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

Wenqi Zhang[1], **Yongliang Shen**[1], **Weiming Lu**[1], **Yueting Zhuang**[1]
Zhejiang University[1]
{zhangwenqi, syl, luwm, yzhuang}@zju.edu.cn

Abstract

Various industries such as finance, meteorology, and energy generate 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 that large language models (LLMs) have showcased promising capabilities in semantic understanding and reasoning, we advocate that the deployment of LLMs could autonomously manage and process massive amounts of data while displaying and interacting in a human-friendly manner. Based on this belief, 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 like an experienced expert, Data-Copilot autonomously transforms raw data into visualization results that best match the user's intent. Specifically, Data-Copilot autonomously designs versatile interfaces (tools) for data management, processing, prediction, and visualization. In real-time response, it automatically deploys a concise workflow by invoking corresponding interfaces step by step for the user's request. The interface design and deployment processes are fully controlled by Data-Copilot itself, without human assistance. Besides, we create a Data-Copilot demo that links abundant data from different domains (stock, fund, company, economics, and live news) and accurately respond to diverse requests, serving as a reliable AI assistant. Our project and demo are available at https://github.com/zwq2018/Data-Copilot.

1 Introduction

In the data-driven 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) [1, 2, 3, 4, 5], particularly the emergence of ChatGPT [6] and GPT-4 [7], has revolutionized AI research and paved the way for advanced AI systems. Leveraging chain-of-thought prompting [8, 9, 10, 11], reinforcement learning from human feedback (RLHF) [12, 13], and instruction-following learning [14, 15, 16], LLMs have demonstrated remarkable abilities in dialogue, reasoning, and generation. 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.

Many works have explored the potential of LLMs in data-related tasks. For instance, LiDA [17] and GPT4-Analyst [18] focus on visualization and data analysis. Beyond that, other works like Sheet-Copilot [19], Visual ChatGPT [20], Audio GPT [21] employ LLMs to evoke domain tools to analyze, edit and transform data. From the perspective of data science, table, visual and audio can all be considered as a form of data, and all these tasks can be viewed as data-related tasks: feeding

Preprint. Under review.

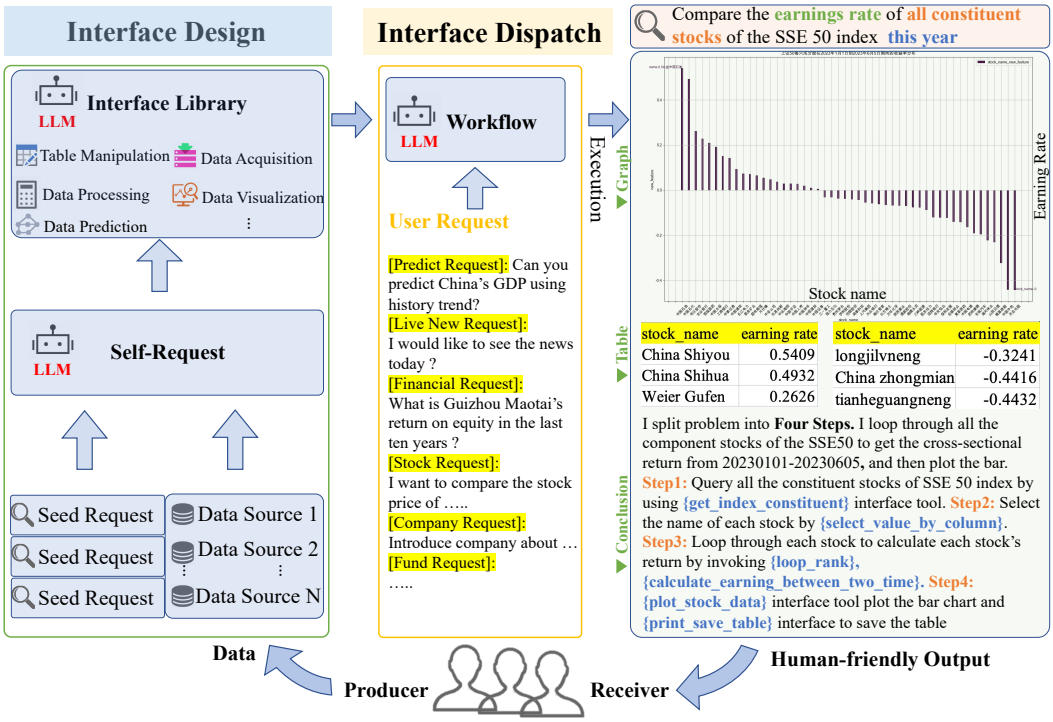


Figure 1: *Data-Copilot is an LLM-based system for data-related tasks, bridging billions of data and diverse user requests*. It independently designs interface tools for the efficient management, invocation, processing, and visualization of data. Upon receiving a complex request, Data-Copilot autonomously invokes these self-design interfaces to construct a workflow to fulfill human intent. Without human assistance, it adeptly transforms raw data from heterogeneous sources, in different formats, into a human-friendly output such as graphics, tables, and text.

data in a certain modality, processing it according to human instructions, and ultimately displaying the results. Therefore, one question arises: *Can LLMs, in the context of generic data, construct automated data science workflows capable of addressing a wide range of data-related tasks?*

To achieve this goal, 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: LLMs are not adept at handling numerical computations and may not have suitable callable external tools to meet diverse user requests, thus limiting the utilization of LLMs. (3) From the task perspective: although LLMs have demonstrated strong few-shot capabilities, many data-related tasks are intricate, requiring a combination of many operations, like data retrieval, computations, and table manipulations, and the results need to be presented in multiple formats including images, tables, and text, all of which are beyond the current capabilities of LLMs. Hence, it is challenging to directly apply the current methods for data-related tasks.

