DATA-COPILOT: BRIDGING BILLIONS OF DATA AND HUMANS WITH AUTONOMOUS WORKFLOW

Wenqi Zhang, Yongliang Shen, Weiming Lu†, Yueting Zhuang
College of Computer Science and Technology, Zhejiang University
{zhangwenqi, luwm}@zju.edu.cn

https://github.com/zwq2018/Data-Copilot

ABSTRACT

Various industries such as finance, meteorology, and energy produce vast amounts of heterogeneous data every day. There is a natural demand for humans to manage, process, and display data efficiently. However, it necessitates labor-intensive efforts and a high level of expertise for these data-related tasks. Considering large language models (LLMs) showcase promising capabilities in semantic understanding and reasoning, we advocate that the deployment of LLMs could autonomously manage and process massive amounts of data while interacting and displaying in a human-friendly manner. Based on this, we propose Data-Copilot, an LLM-based system that connects numerous data sources on one end and caters to diverse human demands on the other end. Acting as an experienced expert, Data-Copilot autonomously transforms raw data into multi-form output that best matches the user's intent. Specifically, it first designs multiple universal interfaces to satisfy diverse data-related requests, like querying, analysis, prediction, and visualization. In real-time response, it automatically deploys a concise workflow by invoking corresponding interfaces. The whole processes are fully controlled by Data-Copilot, without human assistance. We release Data-Copilot-1.0 using massive Chinese financial data, e.g., stocks, funds and news. Experiments indicate it achieves reliable performance with lower token consumption, showing promising application prospects.

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. This wide-ranging, multiform data encapsulates critical insights that could be leveraged for a host of applications, from predicting financial trends to monitoring energy consumption.

Recently, the advancement of large language models (LLMs) (Chowdhery et al., 2022; Zhang et al., 2022; Zeng et al., 2023; Touvron et al., 2023), particularly the ChatGPT (OpenAI, 2022) and GPT-4 (OpenAI, 2023a), has paved the way for advanced AI systems. Leveraging chain-of-thought (Wei et al., 2022c; Kojima et al., 2022; Gao et al., 2022b; Wang et al., 2023c), reinforcement learning from human feedback (Ouyang et al., 2022; Wang et al., 2022b), and instruction-following (Wei et al., 2022a; Iyer et al., 2022; Chung et al., 2022), LLMs demonstrate remarkable abilities in dialogue, reasoning and planning. However, in the face of the sheer magnitude and complexity of data, LLMs are confronted with the colossal challenge of managing, processing, and displaying data.

To achieve this, several challenges must be addressed: (1) From a data perspective: employing LLMs for directly reading and processing massive data is not only impractical but also poses the potential risks of data leakage. (2) From the model perspective: due to limitations of the input context, LLMs are unable to process large-scale data at once and are also not adept at handling numerical computations. (3) From the task perspective: many data-related tasks are intricate and require a combination of many operations, like retrieval, computations, and table manipulations, and present

†Corresponding author.

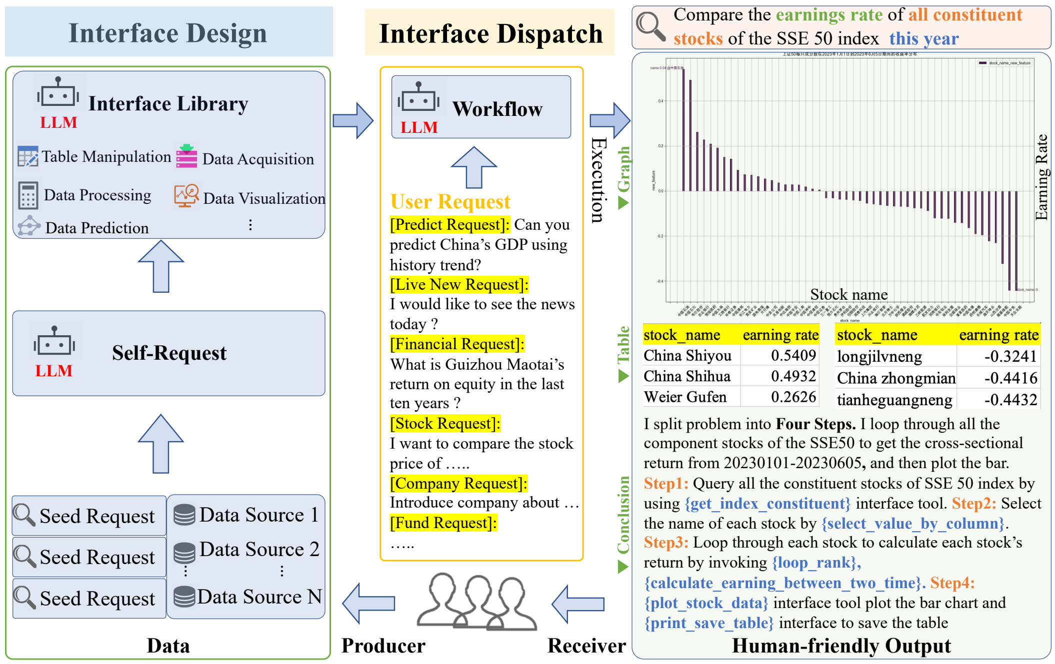


Figure 1: *Data-Copilot is an LLM-based system for data-related tasks, bridging billions of data and diverse user requests*. It designs versatile interface tools for data management, invocation, processing, and visualization. Upon receiving a complex request, Data-Copilot autonomously invokes self-design interfaces and constructs a workflow to fulfill human intent. Lastly, raw data is converted into graphics, tables, and text forms tailored to human tastes.

their results in multiple formats, e.g., images, tables, and text, which exceeds the capabilities of LLMs. These challenges significantly constrain the application of LLMs in data-related tasks.

Recently, many works have embarked on valuable explorations. LiDA (Dibia, 2023) and GPT4-Analyst (Cheng et al., 2023) focus on automated data exploration. Sheet-Copilot (Li et al., 2023b), BIRD (Li et al., 2023c), DIN-SQL (Pourreza & Rafiei, 2023) and DB-GPT (Xue et al., 2023) apply LLMs to Text2SQL. However, our focus is more on how to better process and present data based on human preference, rather than merely retrieving it from databases. Additionally, while SQL is convenient, it can not directly satisfy common data analysis needs such as prediction and visualization.

In light of this, we contemplate **how to integrate LLMs into every facet of data-related tasks**, not just as a Text2SQL solver. More specifically, we explore how LLMs can efficiently and autonomously process vast amounts of data to meet diverse human needs, without the necessity of direct data access. To pursue the answer, we trace back to the origins of data science in the 1970s. The pioneer of data science, Peter Naur (Turing Award winner in 2005), defined data science as follows:

> *Data science is the science of dealing with data and processing large amounts of data. Humans as sources and receivers of data. The data must be chosen with due regard to the transformation to be achieved and the data processing tools available. —Naur, 1974*

His statement elucidates how humans accomplish data-related tasks: humans create various tools that are easy to use and then transform the data into a human-friendly format. Similarly, LLMs also should play a more intelligent role like humans: designing tools for data management, and invoking tools for processing and presenting data.

Additionally, numerous LLM-based agent methods have recently emerged (Wu et al., 2023a; Huang et al., 2023; Chen et al., 2023b; Hong et al., 2023; Wu et al., 2023b). These LLM-based agents showcase the potential of LLMs in meeting human needs through code as an intermediary.

