You are an artificial intelligence assistant. Given some data access
 methods and a user request, please think step by step and generate your thoughts
and actions for each step, and then finally realize the user's request.
###User Request: {User request}
###Data files: {All data files}

###Thought Prompt: Please think about the next action that should be taken to handle
 the user request.
### {Thought: I need to ...}
###Action Prompt: Based on your previous thoughts, please generate a complete Python
 code to accomplish what you just planned.
### {Action: def get-data()....}
###Observation Prompt: Please summarize the results of the code execution just now
and think about whether this result accomplishes what you planned for this step.
### {Action: Yes, I observed that this function successfully fetched the data...}
....
```

Step-by-Step Reflexion

```
###Instruction: You are an artificial intelligence assistant. Given some data access
 methods and a user request, please think step by step and then generate your
thoughts and actions for each step. After the execution of your current action, you
need to reflect on the results until your current plan has been successfully
completed. Then you think about the next step and then generate your next action,
and finally realize the user's request.
###User Request: {User request}
###Data files: {All data files}

###Thought Prompt: Please think about the next action that should be taken to handle
 the user request.
### {Thought: I need to ...}
###Action Prompt: Based on your previous thoughts, please generate a complete Python
 code to accomplish what you just planned.
### {Action: def get-data()....}
###Observation Prompt: Please record the compiler's return results just now and
think about whether this result accomplishes what you planned for this step.
### {Action: No, I observe that the compiler returns an error...}
### Reflection Prompt: Please reflect on the error returned by the compiler and
regenerate a new Python code to resolve the issue. If the compilation passes without
 any errors, reflect on whether the current result is what you planned to do.
### {Action: I revise my solution as follows: def get-data2()....}
....
```

Multi-agent collaboration

```
###Instruction For Manager: You are the manager of the project team and you need to
lead your team to fulfill user requests. You have two experienced programmers under
you (Programmer-A and -B) and you need to assign them the same or different tasks
according to the user request, then organize the discussion, and finally solve the
problem.
###Instruction For Agent1/Agent2: You are an experienced programmer. Your team has a
 colleague who is also a programmer and a manager. You need to write code according
to the manager's arrangement, discuss it with them, improve your program, and get a
consensus conclusion.
###User Request: {User request}
###Data files: {All data files}


--------  Phase1 Discussion for Task Assignment -------
###Task Assignment Prompt For 3 Agents: Now let's start discussing how to solve user
 problems. The speaking order is manager, A and B. After a round of discussion, the
manager decides on a preliminary solution and assigns specific tasks to the two
programmers
###Manager Response: {To fulfill the request... }
###Programmer A Response: {Yes, I agree with the manager. We should first,....}
###Programmer B Response: {That's a good solution. We also should...}
###Manager Response: {After discussion, Programmer A should..., B should ....}


--------  Phase2 Coding and Discussion-------
###Prompts For 3 Agents: Two programmers complete their respective tasks as assigned
 by their manager. Then the manager organizes a discussion on whether the two
options and their running results are reasonable. Lastly, two programmers need to
regenerate the code.
###Programmer A Response: {Solution1 Generated by A}
###Programmer B Response: {Solution1 Generated by B}
###Prompts For 3 Agents: Please evaluate two solutions. {Solution1, Result1, Solution2,
Result2}
###Programmer A Evaluation: {I think ...}
###Programmer B Evaluation: {I agree with ...}
###Programmer A Regeneration: {Revised Solution1}
###Programmer B Regeneration: {Revised Solution2}

--------  Phase3 Discussion-------
....

--------  Phase4 Final Decision-------
###Prompts For manager: Please output a complete and correct solving code for user
request based on the previous discussions of the group.
###Manager Decision: {Final Solution}
```

F.1 Cases for Data Exploration

```
---- The First Request: Interface Design -----
Input Request: Get the daily opening price data of Bank of Hangzhou from Jan 23,
2018 to Mar 13, 2019 and plot the trend.

Response: First, we need to design a function to get the opening price data from the
 bank's database. Then, we need another function to process this data and prepare
them for graphical representation. Finally, we need a function to plot the trend
chart.

New Design Interfaces:
{Interface1=Name:getBankData(), Function:.., Input:.., Output:..}
{Interface2=Name:prepareDataForPlotting(), Function:.., Input:.., Output:..}
{Interface3=Name:plotData(), Function:.., Input:.., Output:..}

The solving process for request: These three new interface functions will meet your
needs. In your program, you would first get the data using getBankData, then process
 the data using prepareDataForPlotting, and finally use plotData to plot the trend.
Complete code: {Complete code using three new interfaces...}

---- The Second Request: Interface Definition Phase-------
Input Request: Get the monthly closing price data of Ningde Times from January 23,
2018 to March 13, 2019, and plot the trend.