To pierce through the fog and find a way, 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 [22]:

> *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.*

This insight inspires us that how effectively we extract information from data depends on the kinds of tools we have at our disposal. Therefore, we advocate that LLM should not handle data directly, but rather act as a brain, creating appropriate interface tools to manage and utilize data, and presenting valuable information in a human-centric manner. Based on this, we propose a system called **Data-Copilot**, which harnesses the capabilities of LLM for creating suitable interfaces and deploying

autonomous workflow for humans. As shown in Figrue 1, to handle data-related tasks with large volumes of data, rich data sources, and complex query intent, Data-Copilot can autonomously design versatile interface tools for **data management, invocation, processing, forecasting, and visualization**, and dispatch these interface tools step by step, forming a data-to-human workflow based on user requests.

More than just a visualization tool, Data-Copilot is a versatile framework that connects numerous data sources from different domains on one end and caters to diverse user demands on the other end. It can **continuously enrich its interface tools with just a small number of seed requests**, thereby expanding its range of capabilities, such as advanced data analysis and more complex data forecasting. To achieve this, it comprises two processes: the Interface Design and the Interface Dispatch.

- **Interface Design:** Data-Copilot adopts an iterative self-request process to fully explore the data and cover most scenarios. As shown in Figure 1, Data-Copilot is instructed to generate a large number of diverse requests from a few seed requests, then abstracts self-generated requests into interface tools and merges interfaces with similar functionalities. Finally, it harvests a handful of versatile interfaces, encompassing data acquisition, processing, forecasting, table manipulation, and visualization.
- **Interface Dispatch:** When a user request is received, Data-Copilot first parses the user intention and then plans an interface invocation process after reviewing the interface description designed by itself. It is capable of flexibly constructing workflows with various structures (including sequential, parallel, and loop structures) to address user requests.

Incorporating two phases, Data-Copilot successfully manages and analyzes large amounts of data via its self-designed interfaces. It bypasses the need for direct data reading and the insufficiency of external tools. Besides, it is also easily extended for emerging requests and data simply by adding a new interface, demonstrating good scalability.

Overall, our contributions can be summarized as follows:

1. To efficiently handle data-intensive tasks on a large scale, we design a universal system, **Data-Copilot**, that connects data sources from different domains and diverse user tastes, by integrating LLM into every stage of the pipeline to reduce tedious labor and expertise.
2. Data-Copilot can autonomously manage, process, analyze, predict, and visualize data. When a request is received, it transforms raw data into informative results that best match the user's intent.
3. Acting as a **designer** and **dispatcher**, Data-Copilot encompasses two phases: an offline interface design and an online interface dispatch. Through self-request and iterative refinement, Data-Copilot designs versatile interface tools with different functions. In the interface dispatch, it invokes the corresponding interfaces sequentially or in parallel for accurate responses.
4. We built a Data-Copilot demo for the Chinese financial market. It can access the stock, funds, economic, financial data, and live news, and provides diverse visualizations: graph, table, and text descriptions, customized to the user's request.

2 Related Works

In the recent past, breakthroughs in large language models (LLMs) such as GPT-3, GPT-4, PaLM, and LLaMa [1, 2, 3, 4, 5, 12, 23, 24, 25] 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 [12, 13, 26].

Besides, recent studies have begun to explore the synergy between external tools and large language models (LLMs). Tool-enhanced studies [27, 28, 29, 30] 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 [29, 20, 31, 32, 33, 21]. In addition, there are many