Inspired by them, we advocate utilizing the coding capabilities of LLMs to create diverse and powerful tools, thereby facilitating the management and utilization of vast amounts of data. Based on this, we propose **Data-Copilot**, a reliable data agent that manages vast data and dynamically fulfills a wide spectrum of human needs. As shown in Figure 1, Data-Copilot autonomously designs versatile interface tools for **data management**, and dispatch these interface tools step by step for diverse human requests, e.g., **data analyzing, forecasting, and visualization**, forming a data-to-human workflow.

As an LLM-based agent, Data-Copilot is **versatile**. It seamlessly connects an array of data sources from various domains on one end, while adeptly catering to diverse user demands on the other. Furthermore, Data-Copilot is **scalable**. It can continuously extend its capability boundaries by developing new interface tools for emerging new data and requests. Our system comprises two processes:

- **Interface Design:** Data-Copilot adopts an iterative self-request process to fully explore the data, mimicking a broader range of human need scenarios. As shown in Figure 1, starting from a few seed requests, it self-generates a large number of requests and then abstracts them into generalized interface tools for efficient utilization. These interfaces empower Data-Copilot to efficiently perform data querying, processing, forecasting, and visualization.
- **Interface Dispatch:** When a user request is received, Data-Copilot first parses the user intention and then plans an interface invoking workflow for the user request. This step-by-step invocation can efficiently fulfill user requirements while minimizing token overhead.

Incorporating two phases, Data-Copilot achieves autonomous data management and utilization without the need for direct data access. Our contributions can be summarized as follows:

1. To efficiently handle data-related tasks, we design a universal system, **Data-Copilot**, that connects massive data sources to diverse user tastes. It leverages LLM for more efficient data management and utilization, reducing tedious labor.
2. Data-Copilot can autonomously manage, process, analyze, and visualize data. For a human request, it transforms raw data into informative results that best match the user's intent.
3. As a reliable assistant, Data-Copilot abstracts a multitude of common needs into various practical interface tools. When faced with real-time requests, it invokes these related interfaces sequentially or in parallel for problem-solving.
4. We evaluate Data-Copilot on massive Chinese financial data, e.g., stocks, funds, macroeconomics, and news. The excellent success rate and lower computational cost showcase its potential and application prospects.

2 DATA-COPILOT

Data-Copilot is an LLM-based agent that accomplishes automated management, invocation, and processing of a large number of data by self-design sophisticated interface tools. Besides, Data-Copilot adeptly comprehends user intentions, and invokes well-designed tool interfaces for request solving, significantly reducing the need for tedious labor.

As shown in Figure 2, the LLM plays the role of a designer (Section 2.1) to independently design various tool interfaces to for data management and utilization. Besides, LLM also acts as a dispatcher (Section 2.2), invoking generated interfaces and deploying the workflow for problem-solving. Through the integration of two stages, Data-Copilot manages to efficiently handle a large volume of data-related tasks and caters to various user needs.

2.1 INTERFACE DESIGN

Just as Peter Naur pointed out, designing a set of versatile interface tools that are easy to manage and use is critical. Therefore, Data-Copilot designs a plethora of interfaces as tools, where each interface is composed of natural language (functional description) and code (implementation), responsible for data acquisition, processing, and visualization.

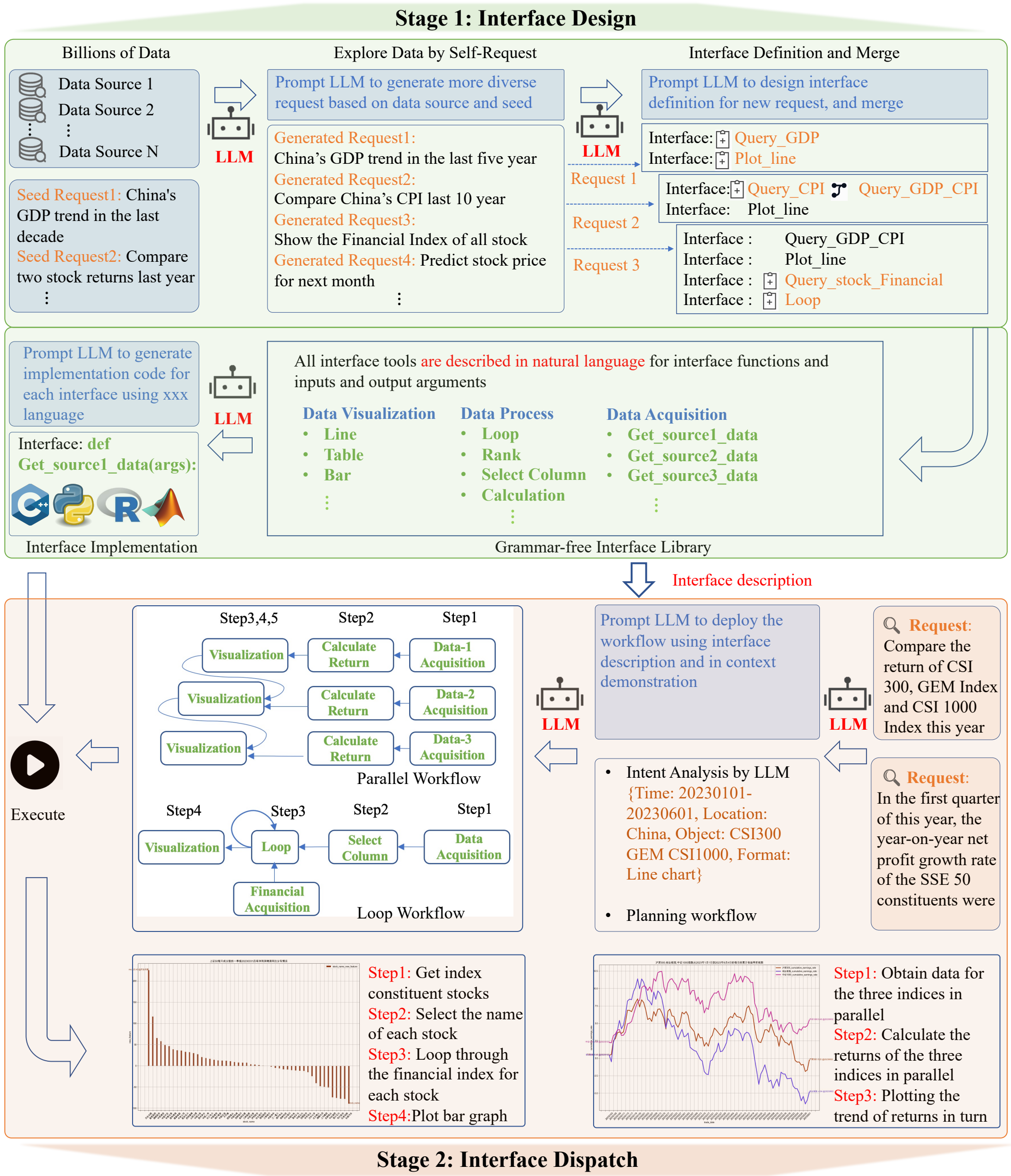


Figure 2: Overview of Data-Copilot. **Interface Design:** We devise a self-request process allowing LLM to generate sufficient requests from a few seed requests autonomously. Then LLM iteratively designs and optimizes interfaces based on generated requests. These interfaces are described in natural language, making them easily scalable and transferable across different platforms. **Interface Dispatch:** Upon receiving user requests, LLM plans and invokes interface tools based on self-design interface descriptions and in-context demonstrations. This allows for the deployment of a logical workflow that fulfills user demands and presents the results to the user in multi-form.

