



ContextMate: a context-aware smart agent for efficient data analysis

Aamir Khan Jadoon¹ · Chun Yu¹ · Yuanchun Shi^{1,2}

Received: 17 September 2023 / Accepted: 26 December 2023
© China Computer Federation (CCF) 2024

Abstract

Pre-trained large language models (LLMs) have demonstrated extraordinary adaptability across varied tasks, notably in data analysis when supplemented with relevant contextual cues. However, supplying this context without compromising data privacy can prove complicated and time-consuming, occasionally impeding the model's output quality. To address this, we devised a novel system adept at discerning context from the multifaceted desktop environments commonplace amongst office workers. Our approach prioritizes real-time interaction with applications, according precedence to those recently engaged with and have sustained prolonged user interaction. Observing this landscape, the system identifies the dominant data analysis tools based on user engagement and intelligently aligns concise user queries with the data's inherent structure to determine the most appropriate tool. This meticulously sourced context, when combined with optimally chosen prefabricated prompts, empowers LLMs to generate code that mirrors user intentions. An evaluation with 18 participants, each using three popular data analysis tools in real-world office and R & D scenarios, benchmarked our approach against a conventional baseline. The results showcased an impressive 93.0% success rate of our system across seven distinct data-focused tasks. In conclusion, our method significantly improves user accessibility, satisfaction, and comprehension in data analytics.

Keywords UI context generation · Context-aware systems · Large language models · Data analysis · Data visualization · Natural language interfaces · Privacy preservation

1 Introduction

Context is any information that can be used to characterize and interpret the situation in which a user interacts with an application at a certain time (Bazire et al. 2005). Without learning the contextual information, queries such as “Plot sales of each year”, “Analyze and export the key findings” or “Which region generated the maximum revenue” are difficult for both machines and humans to comprehend. So, ‘context is key’ (Coutaz et al. 2005) and ‘understanding context is vital’ (Greenberg 2001) for obtaining expected

response from the system. Moreover, A system that possesses the ability to select and implement the most appropriate techniques and tools for a given context can be referred to as context-driven system (Bach and Bolton 2016).

In today's era of parallel processing, it is common for office workers to work on multiple tasks simultaneously, with tasks sometimes overlapping and making it difficult to determine their boundaries. Depending on the domain, office workers and researchers use various applications to analyze their data, and may not limit themselves to a single instance of the target application. Hence, further exacerbates the challenge of effectively analyzing multiple datasets across multiple instances of applications concurrently. However, with the advent of ChatGPT,¹ GPT-4 (OpenAI 2023), users can now utilize natural language queries to efficiently generate spreadsheet formulas (Joshi et al. 2023), SQL queries (Ni et al. 2023) and visualizations (Maddigan et al. 2023) from structural data and even fine-tune pre-generated visualizations (Maddigan et al. 2023) using these pre-trained LLMs.

The provision of detailed context in prompts aids pre-trained LLMs in generating accurate responses. However,

✉ Chun Yu
chunyu@tsinghua.edu.cn

Aamir Khan Jadoon
qinl19@mails.tsinghua.edu.cn

Yuanchun Shi
shiyc@tsinghua.edu.cn

¹ Department of Computer science and Technology,
Tsinghua University, 30 Shuangqing Rd, Haidian District,
Beijing 100084, China

² Qinghai University, Xining 810016, China

¹ <https://openai.com/blog/chatgpt/>

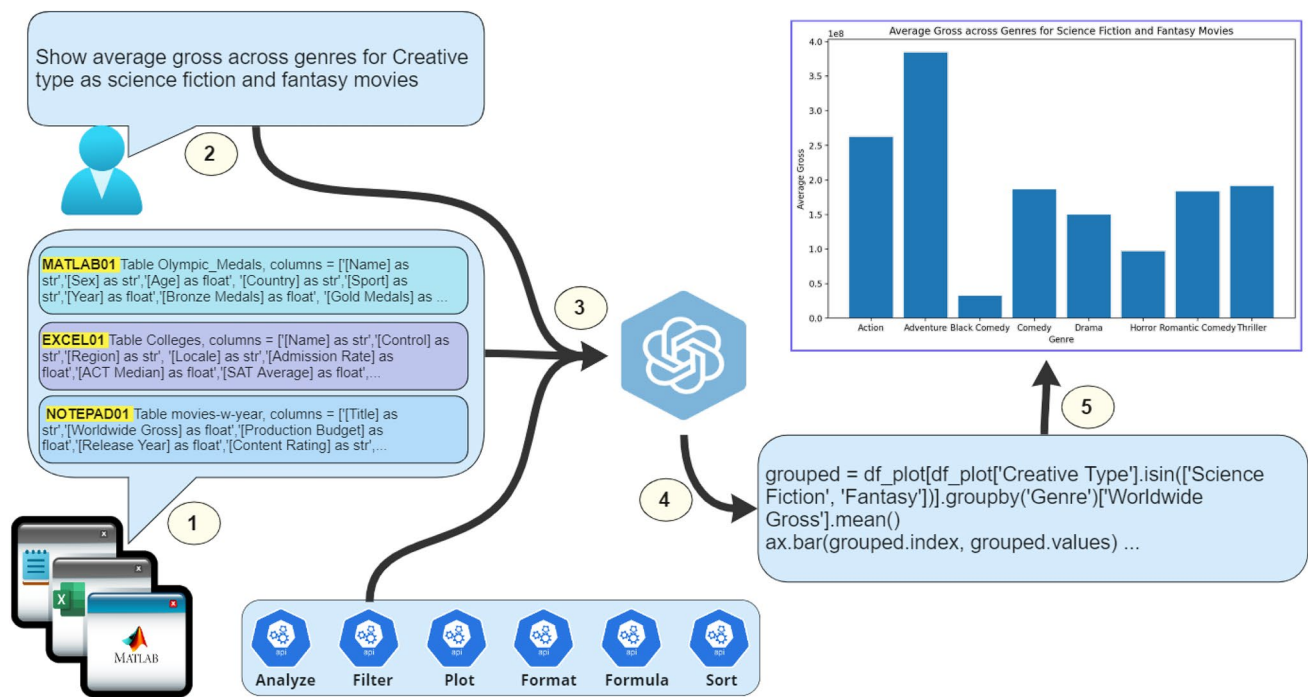


Fig. 1 Illustration of ContextMate's workflow. The user engages in data analysis tasks across multiple applications with different datasets (1). The user interacts with ContextMate through natural language query (2). ContextMate captures the user's interactions with the applications and integrates the contexts and available APIs into the

query, which is then passed to the LLM (3). The LLM contextualizes the user's query, selects the appropriate API, generates the task outline, and produces the desired results (4) in the intended application (5)

one significant challenge is automatically detecting the rapidly changing user's desktop context, as it enables better support for the user. In this research paper, we propose ContextMate: A context-aware smart agent for efficient data analysis that can improve users' performance in data analysis tasks by intelligently generating and seamlessly integrating contextual information into their concise natural language textual queries. We employed the pre-designed few-shot example prompts enriched with real-time context generated for data analysis tasks by utilizing the GPT-4 (OpenAI 2023). Furthermore, our system enhances its performance by exploiting reinforcement learning techniques that incorporate human feedback (Christiano et al. 2017; Ouyang et al. 2022).