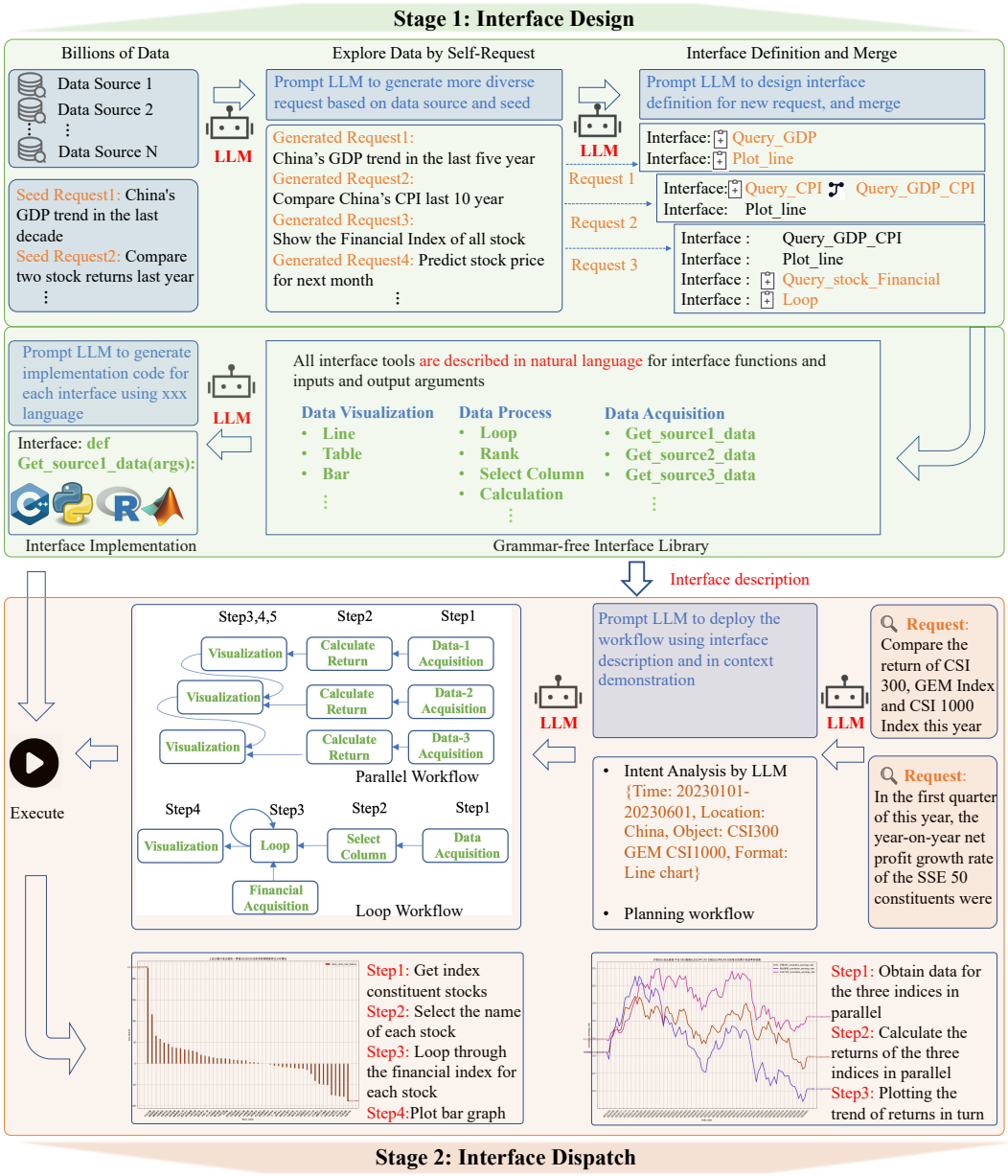


Figure 2: Overview of Data-Copilot. **Interface Design:** We devise a self-request process allowing LLM to generate sufficient requests from a few seed requests autonomously. Then LLM iteratively designs and optimizes interfaces based on generated requests. These interfaces are described in natural language, 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.

excellent applications, such as AutoGPT [1], AgentGPT [2], BabyAGI [3], BMTools [4], LangChain [5] and etc. Most of them are focused on daily tools and do not consider the specificity of data-related tasks.

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

Except for learning to operate the tools, several contemporaneous studies [34, 35] 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.

Distinct from these approaches, our Data-Copilot system is different in the following aspects: (1) Data-Copilot is a general LLM-based system specifically designed for a variety of data-related tasks. It contemplates how LLM can be exploited to access and manage large amounts of heterogeneous data for complex user demands, like querying, analysis, and visualization. (2) Requiring only a few seed requests, Data-Copilot employs a self-request approach to design its interface tools independently. It separates the definition of an interface from its specific implementation, allowing LLM to focus more on accurately describing the functionality of the interface. This approach is a clear difference from previous works and provides the community with a viable solution for automated tool creation. (3) Data-Copilot is capable of constructing more complex interface scheduling processes such as parallel, sequential, and loop workflow based on its own designed interfaces.

3 Data-Copilot

Data-Copilot is an LLM-based system that accomplishes automated management, invocation, and processing of a large number of data from different sources by self-design sophisticated interface tools. As Peter Naur says, humans are the source and receiver of data, and Data-Copilot, serving as a dependable AI assistant, independently manages, processes, and analyzes data, autonomously extracting the most valuable information from the data and presenting it to humans in a friendly manner, significantly reducing the need for tedious labor.

As shown in Figure 2, firstly the LLM plays the role of a designer (Interface Design in Section 3.1) to autonomously construct every component of the workflow. Besides, LLM also acts as a dispatcher (Interface Dispatch in Section 3.2) to deploy the workflow automatically using generated interfaces. Through the integration of these two stages, Data-Copilot manages to efficiently handle a large volume of data-related tasks and caters to various user needs.

3.1 Interface Design

Just as Peter Naur pointed out, designing a set of versatile interface tools for managing, invoking, processing, and visualizing is essential. Therefore, Data-Copilot designs a plethora of interfaces as tools for data management, where the interface is a module composed of natural language (functional description) and code (implementation), responsible for data acquisition, processing, and others.

As shown in Figure 2, Data-Copilot maintains a library of interface tools that store the currently generated interfaces. First, LLM is provided with some seed requests and autonomously generates a large number of requests. Then LLM designs the corresponding interface (only description and arguments) for addressing these requests and gradually optimizes the interface design at each iteration. Finally, we utilize the LLM's powerful code generation ability to generate specific codes for each interface in the interface library. This process detaches the design of the interfaces from the specific implementation, creating a set of versatile interface tools that can cover most requests.

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 [26, 17], Data-Copilot first autonomously explores the data to mine more requests. Then, based on these requests and data, it proceeds to design the interfaces.

Specifically, we generate a parsing file for each data source to help LLM understand the data. Each parsing file includes a description of the data, the name of each column (attribute), an access example, and the first and last rows of its output. This process does not require too much labor, as data providers usually offer a description of the data and methods of access.

Then we adopt a self-request process to explore data, i.e. these parsing files and a few seed requests are fed into the LLM as prompt and the LLM is instructed to generate more diverse requests. These generated requests are used for the next step (Interface Definition). Allowing LLMs to autonomously explore the data and generate requests by themselves is crucial since it ensures that the interface

design in the subsequent step is adequate and versatile. In Figure 2, LLM generates four requests based on the two seed requests and all data sources.

Interface Definition In this step, Data-Copilot defines 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 Data-Copilot as a prompt. As shown in Figure 2, each request is fed into Data-Copilot one by one, and Data-Copilot is prompted to use existing interfaces from the interface library or re-define a new interface to fulfill the current request. The detailed prompt is shown in Figure 10. LLM is prompted to generate the interface library at each iteration in `JSON` format: `Interface1={Interface Name:"", Function description:"", Input and Output:""}, Interface2={Interface Name: "", Function description:"", Input and Output:""}`, and to generate the solution using these interfaces `Solution=" You would first get the data using getBankData, then..."`. Importantly, Data-Copilot only uses natural language to define the functions of interfaces and their arguments, without considering the specific implementation, which is grammar-free. This process is similar to software architecture design, but automatically achieved by Data-Copilot. It allows Data-Copilot to focus more 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 Data-Copilot designs a new interface for a new request, it 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 for `{Interface: Query-GDP}` and `{Interface: Query-CPI}` have been merged into one `{Interface: Query-GDP-CPI}`. This process is similar to software developers wrapping similar modules into a new one in their development. The detailed prompt can be found in Figure 10.