As shown in Figure 2, beginning with some seed requests, LLM autonomously mimics and generates a large number of requests. Then LLM designs common interfaces for these requests, representing a set of universal needs. By gradually refining and expanding their functionalities, these interfaces progressively cover a more diverse array of needs. Ultimately, the LLM translates these interfaces into executable and callable code modules.

Explore Data by Self-Request The design of the interface depends on two aspects: what kind of data is available and also what kind of demand the user proposes. Inspired by (Wang et al., 2022a; Dibia, 2023), Data-Copilot first autonomously explores the data to mine more requests. Then it proceeds to design the interfaces for these requests.

Specifically, we 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. This process does not require too much labor, as data providers usually offer a description of the data and access methods.

Then we adopt a self-request process to explore data, i.e. the parsing files and a few seed requests are fed into the LLM and then LLM is instructed to generate more diverse requests by mimicking. This process aims to explore an extensive array of data sources while covering diverse user requests. In Figure 2, LLM generates four new requests based on the two seed requests and data parsing files. The detailed prompts are shown in Appendix F.1.

Interface Definition In this step, Data-Copilot creates various interface tools to fulfill the previously generated requests. Specifically, we feed all data parsing files and all interfaces stored in the interface library (empty at first iteration) into LLM as a prompt. LLM needs to resolve each request one by one. When addressing each request, LLM can use existing interfaces from the interface library or redefine a new interface to fulfill the task.

Concretely, if existing interfaces are insufficient to address the current request, LLM can create a new interface: `Interface1={Name:"", Function description:"", Input and Output:""}...` After that, LLM proposes a feasible solution using these self-design or existing interfaces: `Solution="These three interface functions will meet your needs. You should first use the interface: {getBankData} to acquire data, then..."`. The prompts and cases are shown in Appendices F.1 and G.1.

Importantly, Data-Copilot only uses natural language to define the interface's functions and their arguments, without considering the specific implementation, which is grammar-free. This process is similar to software architecture design by a human programmer. It allows Data-Copilot to focus on designing the layout and relation of the interfaces with different functionalities, rather than complying with programming grammar.

Interface Merging To make the interface more universal, Data-Copilot considers whether each newly designed interface can be merged with the existing ones in the library. Specifically, when a new interface is designed, Data-Copilot also checks whether this interface is similar to the previous ones, in terms of functionality, arguments, etc. Two similar interfaces are merged to create a new and generalized interface. As shown in Figure 2, two interfaces {`Interface: Query-GDP`} and {`Interface: Query-CPI`} are merged into one {`Interface: Query-GDP-CPI`}. This merging process is similar to software developers wrapping similar modules into a new one.

After this process, a large number of similar interfaces are merged. Each interface within the library possesses distinct functionalities, which are adeptly suited for real-time response applications.

Interface Implementation Data-Copilot iteratively processes each request through interface definition and interface merging. Eventually, when all the requests can be satisfied by these self-design interfaces, Data-Copilot transforms these interface tools into executable and callable code modules.

The whole interface design process is offline. As shown in Figure G3, Data-Copilot automatically produces five types of interfaces in the interface library: data acquisition, processing, prediction, visualization, and DataFrame manipulation. It transforms repetitive and tedious labor into an automated process, and also can effortlessly add new interfaces for additional requests or new data sources. Besides, Data-Copilot easily switches to other programming platforms and databases by simply re-generating the implementation code for the interface, demonstrating excellent scalability.

2.2 INTERFACE DISPATCH

In the first stage, Data-Copilot designs a variety of generic interface tools for data acquisition, processing, and visualization. These interfaces are versatile and easy to use, improving the efficiency of real-world response. These interfaces are invoked in this phase and form an efficient workflow for real-time requests (Figure 2). A detailed prompt and cases are shown in Appendices F.2 and G.2

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, if the question 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 2017 to May 2023 for comparison"`. To achieve this, we first invoke an external API to obtain the local time and network IP address, then feed this external information into LLM along with the original request to generate the parsed result.

Planing Workflow Once the user's intent is accurately understood, Data-Copilot plans a reasonable workflow to process the user's request. We feed the user request and related interface descriptions into LLMs for the automatic deployment of workflows. Which interfaces should be invoked, and in what sequence, is entirely determined by Data-Copilot, based on the user's request and the interface.

To achieve this, Data-Copilot generates a response in `JSON` format representing each step of the scheduling: `###Interface Call: {step1={"arg":"", "function":"", "output":"" ,"description":""}, step2={..}, ..}`. Please refer to Appendix F.2 for a detailed prompt. Data-Copilot meticulously orchestrates the scheduling of the interface in either a sequential or parallel manner.

For instance, the first request in Figure 2: `"Compare the return of CSI 300, GEM Index and CSI 1000 Index this year"`, Data-Copilot first plans five steps in its workflow. In the first two steps, it dispatches {`Data Acquisition`}, {`Calcuate Return`} **in parallel** for three indices. The last three steps successively invoke the {`Visualization`} to sequentially plot three indices on the same canvas. In the second case, Data-Copilot deploys **a loop workflow** to sequentially calculate the indicators of 50 constituent stocks by calling the {`Loop and Rank`}.

Multi-form Output Upon the deployment and 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. This systematic summary not only provides clear and concise results but also sheds light on the steps taken to achieve those results, thereby enhancing transparency and understanding of user requests.

As the example shown in Figure G2, the request is `"Forecasting China's GDP growth rate..."`. Data-Copilot first interprets the user's intent based on local time. 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.

3 Qualitative Evaluation

3.1 Environment and Data Sources

We evaluate Data-Copilot utilizing Chinese financial market data, encompassing massive stocks, funds, economic data, real-time news, and company financial data. To closely mimic real-world scenes, we adopt the data interfaces (APIs) provided by `Tushare`[1] to access data, which are commonly used by many financial companies. The vast and diverse array of financial time-series data enables a comprehensive evaluation of Data-Copilot's performance. In the first phase, we use the `gpt-4-0613`[2] for interface design and `gpt-3.5-turbo-0613` for interface dispatch.

[1] `https://tushare.pro/`

[2] `https://platform.openai.com/`

Table 1: Left: Statistics on human-proposed requests and Self-Request dataset. Right: Detail Statistics on different request types and workflow structures.

Source	#Cases	Form	Usage
Human-proposed	173	Query	Seed
Self-Request	3000	Query	Interface Design
Self-Request + Human-annotated	547	Query, Ans	Testing

Request Types	Workflow Types			
	Sequential	Parallel	Complex	Overall
Stock	79	106	87	272
Fund	55	36	22	113
Corp.	97	30	4	131
Other	6	12	13	31
Total	237	184	126	547

3.2 Seeds and Datasets

Human-proposed Request as Seed We invite 10 students in economics to submit 50 requests each. These requests should satisfy two requirements: 1) cover a wide range of common needs, including stocks, funds, and corporate. 2) involves different levels of difficulty. Then we filter out highly similar requests by computing the sentence embedding of each request. Lastly, we retain 173 high-quality requests, which are used as the seeds for `Interface Design` phase of the Data-Copilot.