Response: To fulfill this requirement, we design a new interface function. In
addition, I need to use two existing interfaces to fulfill your requirement.

New Designed Interfaces:
{Interface4=Name: getStockData()..}

Existing Interfaces
{Interface1=Name:getBankData(), ...}
{Interface2=Name:prepareDataForPlotting(), ...}
{Interface3=Name:plotData(), ...}

The solving process for request: You need to call these functions to meet your needs
. First, use new interface {getStockData()} to get the data of Ningde Times, then use
existing interface {prepareDataForPlotting()} to process this data, and finally use {
plotData()} to plot the trend...

Complete code: {Complete code using three new interfaces...}

---- Interface Optimization Phase -------
The reason for merging: Yes, these interface functions can be merged and
reformulated to eliminate duplicate functions. We can merge getBankData() and
getStockData() into a more generic function get-financial-data(), which can fetch any
financial data, including bank and stock data. In addition, we can keep the
prepareDataForPlotting and plotData functions, as they can also be applied to
different types of financial data]

{Interface1=Interface Name:get-financial-data(), ...}
{Interface2=Interface Name:prepareDataForPlotting(), ...}
{Interface3=Interface Name:plotData(), ...}
```

F.2 Cases for Real-time Request

```
### Request: Today is 2019-03-13. The user request is to please help me plot the
stock price trend of Guizhou Maotai from January 23, 2018 to today.

### Rewritten Request: Please help me plot the closing price of Guizhou Maotai from
Jan 23, 2018 to Mar 13, 2019 (Time:20180123-20190313, Location: China, Object:
Guizhou Maotai stock price, Format: Line).

###Plan: task1={"stock_task": "Get the time series data of Guizhou Maotai's stock
price from 20180123 to 20190313"}, task2={"visualization_task": "Plot the stock
price of Guizhou Maotai from 20180123 to 20190313 on a line chart"}