We evaluate ContextMate, by leveraging commonly used software tools such as Microsoft Excel (Corporation 2019), MATLAB (The MathWorks 2022), and CSV files. These choices were driven by the popularity of these tools in both academic and industrial settings, accommodating users with varying levels of expertise from beginners to advanced practitioners. Each tool brings unique strengths and limitations to the table, collectively providing a comprehensive evaluation environment for our proposed system, ContextMate. For instance, Microsoft Excel is widely employed for its intuitive spreadsheet-based data analysis capabilities but may have

limitations in scalability for complex analyses. In contrast, MATLAB offers powerful numerical computing and advanced data analysis features but requires programming skills, potentially posing a steeper learning curve. Notepad, a simple text editor, serves as a lightweight tool versatile for viewing and handling structured data, albeit lacking advanced data analysis features. This strategic tool selection allows us to rigorously assess the generalizability and effectiveness of ContextMate across diverse applications and user proficiency levels.

For an illustrative use case of ContextMate, as illustrated in Fig. 1, The user involves in data analysis tasks, operates with different datasets across multiple applications. In the first application, MATLAB, the "Olympic_Medals" table is loaded, which comprises information pertaining to athletes' Name, Sex, Age, Country, Sport, Year, Bronze Medals, Gold Medals, Silver Medals and Total Medals, The second application, Microsoft Excel, imports the "colleges" table, which contains data about United States universities, including Name, Region, Admission Rate, Locale, Control, ACT Median, SAT Average, Population, Average Cost, Expenditure, Median Family Income, Average Faculty Salary, Median Earnings and Median Debt. The third application, Text Editor Notepad, holds data about movies, including columns for Title, Budget, Release Year, Worldwide Gross Production, Running Time, Genre, Content Rating, Creative

Type, IMDB Rating and Rotten Tomatoes Rating. In order to determine the user's intention, ContextMate observes the user's interactions with data analysis applications and generates a prioritized list. Based on recent user interactions, the application with which the user has most recently interacted, in this case MATLAB, is positioned at the top of the priority list. The user makes a natural language query to ContextMate requesting "Show average gross across genres for Creative Type as science fiction and fantasy movies". Despite MATLAB being at the top of the priority list, LLM selected the Notepad because "movies" table is highly relevant to the user's query. LLM also designated the plotting tool from the list of available tools as it aligns with the user's intent. Additionally, by leveraging the available few-shot examples for the desired response through the plotting tool prompt, LLM generated task outlines, and ContextMate executed them locally to produce the desired output in the intended application.

We conducted a think-aloud user study with 18 participants, covering 07 different data analysis tasks by utilizing three mentioned software tools in real-life settings. We compared ContextMate with a baseline system that only uses natural language queries without incorporating in-situ context information. For our experiments, we employed the NL4DV (Narechania et al. 2021, 2021) tabular dataset comprised of 08 tables from various fields with 19455 records, with a range of column numbers varying from 08 for the "euro" table to 75 for the "nba-mvp" table. Our results demonstrate that users were able to effectively analyze the given dataset using ContextMate regardless of the application it was opened with. They generated high-quality, multilingual reports with summaries, key findings, and visually-rich representations of data with a high degree of accuracy. The mean expert sufficiency score achieved with ContextMate was 4.25 out of 5 and the average necessity score was 4.50 out of 5. The system's average success rate was 93.0%, which indicates a 20.0% enhancement over the context-unaware baseline. The experimental results indicate that the use of ContextMate reduces the mental and physical workload of users in generating detailed contextual information, thereby facilitating a more natural assessment of the system's performance. Our study revealed that MS Excel users showed a preference for conditional formatting, filtering, sorting and plotting, while using MATLAB and CSV files users expressed interest in automated analysis reports. In all tasks, ContextMate significantly outperformed the baseline in various subjective scoring metrics including user satisfaction, mental effort, accuracy and efficiency. Additionally, the analysis reports generated by ContextMate demonstrated personalized features.

Our approach aims to offer users a platform that allows them to solely concentrate on the primary task of data analysis, freeing them from the need to generate context for the LLM. This not only facilitates effective and efficient data analysis but also provides users with a flexible and customizable system that can be tailored to their specific needs.

Hence, this research effort contributes to enhance the user experience and adaptability in human computer interaction, thereby fostering the creation of context-aware applications, particularly in the domain of data analysis.

2 Background and related work

The development of LLMs (Brown et al. 2020; OpenAI 2023; Thoppilan et al. 2022) capable of generating human-like responses to natural language queries has reignited researchers' interest in the development of natural language interfaces (NLIs) (Feng et al. 2023a, b; Hoque et al. 2022; Liu et al. 2021; Luo et al. 2022, 2021a, b; Maddigan et al. 2023, 2023; Narechania et al. 2021, 2021; Setlur et al. 2016; Shen et al. 2021; Srinivasan et al. 2021; Tang et al. 2022). However, natural language interaction presents several challenges, with one of the primary obstacles being the incorporation of task-specific context to effectively interpret user intent and generate a system response that achieves the desired action (Min et al. 2022). Despite several proposed solutions, none appear tailored to general purposes that can be utilized to alleviate the users' mental and physical burden by automatically integrating their intentions with their input queries. Here, we provide an overview of the issues and resolutions that have been identified in research on human-AI interaction in context-aware scenarios.