Self-Request and Testing Benchmark As mentioned in Section 2.1, we employed a self-request strategy to expand the request set. Specifically, we feed 173 manually proposed seed requests along with all data descriptions and schema into GPT-4. We instruct GPT-4 to generate massive requests, covering diverse data. In this phase, we generate a total of 6480 requests. Then we compute the semantic similarity between all requests and filter out those with high similarity. Lastly, we retain 3547 requests. We randomly select 3000 requests from the total set for subsequent interface design. Meanwhile, the remaining 547 requests are treated as a test set. We manually annotate the final answers, task types, and workflow types for each testing instance. The statistics of three dataset are shown in Table 1 and Appendix C.

3.3 Qualitative Results

The whole system is shown in Figure G9, where Data-Copilot dynamically displays the intermediate process (`Solving Step`) and the final results (bar, chart, text) for the user request. The invocation workflow usually consists of multiple steps, each of which invokes one or multiple interfaces. For example, Data-Copilot addresses the request in Figure 2 (`"Compare the return of ..."`) via a five-step procedure, wherein the first two steps invoke the same interface in parallel.

Data-Copilot designs many versatile interface tools through a self-request. As shown in Figure G3, each interface is clearly defined and versatile, including data acquisition, processing, dataframe manipulation, and visualization. These diverse interfaces endow the LLM with the capability to address complex requests. For instance, `get-stock-prices-data` interface integrates the functions of acquiring data in daily, weekly, and monthly frequencies, controlled by argument: `freq`, which is very similar to the process of programmer development.

Data-Copilot deploys workflows with different structures. Data-Copilot invokes the most appropriate interface and forms an efficient workflow with different structures, e.g., parallel, serial, or loop workflows, tailored to user requests. As shown in Figures G4 and G5, it employs a loop workflow for the requests: `"The year-on-year net profit growth rate of the SSE50 ..."` by invoking `Loop-Rank` interface. It iteratively calculates the financial indicators for each stock. Please refer to Appendix E for more visualization cases.

4 Quantitative Evaluation

We quantitatively assess Data-Copilot in processing real-time user requests, where it could invoke the interfaces designed in the first phase.

Table 2: All methods use `gpt-3.5` as based LLM.

Methods	Exec@1(%)	Pass@1(%)	#Token
GP	50.8	28.5	823.6
ReAct	63.4	44.1	1515.9
Reflexion	75.7	59.0	2463.5
Multi-Agent	73.6	57.4	2835.2
Data-Copilot	**79.9**	**70.2**	**561.2**

Table 3: All samples are divided into three subsets based on its complexity. We compute Pass@1.

Methods	Sequential	Parallel	Complex	Overall
GP	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%**	**70.0%**	**67.1%**	**70.2%**

4.1 EXPERIMENTS SETTINGS

Baselines 1. Global Planning (GP) (Shen et al., 2023; Qin et al., 2023a): We instruct the LLM to directly generate solving codes for user requests. 2. ReAct (Yao et al., 2022): The LLM resolves the user request using neural language and code step-by-step. 3. Reflexion (Shinn et al., 2023): LLM can also reflect on the feedback from the compiler at each step, progressively fulfilling user requests. For the Reflexion strategy, we limit it to a maximum of two rounds of reflection. 4. Multi-agent collaboration (Wu et al., 2023b; Hong et al., 2023; Liang et al., 2023a): We adopt three LLM-based agents, i.e., two coders and a manager, handling user requests simultaneously and outputting the final solution after mutual discussion. All methods are evaluated on the 547 testing instances (Section 3.2) three times. A detailed prompts of baselines is shown in Appendix F.3.

Metrics We design three metrics for solution code and answer: Exec@1 Pass@1 and Efficiency. Exec@1 evaluates whether the code can be executed by a compiler without errors. Pass@1 measures the correctness of the final answer. Considering the diverse output formats, e.g., images, text, numbers, and tables, we employ GPT-4 to automatically assess the answer. If the output is an image, we use `gpt-4-vision-preview` to compare the generated image to the label. Lastly, we calculate the total number of tokens consumed by the input prompt and output response, which denotes the solving efficiency of each method.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

4.2 COMPARISON RESULTS

As shown in Table 2, Data-Copilot surpasses all baselines in success rate (Pass@1), Exec@1, and Efficiency, achieving 11.2% improvements.

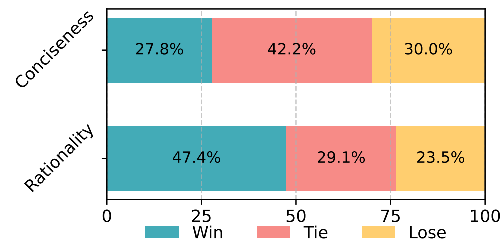
Data-Copilot significantly reduces the risk of failure. Compared to the best baseline (Reflexion), our method increases the success rate from 59% to 70.2%. In most cases, our system merely needs to invoke pre-designed interface tools and generate corresponding arguments to address user requests, significantly reducing the risk of errors. In contrast, baseline strategies must construct all code from scratch, which makes it easy to overlook certain details or make mistakes. Besides, the difference between Pass@1 and Exec@1 also indicates that baseline strategies often miss certain steps, whereas Data-Copilot can fully interpret user requests and methodically complete tasks step by step.

Data-Copilot reduces repetitive overhead for common requests. The consumed tokens represent computational overhead. Data-Copilot only consumes 561 tokens, which is just 30% of the consumption compared to baselines. Specifically, the output of Data-Copilot is very concise, only containing interface names and arguments. The primary token consumption lies in the input prompt which contains related tool descriptions. In contrast, baseline strategies may consume a substantial amount of tokens since they need to output a complete solving code for each request. In the real world, numerous requests are similar or even repetitive. Data-Copilot can reduce repetitive overhead

Figure 3: Results of Data-Copilot with various LLMs.

LLM	Pass@1 (%)	#Token
gpt-4	75.6	561.2
gpt-3.5-turbo	70.2	559.3
Llama-2-70B-chat	49.3	623.6
codellama-13B	50.8	507.4
text-davinci-003	47.5	495.9
Llama-2-13B-chat	36.9	443.8
Vicuna-13B-v1.5	33.2	502.8

Figure 4: Win rate for Data-Copilot Vs. Reflexion on human preferences.



by abstracting common demands into interface tools. However, baseline methods are compelled to repetitively generate code for each request.

4.3 Analysis

Data-Copilot exhibits a pronounced advantage on more challenging tasks. To better analyze the problem-solving capabilities of Data-Copilot, we categorize our dataset into three subsets based on the complexity of the workflow: Sequential, Parallel, and Complex. As shown in Table 3, for sequential tasks, the baselines and our method perform similarly, with only a 4% improvement. However, for the more challenging parallel or complex tasks, our improvements are substantial, with enhancements of 5.3% and 19%, respectively. We find that baseline methods often struggle to fully implement complex functionalities and may omit some critical steps, such as loop processing. In contrast, our methods effectively handle these tasks by invoking versatile interfaces and deploying workflow. These findings elucidate the advantages of Data-Copilot, which reduces task complexity.

We can deploy Data-Copilot locally using small LLMs with commendable performance. We analyze the performance of Data-Copilot with different LLMs. As shown in Figure 3, we test GPT family APIs and various open-source models. We observe GPT-4 surpasses all LLMs, achieving 75.6% Pass@1, while many open-source LLMs also achieve commendable results. For instance, codellama-13B achieves a 50.8% Pass@1, almost similar to the best baseline in Table 2 (Reflexion with `gpt-3.5-turbo`). This indicates that Data-Copilot can be deployed locally at a relatively low cost while maintaining commendable performance.