Through this process, a large number of similar interfaces are merged. Each interface in the library has a very clear function and is significantly different from the others, which is beneficial for deploying a clear and concise workflow in real-time response.

Interface Implementation Each request is successively fed into Data-Copilot for interface definition and interface merging. Eventually, when all the requests can be satisfied by the interfaces in the library, Data-Copilot leverages the powerful code-generation capabilities of LLM to generate implementation code for each interface.

The whole interface design process is offline. As shown in Figure 4, 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 to accommodate additional requests or new data sources. Besides, Data-Copilot also easily switches to other programming platforms and databases by simply re-generating the implementation code for the interface, demonstrating excellent scalability.

3.2 Interface Dispatch

In the previous stage, we harvest a variety of generic interface tools for data acquisition, processing, and visualization. Each interface has a clear and explicit function description. As the two examples shown in Figure 2, Data-Copilot forms a workflow from data to multi-form results by planning and invoking different interfaces in real-time requests.

Intent Analysis To accurately comprehend user requests, Data-Copilot first analyzes user intent. The intent analysis involves parsing the time, location, data object, and output format, which are all critical to data-related tasks. For example, if the user's question is: `"I want to compare the GDP and CPI trend in our area over the past five years"`, Data-Copilot, after accurately understanding the semantics, parses it as: `"Draw a line chart of China's national GDP and CPI per quarter from May 2017 to May 2023 for comparison" (Time: 201705-202305, Location: China, Object: China's quarterly GDP and CPI, Format: Line Chart)`. 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. As shown in Figure 11, we provide a detailed prompt design to instruct the LLM for this stage.

Planing Workflow Once the user's intent is accurately understood, Data-Copilot plans a reasonable workflow to process the user's request. We instruct LLM to generate a fixed format `JSON` representing each step of the scheduling, like `step={"arg":"", "function":"", "output":"" ,"description":""}` (detailed prompt design shown in Figure 11). Except for the interface description generated in the design phase, Data-Copilot also incorporates several demonstrations as part of the prompt for in-context learning. As shown in Figure 11, each demonstration includes a task instruction (`###Instruction:`) and the corresponding interface invocation procedure (`###Function Call:`), which improves Data-Copilot's understanding of the logical relations between the different interfaces.

Prompted by interface description and several demonstrations, Data-Copilot meticulously orchestrates the scheduling of the interface in either a sequential or parallel manner within each step. The LLM autonomously determines which interfaces should be invoked, and in what order, based on the user's request and interface definition (Section 3.1). 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 step, it dispatches `{Data Acquisition Interface}` **in parallel** to obtain the data of the three indices, and the second step is the same for the returns of the three indices. The last three steps successively invoke the `{Visualization Interface}` to sequentially plot the trends of the three indices on the same canvas. In the second case, Data-Copilot deploys **a loop workflow** that accomplishes the calculation of financial indicators for all 50 constituent stocks by calling the `{Loop and Rank Interface}`.

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, Data-Copilot 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 for data-related tasks.

In Figure 3, we provide a detailed example where the user inputs the request: `"Forecasting China's GDP growth rate for the next 4 quarters."`. Data-Copilot first interprets the user's intent based on local time. Subsequently, it deploys a three-step workflow: The first step involves invoking the `{get-GDP-data}` interface to acquire historical GDP data. The second step involves invoking the `{predict-next-value}` interface for forecasting. The final step is visualizing the output.

4 Experiments

4.1 Settings

We have built a Data-Copilot demo using Chinese financial data and the Gradio library, which can access stocks, funds, economic data, real-time news, company financial data, and more. This allows for real-time data query, computation, analysis, and a variety of visualizations. In our experiments, we employe `gpt-4, gpt-3.5-turbo` as LLMs through the OpenAI API [6] and adopted `Tushare` [7] as our data source. In the first phase, Data-Copilot uses the `gpt-4` for interface design and `gpt-3.5-turbo` for the second interface dispatch stages, and the Python code was generated for each interface in the interface implementation phase. We filter the interfaces obtained in the first phase, retaining those that can be run without errors. Besides, to improve planning efficiency, we carry out hierarchical planning in interface dispatch stages: 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 (refer to Figure 11 for more detail). To make the output results more stable, we set the temperature coefficient to 0.

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

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

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

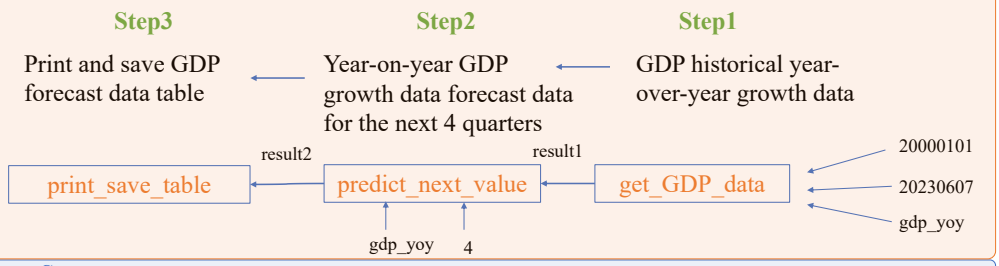
Intent Analysis: Show data and print tables for the next 4 quarters based on China's GDP growth rate for each quarter from 20000101 to June 07, 2023 (today)

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

4.2 Qualitative Results

Our system is shown in Figure 12, where the user proposes a request, Data-Copilot dynamically displays the intermediate process (`Solving Step`) and finally displays the bar chart, table, and summary (`Summary and Result`). Due to limitations on data sources and the length of LLMs input tokens, the current version of Data-Copilot only includes data from the Chinese financial market. In the future, Data-Copilot will support financial market data from more countries and massive data from other industries.