###Function Call:
            step1={
                "arg1": ["Guizhou Maotai", "20180123", "20190313", "daily"],
                "function1": "get_stock_prices_data",
                "output1": "result1",
                "description1": "Guizhou Maotai daily stock price (open, close, high
                , low, etc.) time series data"},
            step2={
                "arg1": ["result1","close"],
                "function1": "calculate_stock_index",
                "output1": "result2v,
                "description1": "Guizhou Maotai's closing price time series data"},
            step3={
                "arg": ["result2", null, "line", "Guizhou Maotai closing price chart
                 from January 23, 2018 to March 13, 2019"], "function": "
                plot_stock_data",
                "output": "result1",
                "description": "Guizhou Maotai Stock Price Line Chart"
            } ###
```

Request: Forecasting China's GDP growth rate for the next 4 quarters

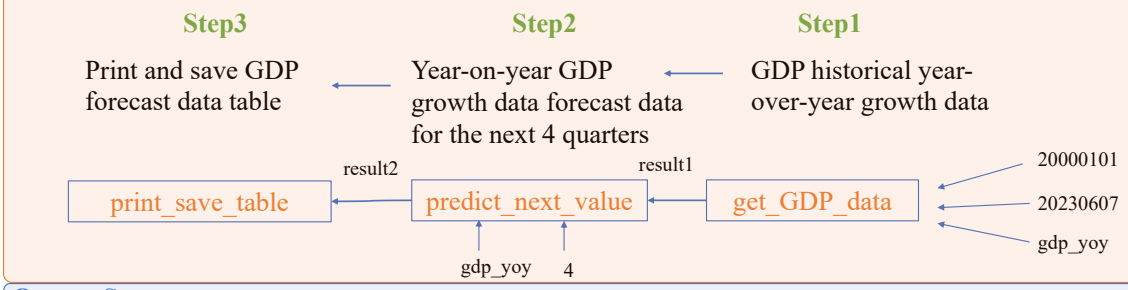
Intent Analysis: Show data and print tables for the next 4 quarters based on China's GDP growth rate for each quarter from 20000101 to June 07, 2023 (today)

WorkFlow Planning :

step1=: {'arg1': ['20000101', '20230607', 'gdp_yoy'], 'function1': 'get_GDP_data', 'output1': 'result1', 'description1': GDP historical year-over-year growth data'}

step2=: {'arg1': ['result1', 'gdp_yoy', 4], 'function1': 'predict_next_value', 'output1': 'result2', 'description1': ' Year-on-year GDP growth data forecast data for the next 4 quarters '}

step1=: {'arg': ['input1', ' Table of GDP forecast data for the next 4 quarters ', True], 'function': 'print_save_table', 'output': 'result1', 'description': ' Print and save GDP forecast data table '}



Output Summary

First, we use the two interface {get_GDP_data},{predict_next_value} to fetch the historical GDP data and predict the GDP for the next four quarters. Then we invoke the {print_save_table} interface to print and save the table.

Finally, we show you the historical yoy GDP and forecast data as following:

2000Q4	China	8.4
2000Q3	China	8.9
2000Q2	China	8.9
2000Q1	China	9
2021Q4	China	8.1
2022Q1	China	4.8
2022Q2	China	2.5
2022Q3	China	3
2022Q4	China	3
pred1	pred1	5.381894409937889
pred2	pred2	5.316734121418554
pred3	pred3	5.251573832899219
pred4	pred4	5.186413544379884

Figure F2: Data-Copilot deploys workflows to solve users' prediction request. It invokes three interfaces step by step and generates arguments for each interface.

Interface design in First Stage by LLM

	Data Acquisition	Index Calculation	Table Manipulation	Visualization	General Processing Logic