4.4 Human Evaluation

To assess whether the solutions align with human preferences, we perform human evaluations across two dimensions: rationality and conciseness. Specifically, we collect the solutions of Data-Copilot and Reflexion. Then we invite experts to compare which solution is better in terms of rationality and conciseness (detail in Appendix C). We present results in Figure 4. It demonstrates that our solution significantly outperforms the competitor in terms of rationality. Specifically, Data-Copilot prevails in 47.4% of instances, trailing behind the competitor in only 23.5%. Besides, as for conciseness, Data-Copilot lags slightly behind its competitor, with 27.8% wins against 30% losses. This is attributed to the fact that these pre-designed interfaces invoked by Data-Copilot are quite universal, accommodating a wide array of functions, which makes our system less concise. The results of human evaluation are consistent with the previous assessment. Although Data-Copilot sacrifices a bit of succinctness, it can generate more rational and correct solutions.

5 Conclusion

We propose a universal framework, Data-Copilot, to address a wide array of data-related tasks. It acts as a bridge between numerous heterogeneous data and humans, effectively managing, processing, and presenting data according to human tastes, reducing the dependence on tedious labor and expert knowledge. Acting like an experienced expert, Data-Copilot autonomously designs universal interface tools for various types of data and potential user demands and invokes these interfaces in real-time response for more reliable problem-solving.

6 ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China (No. 62376245), the Key Research and Development Program of Zhejiang Province, China (No. 2024C01034), the project of the Donghai Laboratory (Grant no. DH-2022ZY0013), and MOE Engineering Research Center of Digital Library.

REFERENCES

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, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *NeurIPS*, 2020.

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

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

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

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

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

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

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416, 2022.

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

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

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

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

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

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

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. Metagpt: Meta programming for a multi-agent collaborative framework, 2023.

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

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

S. Iyer, Xiaojuan Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, Xian Li, Brian O'Horo, Gabriel Pereyra, Jeff Wang, Christopher Dewan, Asli Celikyilmaz, Luke Zettlemoyer, and Veselin Stoyanov. Opt-IML: Scaling Language Model Instruction Meta Learning through the Lens of Generalization. *ArXiv*, abs/2212.12017, 2022.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. StructGPT: A general framework for large language model to reason over structured data. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9237–9251, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.574. URL `https://aclanthology.org/2023.emnlp-main.574`.

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

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

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

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

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

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

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. Taskmatrix.ai: Completing tasks by connecting foundation models with millions of apis, 2023b.

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

OpenAI. Chatgpt. 2022.

OpenAI. Gpt-4 technical report. 2023a.

OpenAI. Gpt-4 technical report, 2023b.

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. Training language models to follow instructions with human feedback. *CoRR*, abs/2203.02155, 2022.

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

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

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

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. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023b.

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

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

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

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

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

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. Llama: Open and Efficient Foundation Language Models. *ArXiv*, abs/2302.13971, 2023.

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

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ICLR 2023 poster*, abs/2203.11171, 2023c.

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

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, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, rushang karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. 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, 2022b.

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

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022a. URL `https://openreview.net/forum?id=gEZrGCozdqR`.

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. Emergent abilities of large language models. *CoRR*, abs/2206.07682, 2022b.

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

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

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

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

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

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. Db-gpt: Empowering database interactions with private large language models. *arXiv preprint arXiv:2312.17449*, 2023. URL `https://arxiv.org/abs/2312.17449`.

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

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. Glm-130b: An Open Bilingual Pre-trained Model. *ICLR 2023 poster*, 2023.

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. Opt: Open Pre-trained Transformer Language Models. *ArXiv*, abs/2205.01068, 2022.

Appendix

A Discussion

A.1 What is Data-Copilot?

- Data-Copilot is an efficient LLM-based data management agent, that processes massive data and rapidly responds to real-time human requests.
- It contains interface design and invocation phases. Interface design involves a self-exploration phase for data sources to create many versatile interfaces.
- For interface invocation, it parses human real-time requests and invokes the designed interfaces to retrieve and process relevant data and generates multi-form outputs, e.g., text, charts, and tables.

A.2 Why Data-Copilot is designed in two stages?

- **Reduces redundant generation:** People often have a large number of similar or repetitive query needs, e.g., many people are interested in seeing which stock has the largest gain today. For these repetitive query requests, employing the LLM to write code from scratch for each user request would be highly inefficient and a significant waste of computational resources. To avoid inefficient and repetitive code generation for numerous similar requests, we abstract common demands into generic interfaces through data self-exploration in advance (Phase-1). For real-time requests (Phase-2), it only needs to invoke these interfaces, reducing redundant generation.
- **Performs better and consumes fewer tokens:** Compared to a complete code, it only needs to invoke the relevant interfaces, which is simpler and less prone to errors. Our results show that it performs better and consumes fewer tokens.
- **Stronger interpretability:** Compared to complex code, interface workflow is easier for humans to read and inspect.

B Experiments Details

During the `interface design` phase, we instruct LLM to generate Python code for each interface in the `interface implementation`. We filter out interfaces that cannot run properly at each iteration. For the `interface Dispatch` phase, we carry out hierarchical planning to improve efficiency: upon receiving a request, Data-Copilot first determines the type of data task involved (stock task, fund task, etc.) and then loads the corresponding interface description, followed by interface planning using the corresponding interface. We set the temperature to 0.2 for more consistent results.

C Human Evaluation Details

We collect the solutions of Data-Copilot and Reflexion on 173 human-proposed requests. Subsequently, we involve experts to evaluate which of the two solving codes is more rational and concise. The criteria for expert are as follows:

```
Rationality:
1) Accurately identifies data of interest to the user.
2) Processes the data logically.
3) Output data in a format that meets human needs, with aesthetic and
concise presentation.

Conciseness:
1) Only extracts and computes data relevant to the user's interest.
2) Code is clear, concise, and well-structured.
3) The solution process is devoid of repetition and redundancy, ensuring
a straightforward and intuitive approach.
```

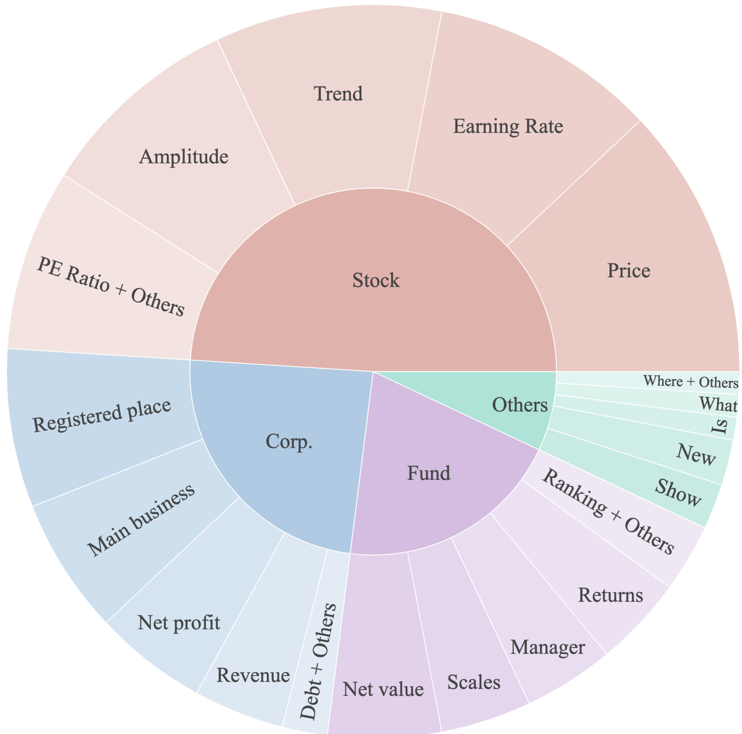


Figure C1: We count the keywords for each request type (Stock, Corp., Fund, Others) in the test set.