2.1 Context-aware natural language interfaces

Incorporating context-awareness in the data analysis can significantly enhance the user experience while querying for a task in progress. The system can provide detailed information to the LLM and generate context based on the current task, which can improve the relevancy of the results presented to the user. This approach can augment the user's concise query, resulting in improved user satisfaction and reduced frustration or confusion that may arise from vague or inaccurate queries. Thus, incorporating context-awareness into the natural language user interface can lead to more efficient and satisfying interactions with the system. (Gauselmann et al. 2023)

Our research sheds new light on existing studies. For instance, Setlur et al. Setlur and Tory (2022) conducted an assessment of a chatbot's interaction design for data analytics. In contrast, our work explores how users working in a data analytics setting can effectively utilize multiple datasets while employing a NLI. We investigate how contextual information can be integrated to better serve users' intentions. Ragavan et al.'s Srinivasa Ragavan et al. (2022) examination of natural language formulas in GridBook is a comparable antecedent to our study, as it pertains to the same application domain of data analysis in spreadsheets, has the same target end-user, and features a NLI. We observed

significant similarities between the two studies, such as our participants' clear acknowledgment of the natural language's convenience in contrast to spreadsheet formulas or programming languages. However, their work is based on a deep learning parser and spatial context generator between cells to understand user's intents underspecified in the natural language query within the scope of GridBook environment. Whereas, our work aims to generate context based on temporal information obtained from users' interactions with multiple instances of various data analysis applications.

Other relevant prior works are the TaskTracer system (Stumpf et al. 1999) and a study by Vuong et al. (2017), which employs a user events observation framework to collect UI traces across diverse office tools. Subsequently, the system leverages data-driven models to infer the active task of a user and link the relevant information entities, such as documents or websites, with the deduced task. Such a method facilitates the delivery of information entities pertinent to the user's task, which is similar to the goal of our research. Notably, our approach does not entail the continuous recording of user interaction traces.

Rath et al. introduced an ontology-based user interaction context model (UICO) (Rath et al. 2009) to improve the accuracy of task detection on a user's desktop. Their model utilizes granular contextual attention metadata collected from the desktop UI operations and employs a combination of heuristic, data retrieval, and machine learning techniques for its autonomous enrichment. Whereas, we used UI interactions with the relevant applications and utilized LLM with the use of prompt engineering to identify users' intent from their queries.

2.2 Few-shot prompting

Instructing language models (LMs) to effectively follow users' queries is a challenging task that can be addressed by utilization of manually crafted examples showcasing the expected responses. Two standard approaches to implementing such techniques include few-shot prompting and regular gradient-based learning. The former involves providing the LM with handful examples to provide extra contextual clues when inferring, without performing any changes to model coefficients, as suggested by Brown et al. (2020). The latter typically involves supervised learning after foundational pre-training aimed at language comprehension, as described in the work of Ouyang et al. (2022) and Chung et al. (2022) with recent work by Taylor et al. (2022) proposing to combine foundational texts with explicitly reasoned human-labeled examples identified by a unique marker. Several researchers employ supervised fine-tuning as a transitional phase, proceeded by reinforcement learning from human feedback (RLHF), as seen in the work of Ouyang et al. (2022) and Nakano et al. (2021). Our work also

employs the human feedback for the purpose of evaluating and enhancing performance. LLMs can learn new tasks from a few examples, even if the labels are wrong. Because other aspects of the demonstrations are crucial for the ultimate task outcome, which includes label space, the dispersion of the entered query, and the comprehensive structure of the examples order (Min et al. 2022).

2.3 In-context External Tools Execution

Few-shot prompting is a widely used method for instructing LMs to effectively perform the desired tasks by presenting them with a few manually authored in-context exhibits of the intended behavior (Press et al. 2022; Suzgun et al. 2022; Wei et al. 2022a, b) and instructing how to use APIs and external tools (Gao et al. 2022; Lazaridou et al. 2022; Liang et al. 2023; Yao et al. 2022). This approach is easy to use as it only needs several handpicked illustrative examples, allows for quick testing as no model refinements are necessary, and allows for reuse of the consistent model across various diverse tasks and tools by altering the given prompt (Brown et al. 2020; Wei et al. 2022b). However, the capability to execute logical thinking with chain-of-thoughts from a few in-context examples only arises when models attain a specific magnitude (Chung et al. 2022; Wei et al. 2022a), and performance depends on the format, choice, and order of the few-shot examples (Jiang et al. 2020; Kumar and Talukdar 2021; Lu et al. 2021; Min et al. 2022; Yao et al. 2022)). Another limitation of this approach is the amount of supervision that can be deliverable, which is constrained by the quantity of examples that can be accommodated within the LM's contextual frame. This is particularly pertinent when 1) a new behavior is so intricate to understand that it calls for more than just a few illustrations, or 2) there is a broad spectrum of feasible actions that the model needs to learn. Furthermore, Since no parameter adjustments are made, the LM's capabilities are entirely linked to the presented instruction, and eliminating it also erases these competencies (Brown et al. 2020).

In consideration of the limitations posed by the context window of LMs, ContextMate generates a brief and focused context. Adds the pertinent schema to the pre-defined few-shot examples that are relevant to the given application. This approach aims to optimize the available supervision by providing a precise set of instructions for the LM to follow within its limited context window.

2.4 LLMs and in-context learning

LLMs are capable of interpreting basic natural language commands (Joshi et al. 2023; Liu et al. 2023a; Maddigan et al. 2023, 2023) and executing them on behalf of the user, thus allowing the creation of a NLI for existing applications.

Table 1 Prevalent NL query examples, supported applications along with contextual information

Natural language queries	Supported apps	Contextual info (Schema + API)
Analyze college table	All	Colleges Table
Analyze the table with at least 15 Key findings		Active Table + Analysis(API)
Highlight Southwest colleges	Excel	Colleges Table
Format female Gymnastics players from the United States		Olympic_medals Table + Format(API)
Show only the private colleges	All	Colleges Table
Select only Canadian “Male” Players with less than 25 years		Olympic_medals Table + Filter(API)
Plot the line chart shows Total Medals with year	All	Olympic_medals Table + Plot(API)
Plot bar chart of Gold, Silver and Bronze Medals Vs Top 10 Countries		
Analyze the table and Export pdf Report	All	Active Table+Export_PDF
Analyze Olympic table and copy Results to the clipboard		Olympic_medal+Export_CB + Analysis(API)
Sort selected column in descending order	All	Active Table
Sort by Sports		Olympic_medals + Sort(API)
Average Expenditure of colleges	All	Colleges Table
Put formula in the current cell for Adding medals for France?		Olympic_medals + Formula(API)