In the first stage, Data-Copilot designs many general-purpose interface tools through a self-request approach. As shown in Figure 4, we list some of these interface tools, which include data acquisition, data processing, DataFrame manipulation, and visualization interfaces. Thanks to the diverse requests generated by self-requests and interface merging mechanisms, each interface definition is very clear and functions are quite versatile.

For instance, the `get-stock-prices-data` interface integrates the functions of acquiring data in daily, weekly, and monthly frequencies, and automatically adds an argument (`freq`) in the input, which is very similar to the process of programmer development.

Aside from combining similar simple interfaces, we observed that Data-Copilot also designed some complex interfaces, such as `Loop-Rank`. This interface accepts a function and a set of variables as arguments, and calls the function for each variable in sequence, realizing a looping process. We also noticed that Data-Copilot designed `Loop-Rank` aim to deal with complex requests like the second case in Figure 2: `"What is the year-on-year net profit growth rate of the SSE`

`50 constituents in the first quarter of this year?"`. After designing the `Loop-Rank` interface, such complex requests can be solved with a concise workflow.

In the second phase, Data-Copilot deploys a workflow using the self-design interfaces. The planning workflow usually consists of multiple steps, each of which invokes an interface, or multiple interfaces in parallel. For example, in the first case in Figure 2 `"Compare the return of CSI 300, GEM Index and CSI 1000 Index this year"`, the first two steps invoke the same interface three times to calculate the earning return for the three indices. We show more cases in the case study.

4.3 Case study

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 5,6, Data-Copilot deploys different structural workflows based on user requirements. In Figure 5, 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 6, 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 7,8,9 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.

5 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 will greatly broaden the application scenarios of Data-Copilot.

2) **System stability** The interface deployment process can occasionally be unstable. The main source of this instability is due to the fact that 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.

6 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. Data-Copilot, by incorporating LLMs into each stage of data-related tasks, autonomously transforms raw data into user-friendly visualization results based on user requests, significantly 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 the interfaces in real-time response, deploying a clear workflow for user requests. Both processes, interface design, and dispatch, are completely controlled by Data-Copilot, with minimal human assistance required. With access to Chinese financial data, Data-Copilot can flexibly handle challenging requests about stock, fund, economics, company finance, and live news, serving as a reliable AI assistant for humans.

Interface design in First Stage by LLM

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 4: 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 :

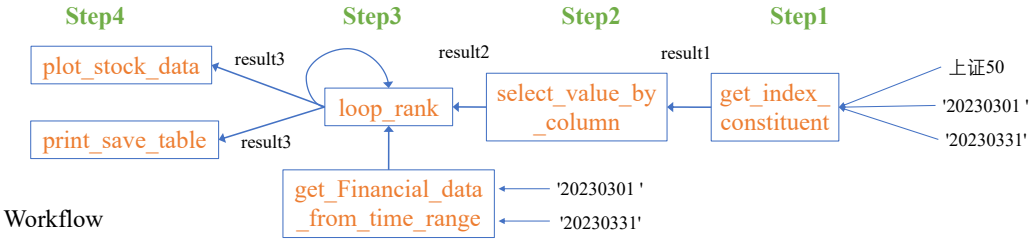
step1=: {'arg1': ['上证50', '20230301', '20230331'], 'function1': 'get_index_constituent', 'output1': 'result1', 'description1': ' Information on all SSE 50 constituents for the month '}

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'}



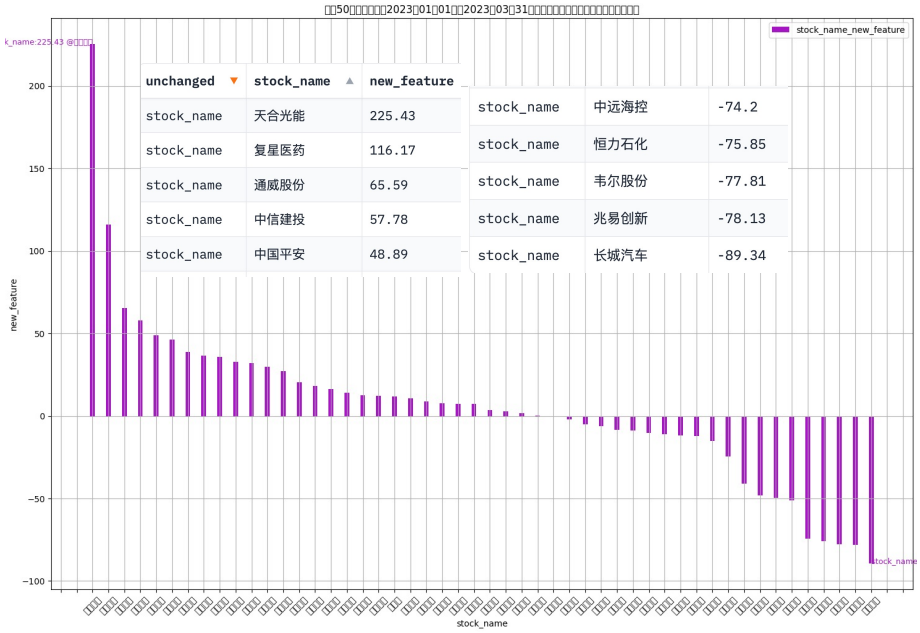


Figure 5: For complex requests about stock financial data, Data-Copilot deploys a loop workflow to solve user requests and finally outputs images and tables in parallel.

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.