Stock	11	9	4	5	5
Fund	6	8	e.g., sort_table select_col	e.g., plot_line_trend generate_report	e.g., loop_rank handle_missing_data
Corp.	4	8			
Others	5	3			

Data Acquisition Interfaces	Name:	**get_stock_prices_data (…)**
	Function:	Retrieves the **daily/weekly/monthly** price data for a given stock name during a specific time period
	Input/output:	(stock_name: str='', start_date: str='', end_date: str='', **freq**:str='daily') -> pd.DataFrame
	Name:	**get_cpi_ppi_currency_supply_data(…)**
	Function:	Query three types of macro-economic data: **CPI, PPI and Money Supply**, each with several different indexes
	Input/output:	(start_month: str = '', end_month: str = '', **type**: str = 'cpi', index: str = '') -> pd.DataFrame
Data Processing Interfaces	Name:	**calculate_stock_index (…)**
	Function:	**Select or Calculate a index** for a stock from source dataframe
	Input/output:	stock_data: pd.DataFrame, index:str='close' -> pd.DataFrame
	Name:	**loop_rank (…)**
	Function:	It **iteratively applies the given function** to each row and get a result
	Input/output:	(df: pd.DataFrame, **func: callable**, *args, **kwargs) -> pd.DataFrame
	Name:	output_mean_median_col
	Function:	It calculates **the mean and median** value for the specified column
	Input/output:	(data: pd.DataFrame, col: str = 'new_feature') -> float:\n
DataFrame Manipulation Interfaces	Name:	**merge_indicator_for_same_stock (…)**
	Function:	Merges two DataFrames (two indicators of the same stock)
	Input/output:	(df1: pd.DataFrame, df2: pd.DataFrame) -> pd.DataFrame
		select_value_by_column (…)
	Name:	Selects a specific column or a specific value within a DataFrame
	Function:	(df1:pd.DataFrame, col_name: str = '', row_index: int = -1) ->
	Input/output:	Union[pd.DataFrame, Any]
Visualization Interfaces	Name:	**plot_stock_data (…)**
	Function:	This function plots stock data for **cross-sectional data or time-series** data using **Line graph or Bar graph**
	Input/output:	(stock_data: pd.DataFrame, ax: Optional[plt.Axes] = None, figure_**type**: str = 'line', title_name: str ='') -> plt.Axes
	Name:	**plot_k_line (…)**
	Function:	Plots a K-line chart of stock **price, volume, and technical index**: macd, kdj, etc.
	Input/output:	(stock_data: pd.DataFrame, title: str = '') -> None
	Name:	**print_save_table (…)**
	Function:	It prints the dataframe and saves it to a CSV file at the specified file path
	Input/output:	(df: pd.DataFrame, title_name: str, save:bool = False ,file_path: str = './output/') -> None
Data Prediction Interfaces	Name:	**predict_next_value (…)**
	Function:	Predict the next n values of a specific column in the DataFrame using linear regression