However, the integration of contextual information is essential for guiding LLM in understanding the user’s intent. Incorporating context explicitly requires a great deal of effort, both mental and physical, and can easily lead to an unsatisfactory user experience (Gauselmann et al. 2023). Hence, the emergence of LLM (Brown et al. 2020; OpenAI 2023) has encouraged researchers to re-evaluate the conventional methods of human–machine interaction and explore novel approaches to achieve more natural and intuitive interactions. Transformer-based LLMs (Vaswani et al. 2017) have recently resulted in significant advancements in NLP. Pre-trained on an extensive corpus of unlabeled textual data, incorporate wide-ranging, universal aspects of language in terms of grammar and semantics. Such characteristics can assist in enabling a more dynamic range of downstream applications (Liu et al. 2023b) - Instead of needing to train a new model for every specialized task, users can just customize the model by feeding it natural language cues during execution, E.g., average gross across genres example in the introduction section. Such capability to identifying the requisite task instantaneously is known as in-situ learning (Brown et al. 2020). Recently, Khatry et al. Khatry et al. (2023) presented a deductive reasoning framework for data manipulation programs based on in-context learning, which utilizes the data context comprehensively by utilizing both task inputs and outputs collectively. The authors introduced a technique called semantic reranking, which involves reranking the programs formulated by LLMs influenced by three indicators: semantic filtering and well-formedness-based score tuning,

semantic interleaving, and results-oriented scoring adjustment by considering the program outputs.

2.5 Natural language programming

It has become increasingly evident that pretrained LLMs Khatry et al. (2023); Chen et al. (2021); Wang et al. (2023a, 2023b) offer cutting-edge efficacy in multiple natural language processing (NLP) endeavors. As a result, these frameworks are becoming increasingly popular in a range of industrial applications that heavily rely on NLP. Additionally, LLMs can generate code from natural language Chen et al. (2021); Joshi et al. (2023); Liu et al. (2023a); Maddigan et al. (2023, 2023); Wang et al. (2023a, 2023b), but only a small portion of natural language is efficient in steering code generation. A new technique called grounded abstraction matching Liu et al. (2023a) is proposed to bridge the gap between natural language and code by translating code back into natural language. This scheme helps end-users in developing their mental model for future interaction with the system. It is implausible that the LLM would generate executable code without any errors. To address this issue, A. Ni, et al. proposed a solution in their work Ni et al. (2023) called LEVER, which involves refining CodeLMs while transforming natural language directives into code by utilizing separate verification models that assess the accuracy of the generated code sequences. These verification models take into account the execution results of the programs.

It is pertinent to mention here that the prior research studies Liu et al. (2023a); Maddigan et al. (2023, 2023); Ni et al. (2023); Wei et al. (2022b); Yao et al. (2022) in natural language to code generation utilized the Codex model, also known as code-davinci-002, for evaluation purposes. However, this model has been surpassed by newer models, including text-davinci-003 and several chat-based models such as gpt-4 and gpt-3.5-turbo. Therefore, in our research, we employed OpenAI's updated gpt-3.5-turbo model for tasks related to generating code and data analysis reports.

2.6 ChatGPT code interpreter

In recent developments, OpenAI has unveiled a Code Interpreter OpenAI (2023) feature integrated within ChatGPT, marking a significant advancement in democratizing data analysis for non-experts. While the system performs admirably in handling data manipulation and graphical representation tasks, it exhibits several limitations that cannot be overlooked. Firstly, the architecture necessitates the transfer of user data to external servers, posing potential risks to data privacy and security—an issue that inhibits some users from leveraging the service. Additionally, the system occasionally falters in accurately capturing the user's intentions, and it is also constrained by its inability to interface with external databases or analytical tools. Furthermore, the platform faces storage capacity limitations, which cumulatively act as barriers to its widespread implementation. In response to these challenges, we advocate for the ContextMate initiative—a solution predicated on local deployment strategies and supported by a context-aware API framework. This approach is designed to overcome the aforementioned limitations, thereby cultivating a more favorable landscape for data analysis applications. ContextMate employs a rigorous methodology that leverages external tools and APIs to generate code through the use of LLMs, obviating the need to upload sensitive data. The generated code is subsequently executed in a secure, local environment, allowing for seamless integration of all required packages and libraries.

3 Formative user study

In the formative user study, we engaged in informal discussions with data analysts to gather insights into user preferences, guiding the development of the ContextMate prototype.

3.1 Participants

To aid in the development of the ContextMate prototype, we conducted informal discussions with seven data analysts from diverse departments, such as procurement, finance,

HR, and R & D, across three distinct organizations. These participants, comprising 4 males and 3 females, ages ranged from 29 to 41 years, and their professional experience varied from 5 to 15 years. All participants graciously volunteered their time and expertise for the study, contributing invaluable insights to our research.

3.2 Findings

Our primary objective was to gather insights into user preferences for specific data analysis applications, including MS Excel, MATLAB, and simple Text-Editors (For CSV Files), along with their preferences for various analysis tasks such as structural data Analysis, Formatting, Filtering, Plotting, Sorting, applying Formulas and Exporting analysis report through NL textual and verbal queries. The gathered information was used to inform the design of the prototype, including the functionalities it should offer and the data analysis tools it should support. Table 1 shows the proposed data analysis APIs with queries' examples, as per our observations using the NL4DV dataset Narechania et al. (2021, 2021).

4 System design

ContextMate consists of two modules (1) Background service and (2) user interface for serving the user queries. The service operates in a dedicated thread and continuously monitors the user's interaction with the data analysis applications. This is achieved through the integration of an event listener that captures user input and system responses, builds the priority list of data analysis applications by placing the latest interaction application at the top of the index, and so on. The service algorithm detailed in Algorithm 1 encapsulates ContextMate's real-time user interaction tracking, showcasing its agility in adapting to dynamic user engagement patterns with data analysis applications.

In our implementation, the generation of priority lists is a generalized process. It is designed to work seamlessly across a spectrum of scenarios, ranging from a single instance of one software to multiple instances of various software running simultaneously. This flexibility is a core feature of ContextMate, ensuring its adaptability to diverse user environments. When only one software instance is active (i.e., there is only one entry in the priority list), ContextMate automatically identifies this scenario. In such cases, our system smartly skips the contextualization prompt, as the contextual information is straightforward with only one active application. This design decision was made to streamline the user experience, reducing unnecessary steps and enhancing the system's efficiency. For the source code of priority list generation service, refer to Appendix 1.