Experts are required to evaluate which of the two solutions is better based on these principles, and then we calculate the win rate of Data-Copilot against the Reflexion.

D 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, 2023b; Wei et al., 2022b; Wu et al., 2023c) 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., 2022a; Qin et al., 2023a; Hao et al., 2023; Qin et al., 2023b) 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[3], AgentGPT[4], BabyAGI[5], BMTools[6], LangChain[7], 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

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

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. 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) focus on automated data exploration. Besides, many reseraches (Liu et al., 2023; Chang & Fosler-Lussier, 2023; Dong et al., 2023), like Sheet-Copilot (Li et al., 2023b), BIRD (Li et al., 2023c), DAIL-SQL (Gao et al., 2023), DIN-SQL (Pourreza & Rafiei, 2023), DB-Copilot (Wang et al., 2023b), MAC-SQL (Wang et al., 2023a), ChatDB (Hu et al., 2023), SQL-PaLM (Sun et al., 2023), and DB-GPT (Xue et al., 2023) 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.

Distinct from these approaches, our Data-Copilot system is different in the following aspects:

(1) Different Task Settings: Unlike Text2SQL methods, Data-Copilot focuses more on how to utilize data and present it according to user preferences, rather than retrieving data from databases. We assume that Data-Copilot has access to a vast amount of data, which is available via API.

(2) Richer Usage Scenarios: Data-Copilot needs to meet a diverse range of human needs, such as forecasting and plotting. Therefore, it employs programmable language like Python for data processing, offering more varied use cases than SQL.

(3) Distinct Implementation Approach: Data-Copilot designs its tools for data processing, rather than relying on SQL or pre-existing tools. This approach is more flexible and can meet complex user requirements.

E 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 G4,G5, Data-Copilot deploys different structural workflows based on user requirements. In Figure G4, 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 G5, 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 interface design and dispatch process of Data-Copilot are rational and effective.

Diverse Sources Figure G6, G7, G8 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.

F DETAILED PROMPTS

We provide detailed prompts for our Data-Copilot and baselines. Specifically, for the interface design, we provide detailed prompts in Appendix F.1 for four phases: Self-Request, Interface Definition, Interface Merging, and Interface Implementation. For the interface dispatch, we also provide prompts for three key procedures in Appendix F.2: Intent Analysis, Task Selection, and Planning Workflow. Additionally, we outline prompts for all baselines in Appendix F.3: Global Planning, ReAct, Reflection, and Multi-Agent Collaboration strategies.

F.1 Prompts For Interface Design

```
Explore Data by Self-Request 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,...}
###Data files:{
    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, ...}
        }
    },
    stock_basic:...,
}

Interface Definition Phase:
###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 (similar to pseudocode) 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 do not write any implementation code in the
new interface, just design the interface as existing interfaces.
(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 files}
###The existing interfaces: {all interfaces in library}
```

```
Interface Merging Phase
###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. Finally
(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}
All interfaces after merging: {interface1, interface2, merged interface3,
...}
```

```
Interface Implementation Phase:
###Instruction: Please generate the complete code according to the
definition of a given interface and data access methods. The generated
code must be in strict accordance with the definition of the interface
and the formatting requirements of the arguments.
###Interface: {Input Interface}
###Data Files: {All data files}
```

G CASE STUDY

F.2 Prompts For Interface Dispatch

```
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.
```

F.3 PROMPTS FOR BASELINES

```
Global Planning
###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}
```

G.1 CASES FOR INTERFACE DESIGN

```
---- The First Request:  Interface Definition Phase -----
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=Interface Name:getBankData, Function:.., Input:.., Output:..}
{Interface2=Interface Name:prepareDataForPlotting, Function:.., Input:..,
Output:..}
{Interface3=Interface 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.
---- 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=Interface Name:  getStockData..}

Existing Interfaces
{Interface1=Interface Name:getBankData, ...}
{Interface2=Interface Name:prepareDataForPlotting, ...}
{Interface3=Interface 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...
---- Interface Merge 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-stock-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-stock-data, ...}
{Interface2=Interface Name:prepareDataForPlotting, ...}
{Interface3=Interface Name:plotData, ...}
---- Interface Implementation -------
Using get_stock_data as examples:
def get_stock_data(name:str='',start_d:str='',end_d:str='',freq:str)->pd.
DataFrame:
    """
    Retrieves the daily/weekly/monthly price data for a given stock code
    during a specific time period.
    Args:
    -name (str): The name of the stock.
    -start_d (str): The start date in the format 'YYYYMMDD'.
    -end_d (str): The end date in 'YYYYMMDD'.
    -freq (str): The frequency of the price, can be 'daily', 'weekly', or
     'monthly'.
    Returns:
    -pd.DataFrame: A dataframe that contains the daily/weekly/monthly
    data. The output columns contain stock_code, trade_date, open, high,
    low, close, pre_close, change), pct_chg,vol,amount
    """
    stock_code = get_stock_code(name)
    if freq == 'daily':
    .....
```

G.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

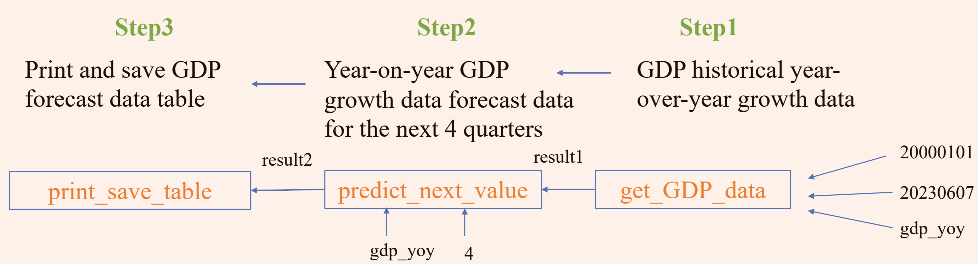
Intent Analysis: Show data and print tables for the next 4 quarters based on China's GDP growth rate for each quarter from 20000101 to June 07, 2023 (today)

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 G2: 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

Category		
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
	Name:	select_value_by_column
	Function:	Selects a specific column or a specific value within a DataFrame
	Input/output:	(df1:pd.DataFrame, col_name: str = '', row_index: int = -1) -> 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 G3: 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'}