	Input/output:	(df: pd.DataFrame, pred_index: str = 'nt_yoy', pred_num:int = 1.) -> pd.DataFrame

Figure F3: We have listed some of Data-Copilot's self-design interfaces, including five categories. Most of the interfaces are a combination of several simple interfaces. For example, the text marked in blue indicates that the interface includes multiple functions.

Request: The year-on-year net profit growth rates of the SSE 50 index constituents in the first quarter of this year were

Intent Detecting Stage: Search the distribution of the quarterly financial indicators of the SSE 50 index constituents' net profit growth rate (netprofit_yoy) from 01/01/2023 to 03/31/2023.

WorkFlow Planning :

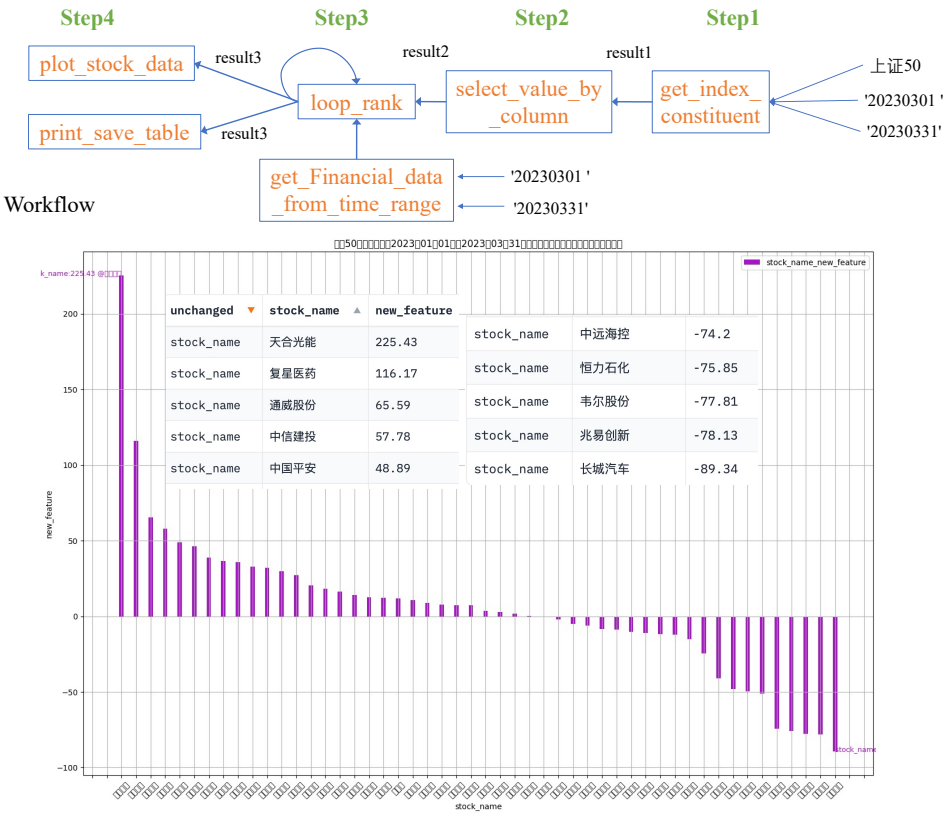
step1=: {'arg1': ['上证50', '20230301', '20230331'], 'function1': 'get_index_constituent', 'output1': 'result1', 'description1': ' Information on all SSE 50 constituents for the month '}

step2=: {'arg1': ['result1', 'stock_name', -1], 'function1': 'select_value_by_column', 'output1': 'result2', 'description1': ' Stock names of SSE 50 constituent stocks '}

step3=: {'arg1': ['result2', 'get_Financial_data_from_time_range', '20230331', '20230331', 'netprofit_yoy'], 'function1': 'loop_rank', 'output1': 'result3', 'description1': ' Cross-sectional data of the year-on-year growth rate of net profit of each constituent stock of the SSE 50 for the first quarter 20230331'}

step4=: {'arg': ['input1', None, ' bar ', 'SSE50 quarterly report 20230331 year-on-year distribution of net profit growth of each constituent stock '], 'function': 'plot_stock_data', 'output': 'result1', 'description': ' Plotting bar graphs '}

step5=: {'arg': ['input1', 'new_feature', 'SSE50 quarterly report 20230331 year-on-year distribution of net profit growth of each constituent stock '], 'function': 'print_save_table', 'output': 'result2', 'description': 'print and save table'}



unchanged	stock_name	new_feature
stock_name	天合光能	225.43
stock_name	复星医药	116.17
stock_name	通威股份	65.59
stock_name	中信建投	57.78
stock_name	中国平安	48.89

stock_name	中远海控	-74.2
stock_name	恒力石化	-75.85
stock_name	韦尔股份	-77.81
stock_name	兆易创新	-78.13
stock_name	长城汽车	-89.34

Figure F4: For complex requests about stock financial data, Data-Copilot deploys a loop workflow to solve user requests and finally outputs images and tables in parallel.

Request: Compare the change in the P/E ratio of Ningde Times and Guizhou Maotai in the last three years

Intent Detecting Stage:

Please show the technical indicator price-to-earnings valuation (pe-ttm) charts of Ningde Times(宁德时代) and Guizhou Maotai (贵州茅台) from June 6, 2020 to June 6, 2023 to compare the change in their PE.