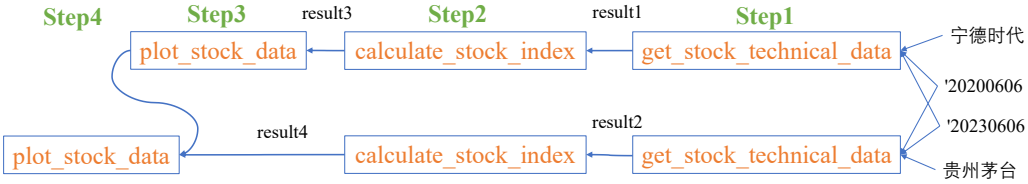
WorkFlow Planning :

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

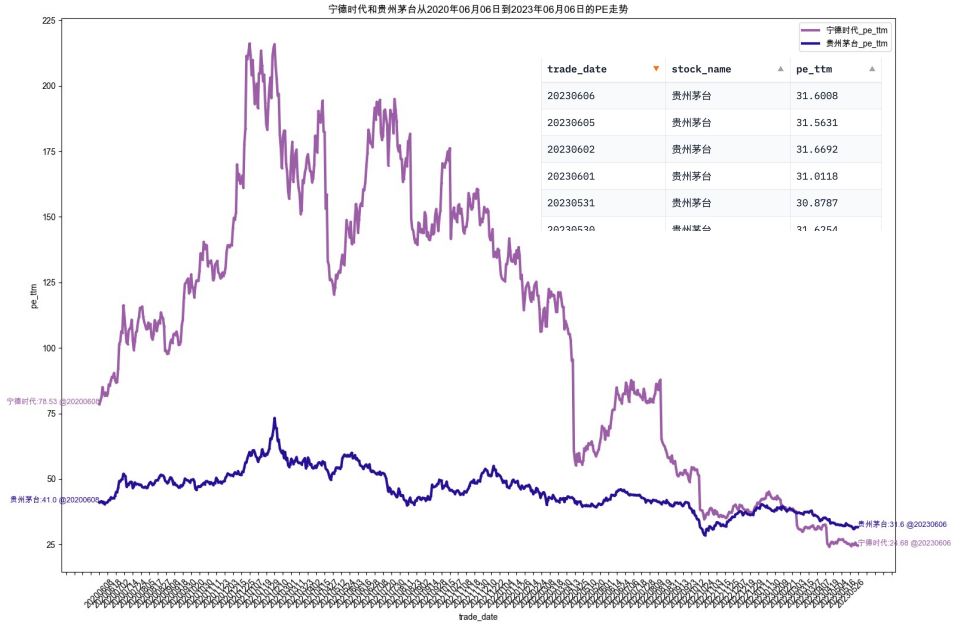
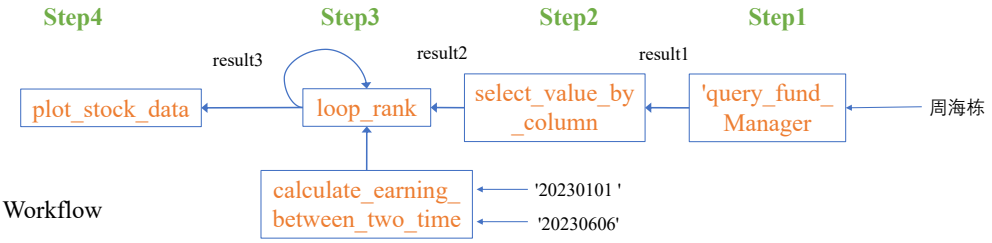


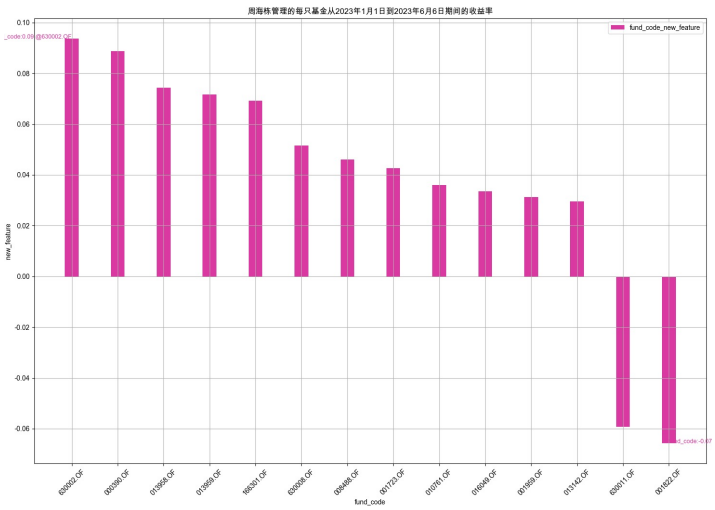
Figure 6: Data-Copilot acquires the P/E data of two stocks in parallel and plots them in a graph.

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 '}





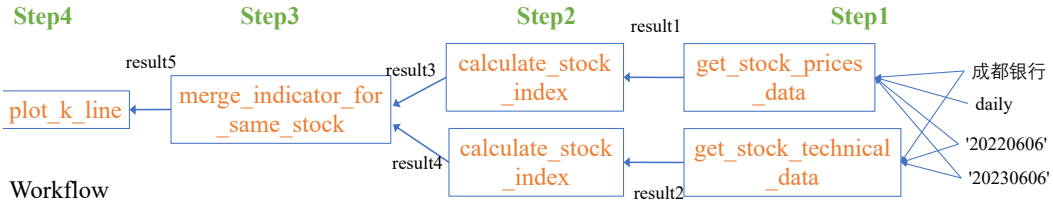
unchanged	fund_code	new_feature
fund_code	630002.OF	0.0937
fund_code	000390.OF	0.0888
fund_code	013958.OF	0.0744
fund_code	013959.OF	0.0716
fund_code	166301.OF	0.0692
fund_code	630008.OF	0.0516
fund_code	008488.OF	0.046
fund_code	001723.OF	0.0426
fund_code	010761.OF	0.036
fund_code	016049.OF	0.0335
fund_code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