Algorithm 1 Generation of priority list

```

1: Define target_apps  $\leftarrow$  ['excel.exe', 'matlab.exe', 'notepad.exe']
2: Initialize active_apps_history  $\leftarrow$  []
3: function MONITOR_WINDOWS
4:   while True do
5:     REMOVE_CLOSED_APPLICATIONS
6:     interacting_apps  $\leftarrow$  GET_INTERACTING_APPLICATIONS
7:     if interacting_apps is not empty then
8:       UPDATE_ACTIVE_APPS_HISTORY(interacting_apps[0])
9:     end if
10:    Share active_apps_priority_list with ContextMate
11:    wait asynchronously for a short duration
12:  end while
13: end function
14: function REMOVE_CLOSED_APPLICATIONS
15:   Initialize current_windows  $\leftarrow$  {}
16:   for each window in all open windows do
17:     pid, title  $\leftarrow$  GET_WINDOW_THREAD_PROCESS_ID(window), window.title
18:     Add (pid, title) to current_windows
19:   end for
20:   Remove entries from active_apps_priority_list if not in current_windows
21: end function
22: function UPDATE_ACTIVE_APPS_HISTORY(foreground_app)
23:   current_time  $\leftarrow$  TIME
24:   if foreground_app and foreground_app.name in target_apps then
25:     new_active_key  $\leftarrow$  (foreground_app.pid, foreground_app.title)
26:     updated  $\leftarrow$  False
27:     for each app in active_apps_priority_list do
28:       if (app.pid, app.title) = new_active_key then
29:         Copy app to app_copy
30:         Remove app from active_apps_priority_list
31:         Update app_copy.total_active_time and app_copy.last_update_time
32:         Insert app_copy at the beginning of active_apps_priority_list
33:         updated  $\leftarrow$  True
34:       break
35:     end if
36:   end for
37:   if not updated then
38:     Initialize foreground_app.total_active_time and last_update_time
39:     Insert foreground_app at the beginning of active_apps_priority_list
40:   end if
41: end if
42:   Update last_update_time for other apps in active_apps_priority_list
43: end function
44: function GET_INTERACTING_APPLICATIONS
45:   Initialize interacting_apps  $\leftarrow$  []
46:   Get list of all running processes, filter by target_apps
47:   Get currently focused window 11
48:   if focused window's process is in target_apps then
49:     Update active_apps_priority_list
50:     Add focused window to interacting_apps
51:   end if
52:   Iterate through open windows of target_apps
53:   Check window state changes and update interacting_apps
54:   return interacting_apps
55: end function
56: Main: Run MONITOR_WINDOWS asynchronously

```

ContextMate has web-based UI as shown in Figure 2, constructed using Streamlit Open Source (2023a), which is an open-source Python library for building web-based applications, enabling users to input query via manual entry or voice commands. The integration of the proposed technology facilitates the encapsulation of several key components, including the UI, LLM connectivity, incorporation of contextual information with the available tools' descriptions via prompt engineering, and feedback mechanism for evaluation of ContextMate. It also demonstrates the system's interactive and user-centric design.

4.1 Architecture

Figure 3 depicts the architecture of ContextMate, showcasing the inter-connectivity and interdependence of these components in realizing the system's objectives.

4.2 Context generation

Structured data is composed of two components: headers and the actual data. Headers that are clear and meaningful assist in comprehending the data field, thereby benefiting for data analysis by using LLM. By analyzing the data, we determine the datatype of the field, which aids in filtering and analyzing queries. For instance, date/time queries have different implications than string-related queries. Additionally, statistics such as mean ($\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$), mode, range ($\max(x_i) - \min(x_i)$), median, minimum ($\min(x_i)$), maximum ($\max(x_i)$), count, and sum ($\sum_{i=1}^n x_i$) can be computed on the numerical fields. Additionally, sample variance ($s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$), standard deviation ($\text{st.dev} = \sqrt{s^2}$), standard error ($SE = \frac{s}{\sqrt{n}}$), kurtosis ($\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^n \left(\frac{x_i - \bar{x}}{s} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$), and skewness

Algorithm 2 Create table schema

```

1: procedure CREATETABLESCHEMA
  ▷ Call the function to get column names and data types
2:   columnNames, columnDataTypes ← GETCOLUMNSNAMESANDDATATYPES
  ▷ Initialize separator for merging column names and data types
3:   separator ← ' as '
  ▷ Merge column names and data types using the separator
4:   mergedList ← [f' [{a}] {separator} {b}'
    for a, binzip(columnNames, columnDataTypes)]
  ▷ Create a string schema with the merged list of columns
5:   schema ← "Table MyDataTable, columns = [{ }].format(
    ", ".join([f' {s}' for s in mergedList]))
  ▷ Return the schema
6:   return schema
7: end procedure
8:
9: procedure GETCOLUMNSNAMESANDDATATYPES
  ▷ Get the active workbook from the session state
10:  active_wb ← st.session_state.active_wb
  ▷ Get the active worksheet from the session state
11:  active_ws ← st.session_state.active_ws
  ▷ Get column names from the first row of the worksheet
12:  columnNames ← active_ws.range('A1').expand('right').value
  ▷ Get data types from the second row of the worksheet
13:  columnRange2 ← active_ws.range('A2').expand('right')
14:  columnDataTypes1 ← [type(cell.value) for cell in columnRange2]
15:  columnDataTypes ← [str(elem).split("'")[1] for elem in columnDataTypes1]
  ▷ Return column names and data types
16:  return columnNames, columnDataTypes
17: end procedure