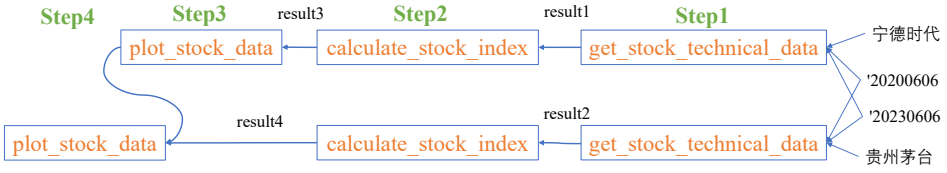
WorkFlow Planning :

step1=: {'arg1': ['宁德时代', '20200606', '20230606'], 'function1': 'get_stock_technical_data', 'output1': 'result1', 'description1': 'Time series data of Ningde times technical indicators ', 'arg2': ['贵州茅台', '20200606', '20230606'], 'function2': 'get_stock_technical_data', 'output2': 'result2', 'description2': ' Time series data of technical indicators of Guizhou Maotai '}

step2=: {'arg1': ['result1', 'pe_ttm'], 'function1': 'calculate_stock_index', 'output1': 'result3', 'description1': ' Ningde Time's pe value time series data ', 'arg2': ['result2', 'pe_ttm'], 'function2': 'calculate_stock_index', 'output2': 'result4', 'description2': ' Guizhou Maotai's pe value time series data '}

step3=: {'arg': ['input1', None, 'line', ' PE trend of Ningde Times and Guizhou Maotai from Jun 06, 2020 to Jun 06, 2023 '], 'function': 'plot_stock_data', 'output': 'result1', 'description': ' Plotting the PE trend of Ningde Times '}

step4=: {'arg': ['input2', 'result1', ' line ', ' PE trend of Ningde Times and Guizhou Maotai from Jun 06, 2020 to Jun 06, 2023 '], 'function': 'plot_stock_data', 'output': 'result2', 'description': 'Plotting the PE trend of Guizhou Maotai'}



Workflow

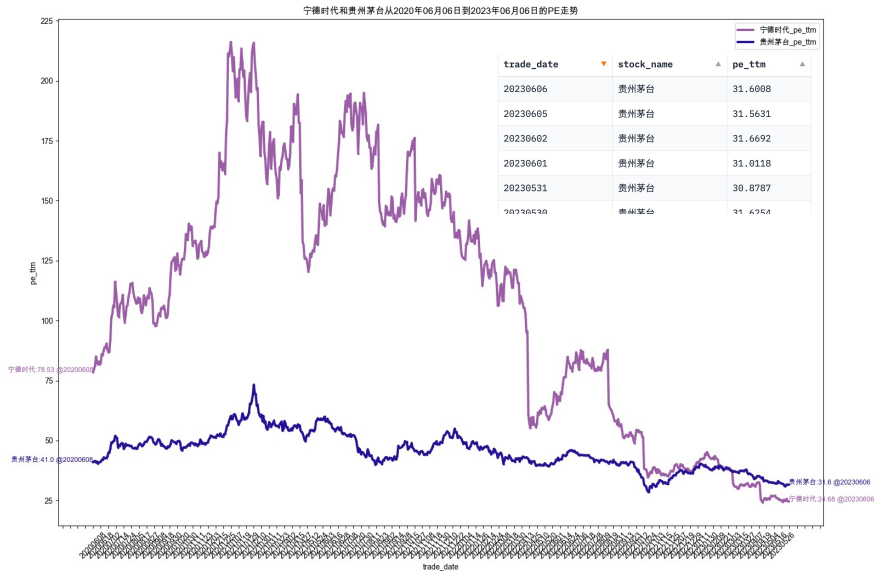
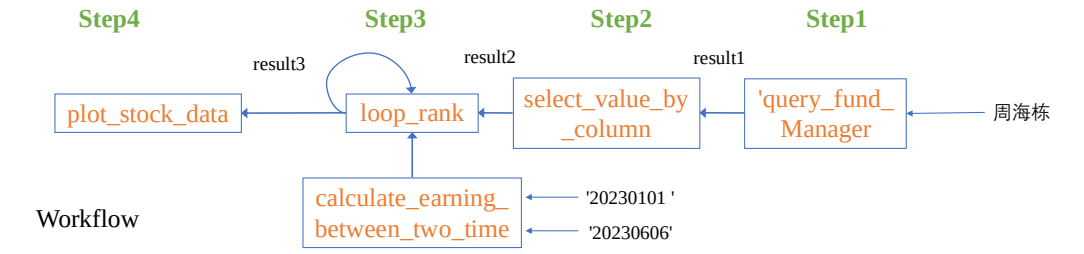


Figure F5: Data-Copilot acquires the P/E data of two stocks in parallel and plots them in a graph.

Request: Returns of all funds managed by fund manager Zhou Haidong (周海栋) this year

Intent Detecting Stage: View fund managers, showing a cross-section of returns for each fund managed by Haidong Zhou from January 1, 2023 to June 6, 2023

WorkFlow Planning :
step1=: {'arg1': ['周海栋'], 'function1': 'query_fund_Manager', 'output1': 'result1', 'description1': ' Data of all funds managed by Zhou Haidong '}
step2=: {'arg1': ['result1', 'fund_code', -1], 'function1': 'select_value_by_column', 'output1': 'result2', 'description1': ' List of codes of all funds managed by Zhou Haidong '}
step3=: {'arg1': ['result2', 'calculate_earning_between_two_time', '20230101', '20230606', 'adj_nav'], 'function1': 'loop_rank', 'output1': 'result3', 'description1': 'Cross-sectional return data from 20230101 to 20230606 for each fund managed by Haidong Zhou's'}
step4=: {'arg': ['input1', None, 'bar', ' Return of each fund managed by Haidong Zhou from January 1, 2023 to June 6, 2023 '], 'function': 'plot_stock_data', 'output': 'result1', 'description': 'Plotting fund return histogram '}



unchanged	fund_code	new_feature
fund_code	630002.OF	0.0937