Figure 7: 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 '}



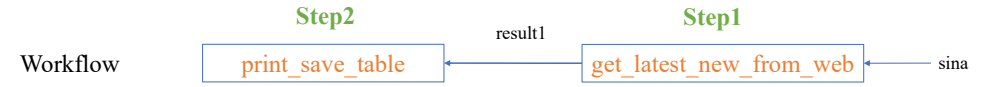
Workflow

Figure 8: 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 '}

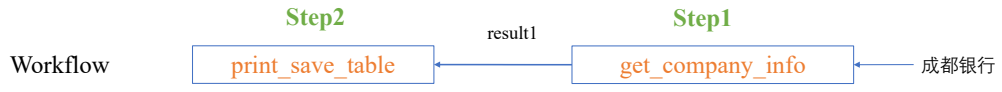


datetime	content
[datetime]: 2023-06-07 04:04:14	[content]: 道琼斯指数6月6日（周二）收盘上涨10.42点，涨幅0.03%，报33573.28点； 标普500指数6月6日（周二）收盘上涨9.14点，涨幅0.21%，报4282.93点； 纳斯达克综合指数6月
[datetime]: 2023-06-07 04:02:19	[content]: 美国财政部现金余额在周一（6月5日）大幅增加至712亿美元，上周五（6月2日）为234亿美元。单日增幅为479亿美元，创4月18日以来新高。
[datetime]: 2023-06-07 04:01:34	[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 9: Data-Copilot can provide the latest financial news and company information by deploying the corresponding workflows.

A Case for Interface design

Interface Definition:
Design_prompt: You are an experienced program designer, I propose a request and you can use the interface in interface library to solve the request or design new interface functions (similar to pseudocode) to implement my request. Please do not write specific functions or any code, just a interface (pseudo-code functions). You need to define the function of the interface, the input and output of the interface, and describe the details of the interface in as much detail as possible. Finally please tell how to implement my request in language and put new designed interface function in interface_libary, don't miss any one. Please answer in following format:
Interface function Library**:** Interface1={Interface Name: %s, Function description:%s, Input and Output:%s}, Interface2={Interface Name: %s, Function description:%s, Input and Output:%s},..}. The solution using these interface function: solution=[%s]

Iteration 1
Request: Get the daily opening price data of Bank of Hangzhou from Jan 23, 2018 to Mar 13, 2019 and plot the trend. Please do not write the code, just design the pseudo code function in function_libary

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.

Interface function Library
Interface 1={Interface Name: getBankData….}
Interface 2={Interface Name: prepareDataForPlotting….}
Interface 3={Interface Name: plotData, ...}
Solution=[These three interface functions will meet your needs, and they are all stored in the interface_library. 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…..]

Iteration 2--New design

Request: Get the monthly closing price data of Ningde Times from January 23, 2018 to March 13, 2019, and plot the trend

Response:
To meet this demand, we can make a slight modification in the existing function_library...
Interface 1={Interface Name: getBankData...}
Interface 2={Interface Name: prepareDataForPlotting...}
Interface 3={Interface Name: plotData….}
Interface 4={Interface Name: getStockData….}
Merge Solution=[You need to call these functions in order to meet your needs. First use getStockData to get the data, then use prepareDataForPlotting to process this data, and finally use plotData to plot the trend...]

Merge_prompt: Please check if the functions in the interface library you designed can be merged. Especially for newly designed interfaces in the previous step, ,merging interfaces with similar functions and similar input and output formats into a new interface. You can use argument to control different inputs and outputs. Please export the merged interface library as following:
Interface function Library**:** Interface1={Interface Name: %s, Function description:%s, Input and Output:%s}, Interface2={Interface Name: %s, Function description:%s, Input and Output:%s},..}.
The merge solution: Merge Solution=[%s]

Iteration 2--Merge
Merge Solution=[Yes, function_library can be merged and reformulated to eliminate duplicate functions. We can merge getBankData and getStockData into a more generic function getFinancialData, 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]

Interface 1={Interface Name: getFinancialData...}
Interface 2={Interface Name: prepareDataForPlotting...}
Interface 3={Interface Name: plotData….}

Figure 10: A simple example of iterative interface design, including interface definition and interface merging.

Prompt for Interface dispatch

Intent Analysis:

Analysis prompt: Please parse the input instruction for time, place, object and output format and rewrite the instruction according to today's date, the rewritten new instruction must be semantically consistent and contain a specific time and specific indicators. Use the following format to output{Time:%s, Location :%s, Object :%s, Format :%s}.
Demonstration1:
Instruction: Today's date is 2019-03-13,please help me plot the stock price trend of Guizhou Maotai from January 23, 2018 to today.
###New Instruction: 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)

Task Selection

Select_prompt: Please select the most suitable task according to the given instruction 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. Use the following format to output task1={%s: %s},task2={%s: %s}.