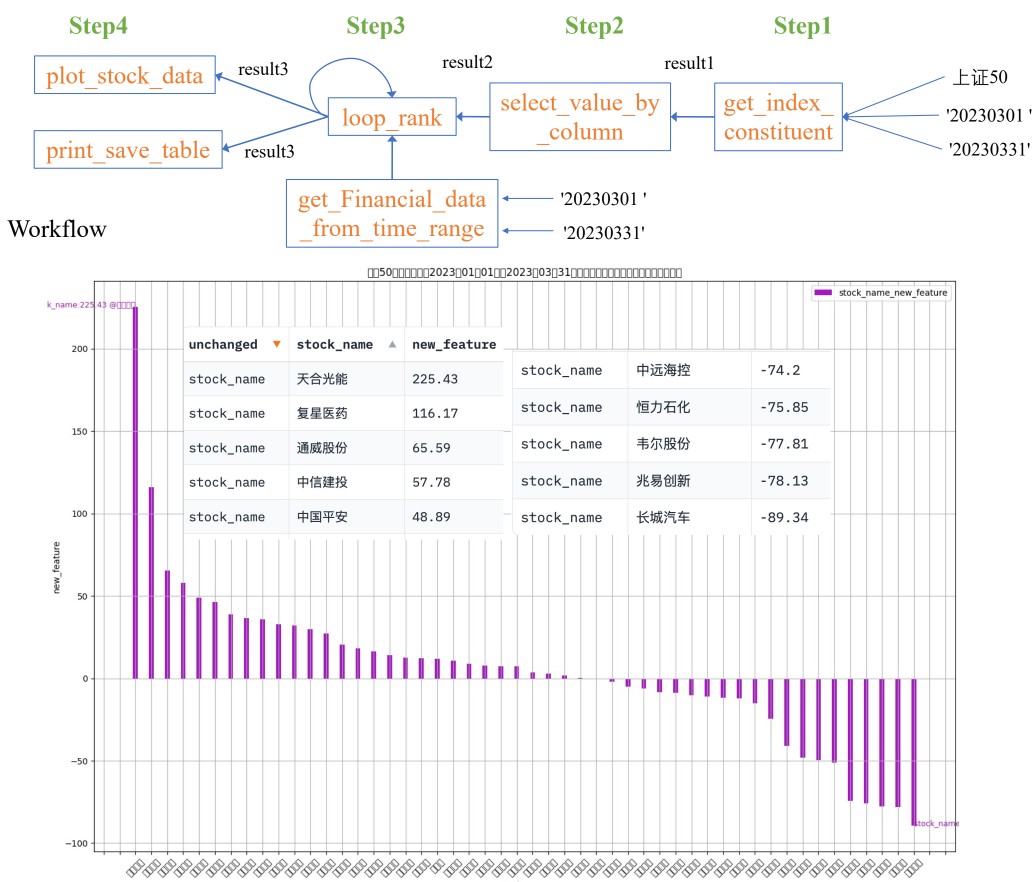


Figure G4: For complex requests about stock financial data, Data-Copilot deploys a loop workflow to solve user requests and finally outputs images and tables in parallel.

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.

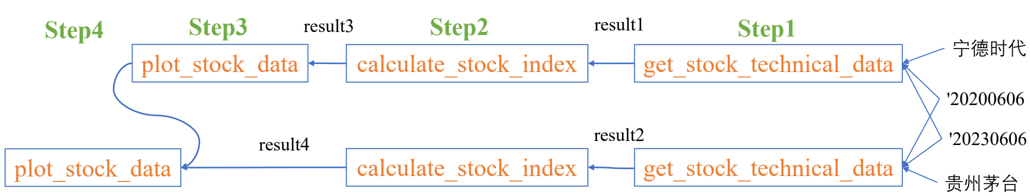
WorkFlow Planning :

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

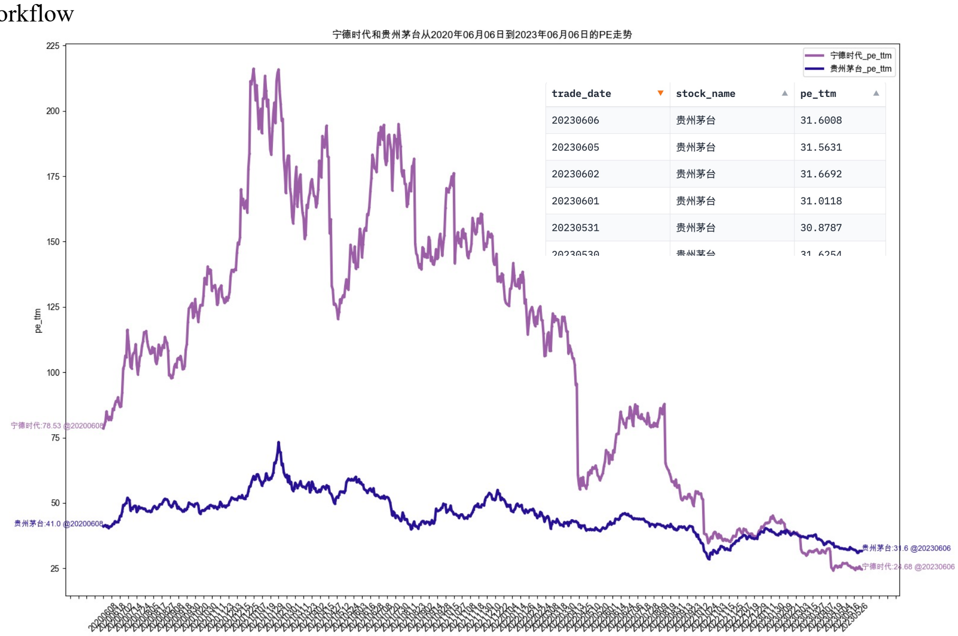
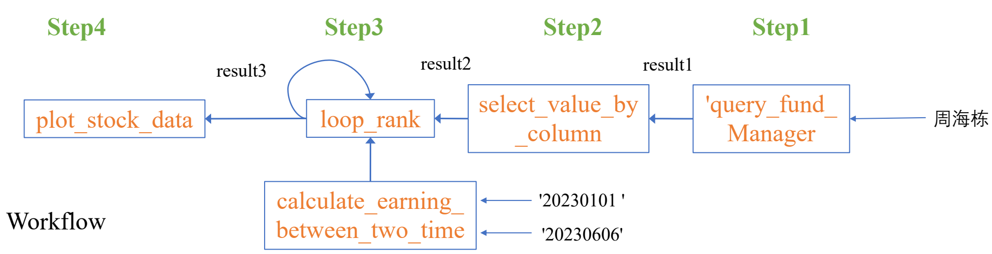


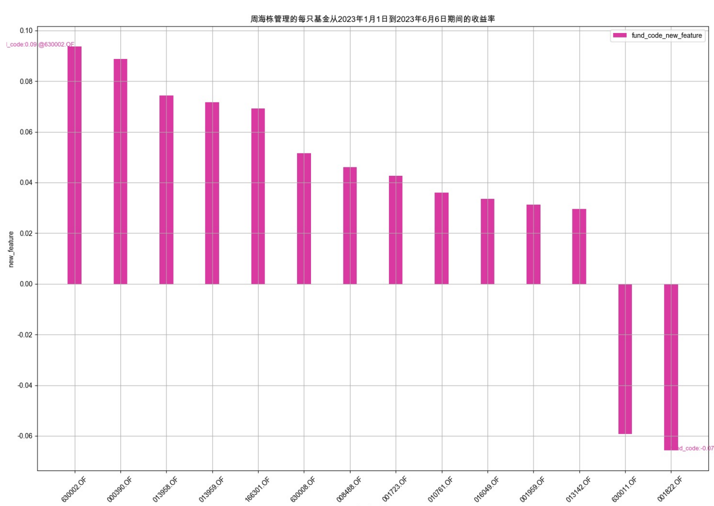
Figure G5: Data-Copilot acquires the P/E data of two stocks in parallel and plots them in a graph.