```

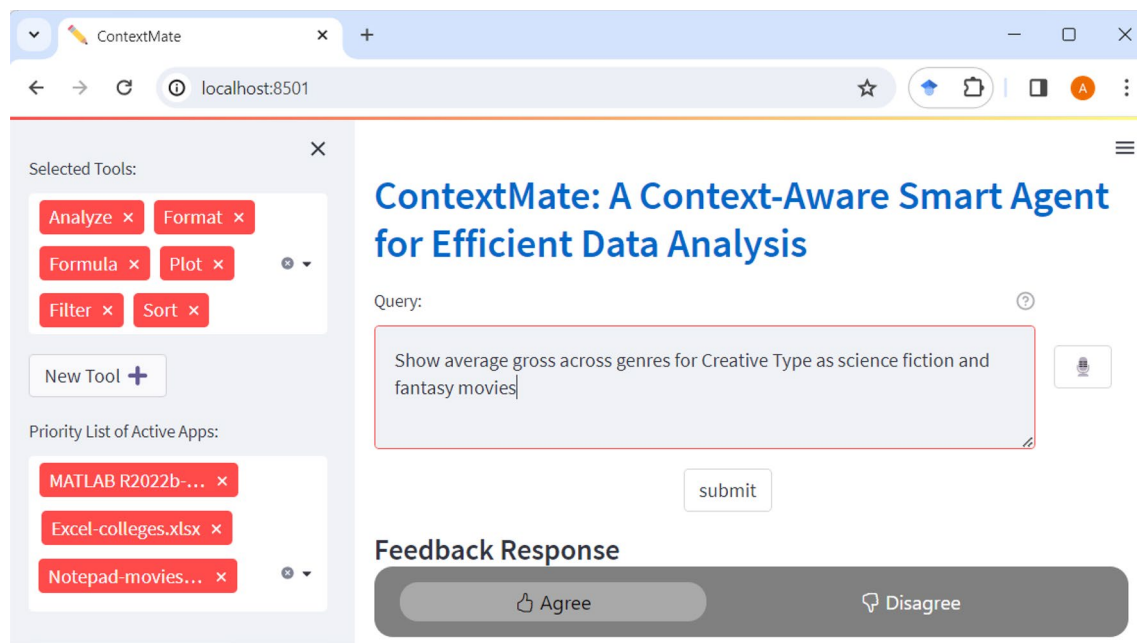


Fig. 2 Web-based UI of ContextMate displaying the tool selection panel, real-time priority list of active applications, query input area with oral input option, and feedback response widget for user interaction and system evaluation

$(\frac{n}{(n-1)(n-2)} \sum_{i=1}^n (\frac{x_i - \bar{x}}{s})^3)$, are also important statistical measures that can be calculated. Combination of the headers with their corresponding data-types is known as a schema. Figure 4: Shows the data schema of Olympic_Medals from NL4DV (Narechania et al. 2021, 2021) tabular dataset.

When user submits the analysis query, ContextMate generates the schema of the imported data from all active instances of the concerned applications and prioritize them according to the priority list generated by the service. This information is integrated into contextualization prompt (Table 3), requesting LLM to select the most suitable application instance based on the provided context. LLM

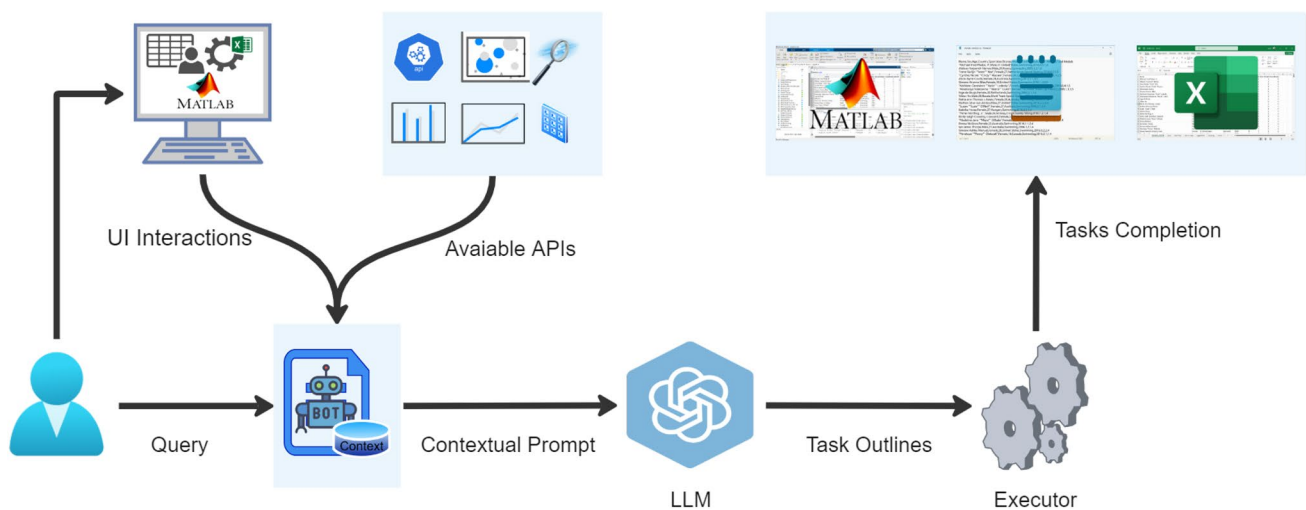


Fig. 3 Overview of ContextMate: Given a user query ContextMate augment it with the UI Interaction traces and available APIs descriptions, engineer a prompt by utilizing pre-drafted template feed to

LLM. Then, the LLM contextualize and generates the task outlines considering the best matched app with relevant APIs. Next, Executor runs the action code and update the intended app

Table Olympic_Medals, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float']

Fig. 4 Sample of a tables schema

Table 2 Customized parameters of LLM for various APIs

Param/APIs	Context	Analysis	Filter	Format	Sort	Plot
Temperature	0	0.5	0	0	0	0
Max Tokens	50	4000	300	300	300	300
stop	;	\n\n	< /code>	< /code>	< /code>	< /code>
			;			plt.show

contextualize the user query and returns the instance that best matches it.

4.3 Data analysis assisting APIs

To support our hypothesis, along with Contextualize API, we implemented analysis aiding custom APIs based on the insights gathered from our initial informal discussions with data analysts, i.e., analysis, format, filter, sort, plot, export, and formulas. In addition, there are some application specific APIs i.e., new sheet, add/remove columns/rows etc. Each API is accompanied by a textual description that outlines its specific role.

In the development of our Data Analysis Assisting APIs, we strategically selected Python libraries to enhance functionality and efficiency. XLWings (Open Source 2023b) is used for its adept integration with Microsoft Excel, allowing for powerful data manipulation. For MATLAB integration, The MATLAB Engine API for Python (The MathWorks 2022) is employed, enabling Python to utilize MATLAB as a computational engine. Additionally, the pandas library (team 2023) is incorporated for its proficiency in handling and processing CSV file data, ensuring efficient data analysis. These choices reflect our commitment to leveraging advanced, efficient tools in computational analysis.

After contextualization, the selected context information and the descriptions of available APIs are incorporated with the user's input query. This combined data is then fed to the LLM, which determines the appropriate analysis API according to the description of the API. Next, a specific tool chain is selected based on the identified API. Finally, the LLM is consulted again using pre-drafted prompt (Appendix 2) of the nominated API to generate the tasks outlines for the executor. This process allows for the seamless integration of contextual information and the efficient selection of appropriate APIs for data analysis tasks.