fund_code	000390.OF	0.0888
fund_code	013958.OF	0.0744
fund_code	013959.OF	0.0716
fund_code	166301.OF	0.0692
fund_code	630008.OF	0.0516
fund_code	008488.OF	0.046
fund_code	001723.OF	0.0426
fund_code	010761.OF	0.036
fund_code	016049.OF	0.0335
fund_code	001959.OF	0.0312
fund_code	013142.OF	0.0296
fund_code	630011.OF	-0.0592

Figure F6: Data-Copilot also has access to fund data and can query the returns of all funds managed by the fund manager.

Request: Bank of Chengdu candlestick and KDJ indicator for the past year

Intent Detecting Stage: Display the latest financial news and market developments on financial websites

WorkFlow Planning :
step1=: {'arg1': ['成都银行', '20220606', '20230606', 'daily'], 'function1': 'get_stock_prices_data', 'output1': 'result1', 'description1': ' Bank of Chengdu daily stock price (open, close, high, low and other related prices) time series data ', 'arg2': ['成都银行', '20220606', '20230606'], 'function2': 'get_stock_technical_data', 'output2': 'result2', 'description2': ' Time series data of Bank of Chengdu technical indicators '}
step2=: {'arg1': ['result1', 'candle_K'], 'function1': 'calculate_stock_index', 'output1': 'result3', 'description1': ' Bank of Chengdu candlestick related data ', 'arg2': ['result2', 'KDJ'], 'function2': 'calculate_stock_index', 'output2': 'result4', 'description2': ' Time series data of Bank of Chengdu KDJ indicator '}
step3=: {'arg1': ['result3', 'result4'], 'function1': 'merge_indicator_for_same_stock', 'output1': 'result5', 'description1': ' Bank of Chengdu Candlestick Indicator and KDJ Indicator Combined Time Series Data '}
step4=: {'arg': ['input1', 'The K plot and KDJ index for chengduyinhang from 2022/06/06 to 2023/06/06 '], 'function': 'plot_k_line', 'output': 'result1', 'description': ' Plotting candlestick and KDJ charts '}

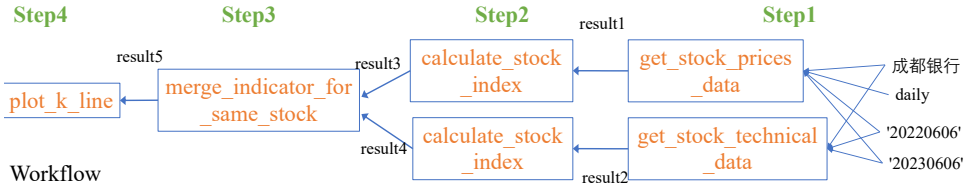


Figure F7: Data-Copilot can plot multiple indicators in a single graph by deploying workflows in parallel.

Request: I want to see the current news or the latest news

Intent Detecting Stage: Display the latest financial news and market developments on financial websites

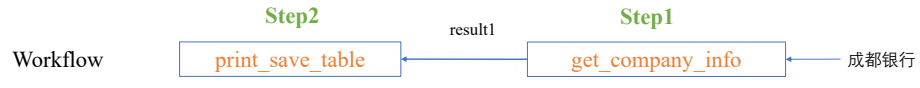
WorkFlow Planning :
step1=: {'arg1': ['sina'], 'function1': 'get_latest_new_from_web', 'output1': 'result1', 'description1': ' The latest financial news data '}
step2=: {'arg': ['input1', ' The latest financial news and market developments ', False], 'function': 'print_save_table', 'output': 'result1', 'description': ' Scroll to show news '}

datetime	content
[datetime]: 2023-06-07 04:04:14	[content]: 道琼斯指数6月6日（周二）收盘上涨10.42点，涨幅0.03%，报33573.28点； 标普500指数6月6日（周二）收盘上涨9.14点，涨幅0.21%，报4282.93点； 纳斯达克综合指数6月