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 '}





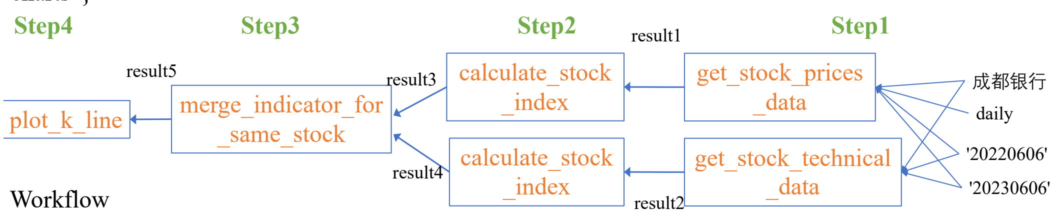
unchanged	fund_code	new_feature
fund_code	630002.OF	0.0937
fund_code	000390.OF	0.0888
fund_code	013958.OF	0.0744
fund_code	013959.OF	0.0716
fund_code	166301.OF	0.0692
fund_code	630008.OF	0.0516
fund_code	008488.OF	0.046
fund_code	001723.OF	0.0426
fund_code	010761.OF	0.036
fund_code	016049.OF	0.0335
fund_code	001959.OF	0.0312
fund_code	013142.OF	0.0296
fund_code	630011.OF	-0.0592
fund_code	001822.OF	-0.0657

Figure G6: 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 '}



Workflow

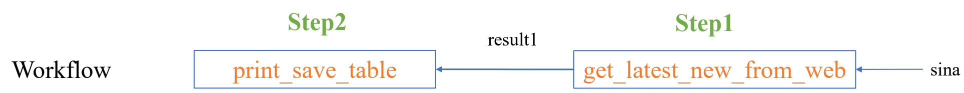
Open	High	Low	Close	Volume	stock_name	kdj_k	kdj_d	kdj_j
17.817800000000002	18.38473	17.52855	18.35002	235693.55	成都银行	63.08	49.25	90.738
18.31531	19.391320000000004	18.176470000000002	19.206200000000003	337528.97	成都银行	72.623	57.041	103.787
19.0905	19.54173	19.03265	19.36818	186443.96	成都银行	79.061	64.381	108.42
19.391320000000004	19.79627	19.37975	19.70371	213204.41	成都银行	84.758	71.173	111.928
19.599580000000003	20.1318	19.5533	19.98139	258296.99	成都银行	88.01	76.786	110.46
19.84255	20.22436	19.79627	20.06238	248768.19	成都银行	90.102	81.225	107.857
20.0161	20.363200000000003	19.7847	20.21279	262268.54	成都银行	91.633	84.694	105.51
20.4789	20.4789	19.9004	20.06238	272249.43	成都银行	89.716	86.368	96.412

Figure G7: 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 '}

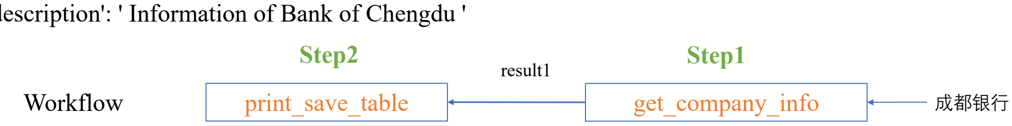


datetime	content
[datetime]: 2023-06-07 04:04:14	[content]: 道琼斯指数6月6日（周二）收盘上涨10.42点，涨幅0.03%，报33573.28点； 标普500指数6月6日（周二）收盘上涨9.14点，涨幅0.21%，报4282.93点； 纳斯达克综合指数6月
[datetime]: 2023-06-07 04:02:19	[content]: 美国财政部现金余额在周一（6月5日）大幅增加至712亿美元，上周五（6月2日）为234亿美元。单日增幅为479亿美元，创4月18日以来新高。
[datetime]: 2023-06-07 04:01:34	[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 G8: Data-Copilot can provide the latest financial news and company information by deploying the corresponding workflows.

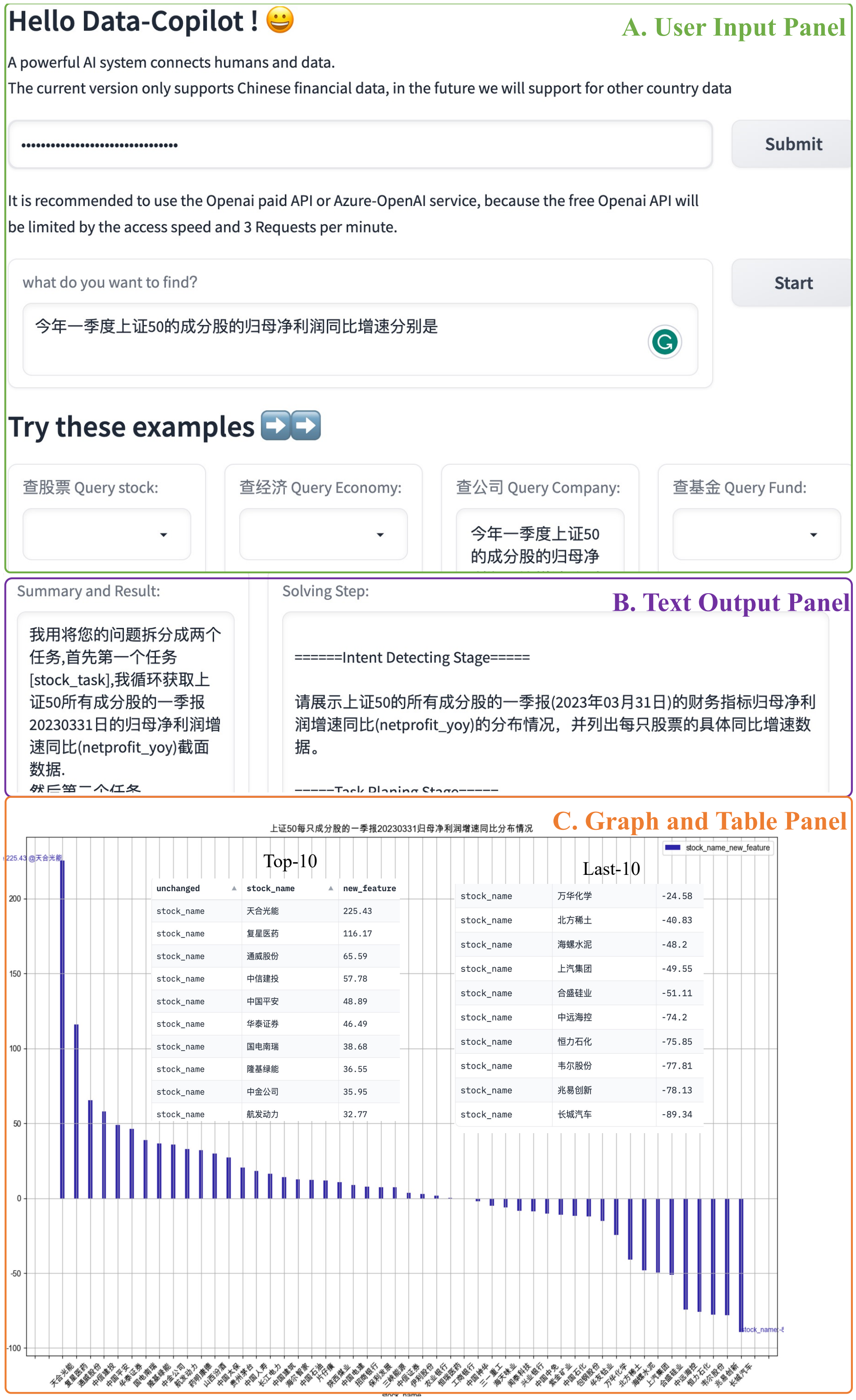


Figure G9: The user interface of our system. The green box (A) is the user input panel, and the purple (B) and red parts (C) are the results returned by the system.