Compared to ContextMate, the baseline version does not integrate the contextual information about the target application and table schema with the users' query, i.e., the context information path represented by the UI interaction arrow of Fig. 3 is removed. Both versions share the pre-defined prompts and analysis APIs.

4.4 LLM customized parameters

In our implementation, we used the gpt-3.5-turbo² model from the OpenAI API as the underlying LLM. This particular model was selected for its exceptional ability to follow instructions accurately and efficiently. Both the ContextMate and the baseline version rely on the LangChain³ (Chase 2023) framework, which provides an interface for cooperating LLM with the various custom APIs. We used the text-completion endpoint of the OpenAI API and customize the parameters as described in Table 2. Model's temperature parameter that regulates the degree of randomness in the generated text is zero for all tools except analysis tool increasing the temperature value (e.g., 0.6) tends to produce more creative and diverse outputs, whereas a lower temperature (e.g., 0.1) results in more focused and deterministic outputs, for code generation task its recommended (Srinivasa Ragavan et al. 2022) to set it zero. The response token limit differs for each tool. E.g., the analysis tool has a token limit of 4000, while the formula, filter, plot and format tools have a token length of 300. The stop parameter also varies depending on the tool being used. With the exception of the contextualize and analysis tools, all other tools require Python code or SQL queries. Therefore, we set the stop parameter to "< /code >" as we want to avoid the LLM to generate the multiple scripts and we also include a semi-colon for SQL queries and "plt.show()" for the plot tool.

² <https://platform.openai.com/docs/models/overview>

³ <https://github.com/hwchase17/langchain>

4.5 Prompting LLM

We propose a novel class of prompting techniques specifically designed to adapt LLMs for data analysis tasks involving structural data. LLMs have the capability to enable learning within the context using minimal examples through prompting, which involves providing the LLM with a few input–output samples from the specific downstream task (Brown et al. 2020; Chowdhery et al. 2022; Wei et al. 2022b; Zhou et al. 2022), instead of the traditional approach of fine-tuning or readjusting models for every novel assignment. Previous research has shown that few-shot prompting competes well with traditional benchmarks (Brown et al. 2020) for various NLP tasks like translation or question-answering. However, applying prompting techniques to LLMs for data analysis poses unique challenges and lacks established methodologies. Firstly, feeding the sensitive data to LLM, may raise security fears from some individuals and organization for their data confidentiality. Moreover, directly inputting the whole structured data into LLMs is impractical as it may comprises billions of data samples, which can readily surpass the input length limits of LLMs. Furthermore, there could be relationship between multiple tables need to upload all tables again cumbersome and charge per token by LLM.

Therefore, In our methodology, we have meticulously prioritized data privacy and security. To this end, our system is intricately designed to operate exclusively on the data schema, detailed in Algorithm 2, and illustrated in an example of the generated schema in Fig. 4 effectively circumventing the need to handle or transmit actual data instances. This strategic approach is firmly rooted in our commitment to uphold privacy-conscious practices, ensuring that no real data values are utilized or compromised during the task generation process with the LLM. This approach also supports to restrict token utilization, considering the cost charged per token by the LLM.

Table 3 illustrates the sample contextualization prompt, which contains application instances with data schema followed by question-answer pairs. ContextMate provides contextual information, such as application name and instance ID, which is sorted according to user interaction in temporal order. Additionally, the table 3 includes a user's NL query and the response generated by the LLM.

After contextualizing the user's query, the subsequent step is to categorize it with an appropriate analysis tool. For this, we utilize the concept of agents and tools from LangChain (Chase 2023), which facilitates the selection of the optimal tool among the available data analysis tools for the user's query. Each tool is composed of three parameters: a name, function, and description. The list of available tools is provided to the LLM, which correlates the user's query with the tool description and selects the best tool. Once the tool

is selected, ContextMate executes it to provide a response to the user's query. The prompts are designed to cater to isolated tasks within each software, and there is no direct data communication or connection established among the three selected tools. For additional technical information on the available prompts and corresponding sample responses, please refer to Fig. 7 in Appendix 2.

4.6 Executor functionality

The executor component is essential for carrying out the actionable code generated in response to user queries, within the selected data analysis application. This execution process is facilitated by a Python Read, Evaluate, Print, and Loop (REPL) system (Van Binsbergen et al. 2020), adept at handling dynamic code execution, is integrated with various libraries: xlwings (Open Source 2023b) for Microsoft Excel, MATLAB Engine API for Python (The MathWorks 2022) for MATLAB, and the pandas library (team 2023) for managing CSV files. The integration of these tools allows ContextMate to translate the prompts from the LLM into actionable Python code, which runs within the REPL environment. The Python REPL's not only executes the generated code but also embeds the results directly into the context of the user's data analysis application. The user feedback loop is incorporated to continuously enhance the system's accuracy.

Table 4 Shows contextual information and natural language queries along with their corresponding results.

4.7 User feedback mechanism

ContextMate employs a user feedback mechanism within its GUI to assess the accuracy of generated context. Users can provide immediate feedback through thumbs up and thumbs down buttons, offering valuable insights. User feedback serves as a catalyst for improvement, especially when disagreements arise. In such cases, an automatic learning process is initiated, prompting a follow-up dialogue with users to pinpoint specific issues, whether related to API selection or chosen applications. The system then consults the LLM, excluding previously identified error-causing application or API, iteratively learning from mistakes. This feedback loop will also aid in fine-tuning the LLM, enhancing its ability to generate context accurately and lays the foundation for future system refinements.