[datetime]: 2023-06-07 04:02:19	[content]: 美国财政部现金余额在周一（6月5日）大幅增加至712亿美元，上周五（6月2日）为234亿美元。单日增幅为479亿美元，创4月18日以来新高。
[datetime]: 2023-06-07 04:01:34	[content]: 【热门中概股普涨 爱奇艺涨超7%】热门中概股普涨，爱奇艺涨超7%，理想汽车涨超6%，哔哩哔哩、网易涨超5%，京东、百度、唯品会涨超3%。富途控股、腾讯音乐、阿里巴巴、蔚来
[datetime]: 2023-06-07 04:01:12	[content]: 【美股三大指数集体收涨 热门科技股多数上涨】美股三大指数集体收涨，道指涨0.03%，纳指涨0.36%，标普500指数涨0.23%，热门科技股多数上涨，AMD涨超5%，英特尔涨超3%，
[datetime]: 2023-06-07 03:56:52	[content]: 【800V平台有望破解电动车续航焦虑，车企争相抢滩高压超快充赛道】 随着SiC（碳化硅）产能的持续扩张以及全球首个800V高压平台车型保时捷Taycan正式推出，高压快充路线
[datetime]: 2023-06-07 03:52:58	[content]: 市场消息：美国政府将对境内所有主要铁路线路进行安全状况评估。
[datetime]: 2023-06-07 03:52:43	[content]: 比特币/美元日内涨幅达5.0%，报26959美元。
[datetime]: 2023-06-07 03:52:34	[content]: 【两年期美债收益率涨约6个基点，前美联储副主席克拉里达称鲍威尔2024年之前不可能降息】 周二（6月6日）纽约尾盘，美国10年期基准国债收益率上涨0.97个基点，报3.6928%
[datetime]: 2023-06-07 03:43:49	[content]: 【芝加哥大豆油期货周二上涨3.9%，玉米期货也涨将近1.7%】 周二（6月6日）纽约尾盘，谷物分类指数涨0.77%，报43.7189点。CBOT玉米期货涨1.67%，报6.07-1/2美元/蒲式
[datetime]: 2023-06-07 03:35:50	[content]: 私募阿波罗（Apolo Global Management）的CEO Marc Rowan：美国遭受一场"非衰退式经济衰退"，金融市场遭受某些痛楚、而整体经济形势仍然强劲。
[datetime]: 2023-06-07 03:26:00	[content]: 加密货币持续反弹，比特币日内大涨5.00%，现报26934美元/枚，以太坊涨超4%，报1882美元/枚。
[datetime]: 2023-06-07 03:11:27	[content]: 市场消息：社交媒体Reddit将解雇5%左右的员工。
[datetime]: 2023-06-07 03:07:37	[content]: 美国联邦选举监管机构文件显示，前新泽西州州长克里斯·克里斯蒂将寻求2024年共和党总统提名。
[datetime]: 2023-06-07 03:03:26	[content]: 在岸人民币兑美元北京时间03:00收报7.1180，较上一交易日夜盘收盘跌171点。成交量452.55亿美元。
[datetime]: 2023-06-07 03:03:14	[content]: 恒指期货夜盘收涨1.91%，报19351点，高水251点。
[datetime]: 2023-06-07 03:00:37	[content]: 市场消息：由于保守派的反对，美国众议院否决了共和党保护燃气灶的投票。美国保守派共和党人称，他们对议长麦卡锡与美国总统拜登达成协议提高债务上限的做法感到愤怒，所以

Request: Introduce the Bank of Chengdu, what is the company, what is the main business

Intent Detecting Stage: Briefly introduce the listed company Bank of Chengdu and its main business

WorkFlow Planning :
step1=: {'arg1': ['成都银行'], 'function1': 'get_company_info', 'output1': 'result1', 'description1': ' Information of Bank of Chengdu '}
step2=: {'arg': ['input1', '成都银行及其相关信息', True], 'function': 'print_save_table', 'output': None, 'description': ' Information of Bank of Chengdu '



股票名称	股票代码	交易所代码	法人代表	总经理	董秘	注册资本	注册日期
成都银行	601838.SH	SSE	王晖	王涛	罗结	373573.626	1997050

所在省份	所在城市	公司介绍
四川	成都市	公司系四川省首家城市商业银行。公司以建设成为"协调发展、富有特色、价值领先

Figure F8: Data-Copilot can provide the latest financial news and company information by deploying the corresponding workflows.

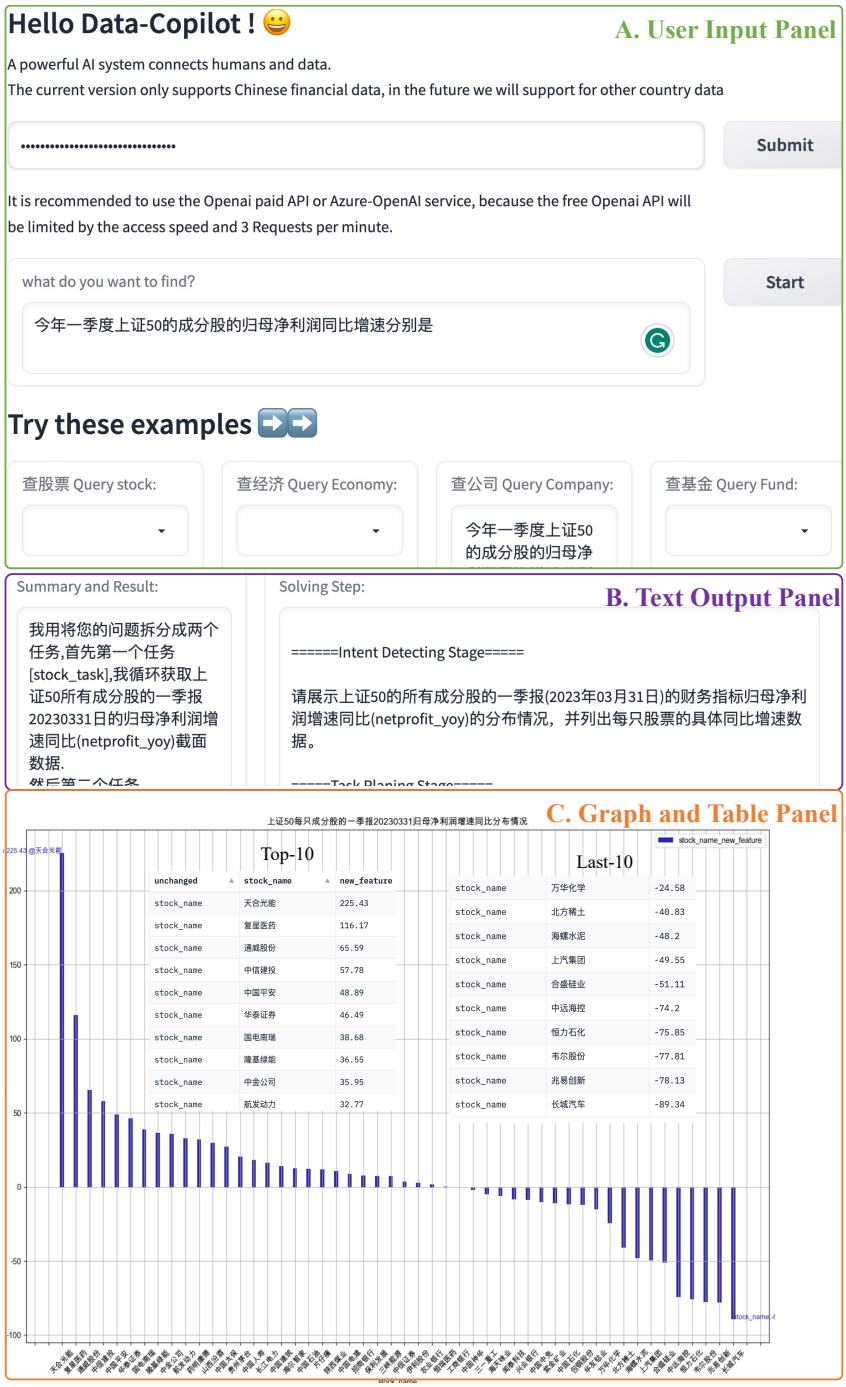


Figure F9: The user interface of our system. The green box (A) is the user input panel, and the purple (B) and red parts (C) are the results returned by the system.