Demonstration1:
###Instruction: Please help me plot the closing price of Guizhou Maotai from Jan 23, 2018 to Mar 13, 2019
###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 \"}

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

Demonstration1:
###Instruction: Get the time series data of Guizhou Maotai's daily closing price from Jan 23, 2018 to Mar 13, 2019
####Function Call: step1={\n \"arg1\": [\"Guizhou Maotai\",\"20180123\",\"20190313\",\" daily\"],\n \"function1\": \"get_stock_prices_data\",\n \"output1\": \"result1\",\n \"description1\": \"Guizhou Maotai daily stock price (open, close, high, low, etc.) time series data\",\n \"\n}, step2={\n \"arg1\": [\"result1\",\"close\"],\n \"function1\": \"calculate_stock_index\",\n \"output1\": \"result2\",\n \" description1\": \"Guizhou Maotai's closing price time series data\"\n}, step3={\n \"arg\": [\"result2\", null, \"line\", \" Guizhou Maotai closing price chart from January 23, 2018 to March 13, 2019 \"],\n \"function\": \"plot_stock_data\",\n \"output\": \"result1\",\n \"description\": \" Guizhou Maotai Stock Price Line Chart \"\n} ###

As Json Format:

```
step1 = {
"arg1": ["Guizhou Maotai", "20
"function1": "get_stock_prices
"output1": "result1",
"description1": "Guizhou Mao
low, etc.) time series data"
}
```

```
step2 = {
"arg1": ["result1", "close"],
"function1": "calculate_stock_inc
"output1": "result2",
"description1": "Guizhou Maotai
}
```

```
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"
}
```

Figure 11: Prompt and Demonstration design for interface dispatch stage.

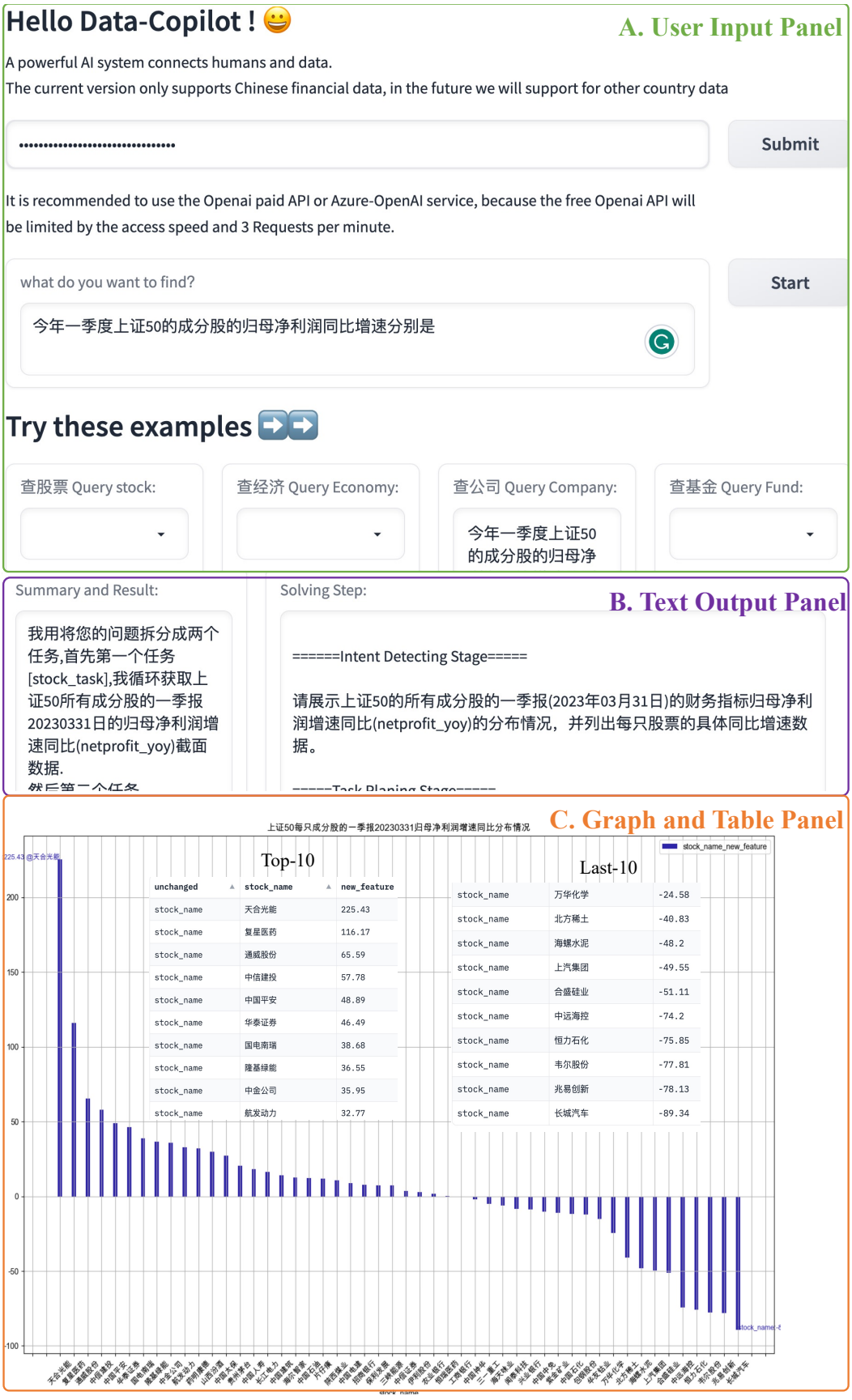


Figure 12: The user interface of our system. The green box (A) is the user input panel, and the purple (B) and red parts (C) are the results returned by the system.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] OpenAI. Chatgpt. 2022.

[7] OpenAI. Gpt-4 technical report. 2023.

[8] 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)*, 2022.

[9] 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.

[10] 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, 2022.

[11] 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, 2023.

[12] 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.

[13] 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, 2022.

[14] 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*, 2022.

[15] 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.

[16] 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.

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

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

[19] 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*, 2023.

[20] 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*, 2023.

[21] 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.

[22] Peter Naur. Concise survey of computer methods. 1974.

[23] OpenAI. Gpt-4 technical report, 2023.

[24] 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, 2022.

[25] 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*, 2023.

[26] 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, 2022.

[27] 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.

[28] 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, 2022.

[29] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models, 2023.

[30] 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.

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

[32] 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.

[33] 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, 2023.

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

[35] 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.