5 Evaluation

To evaluate our prototype system's usability, we conducted a user study in a real-life situation. The aim of our study was to evaluate the efficacy of seamlessly integrating

Table 3 Contextualizing prompt

Contextualizing prompt	
Prefix	<p>Classify the questions according to the Tables in the application</p> <p>Here are some Examples</p> <p>Question: MATLAB01 application contains Table Olympic_Medals, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float']</p> <p>EXCEL01 application contains Table Colleges, columns = ['[Name] as str', '[Control] as str', '[Region] as str', '[Locale] as str', '[Admission Rate] as float', '[ACT Median] as float', '[SAT Average] as float', '[Population] as float', '[Average Cost] as float', '[Expenditure] as float', '[Average Faculty Salary] as float', '[Median Debt] as float', '[Median Family Income] as float', '[Median Earnings] as float']</p> <p>NOTEPAD01 application contains Table movies-w-year, columns = ['[Title] as str', '[Worldwide Gross] as float', '[Production Budget] as float', '[Release Year] as float', '[Content Rating] as str', '[Running Time] as float', '[Genre] as str', '[Creative Type] as str', '[Rotten Tomatoes Rating] as float', '[IMDB Rating] as float']</p> <p>if question does not classify to any of the Tables in the application. respond with the first app like MATLAB01 in the example</p> <p>Q: Plot bar chart of Normalized data Worldwide Gross, Production Budget ratings Vs Genre?</p> <p>A: NOTEPAD01</p> <p>Q: Plot the Admission by Regions in the desc order?</p> <p>A: EXCEL01</p> <p>Q: Format female players those play Gymnastics and from United States?</p> <p>A: MATLAB01</p>
Context	<p>Question: EXCEL05 application contains Table Olympic_Medals, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float']</p> <p>NOTEPAD02 application contains Table Colleges, columns = ['[Name] as str', '[Control] as str', '[Region] as str', '[Locale] as str', '[Admission Rate] as float', '[ACT Median] as float', '[SAT Average] as float', '[Population] as float', '[Average Cost] as float', '[Expenditure] as float', '[Average Faculty Salary] as float', '[Median Debt] as float', '[Median Family Income] as float', '[Median Earnings] as float']</p> <p>MATLAB03 application contains Table movies-w-year, columns = ['[Title] as str', '[Worldwide Gross] as float', '[Production Budget] as float', '[Release Year] as float', '[Content Rating] as str', '[Running Time] as float', '[Genre] as str', '[Creative Type] as str', '[Rotten Tomatoes Rating] as float', '[IMDB Rating] as float']</p> <p>MATLAB04 application contains Table MyDataTable, columns = ['[Product Category] as str', '[Region] as str', '[Year] as int', '[Sales] as str', '[Revenue] as float', '[Profit Margin] as float', '[Sales Channel] as str', '[Payment Method] as str', '[Advertising Spend] as int', '[Customer Segment] as str', '[Customer Lifetime Value] as int', '[Sales Rep] as str']</p>
User query	Q: Show average gross across genres for Creative Type as science fiction and fantasy Movies?
Response	A: MATLAB03

(1) Prompt prefix contains application instances with data schema and associated question-answer pairs. (2) ContextMate generated contextual information that includes application name coupled with instance ID, data schema sorted according to user interaction in chronological order. (3) User's Natural Language Query and (4) Response Generated by Language Model

contextual information to improve the response of LLMs in data analysis tasks. We implemented ContextMate prototype for test-bed applications such as Microsoft® Excel® (16.0), MATLAB R2022b, and Notepad on Microsoft windows 11 OS, using Python 3 and compared it with a context-unaware baseline version that only uses natural language queries. Using a within-subject design, we asked all participants to use both versions of the assistant. In the baseline version, users are required to manually input their intent and provide information to the LLM regarding the source application and tables of interest in order to establish the necessary context.

5.1 Participants

Our study comprised 18 participants (P1-P18), with a gender distribution of 7 females and 11 males. These participants were selected from two different research and development organizations, specifically their technical and data analysis departments, along with graduate students from a local university. The age range of the participants varied from 23 to 41 years, with an average age of 29.

Among the participants, six (P1-P6) were senior data analysts who possessed extensive working experience of more than eight years. They demonstrated proficiency in

Table 4 Contextual info and NL queries along with corresponding results

Active MATLAB Instance

Olympic_medals Table

Analysis API

Analyze the table with at least 15 Key findings

Refer to Appendix B1

Active Notepad Instance

colleges Table

Formula API

Average Expenditure of colleges

SELECT AVG([Expenditure]) FROM colleges

AVG([Expenditure])

010,446.0931

Active Excel Instance

Olympic_medals Table

Format API

Format female Gymnastics players from the United States

Active Notepad Instance

Olympic_medals Table

Plot API

Plot bar chart of Gold, Silver and Bronze Medals Vs Top 10 Countries

Active Excel Instance

Olympic_medals Table

Filter API

Select only Canadian 'Male' Players with less than 25 years

Active Excel Instance

Housing Table

Sort API

Sort houses by Lot Area

A	B	C	D	E	F	G	H	I	J
Name	Sex	Age	Country	Sport	Year	Bronze	Gold	Silver	Total
Courtney Lynn McCool	F	16	United States	Gymnastics	2004	0	0	1	1
Tatyana Vasilyevna Ananko	F	16	Belarus	Rhythmic G	2000	0	0	1	1
Andreea Roxana Acatrinei	F	16	Romania	Gymnastics	2008	1	0	0	1
Lauren Zoe "Laurie" Hernan	F	16	United States	Gymnastics	2016	0	1	1	2
Jiang Yuyuan	F	16	China	Gymnastics	2008	0	1	0	1
Carly Rae Patterson (Caldw	F	16	United States	Gymnastics	2004	0	1	2	3

A	B	C	D	E	F	G	H	I
Lot Area	Lot Config	Home Type	Roof Style	Foundation	Basement	Heating Ty	Central Air	Rooms
1300	Inside	Townhous	Gable	CBlock	561	GasA	Y	5
1477	Inside	Townhous	Gable	CBlock	630	GasA	Y	3
1491	Inside	Townhous	Gable	CBlock	630	GasA	Y	3
1526	Inside	Townhous	Gable	CBlock	630	GasA	Y	3
1533	Inside	Townhous	Gable	CBlock	546	GasA	Y	6
1533	Inside	Townhous	Gable	CBlock	630	GasA	Y	3

programming languages such as SQL and Python, as well as data analysis software like MS Excel (Corporation 2019) and MATLAB (The MathWorks 2022). Another group of six participants (P7-P12) were data analysis researchers who had over five years of experience in NLIs and had published research papers related to the field. The remaining six participants (P13-P18) were graduate students enrolled at a prestigious university. When asked about their expertise in data analysis using NLIs, they self-rated themselves as either intermediate users or beginners. All participants voluntarily agreed to participate in the study.

Notably, three participants (P2-P4) who took part in the initial study phase were also included in the comprehensive evaluation of the ContextMate system. This decision was intentional and aimed at maintaining a continuous and nuanced understanding of user interactions across different stages of system development. By including participants with prior exposure to the system, we sought to gather insights into their evolving experience

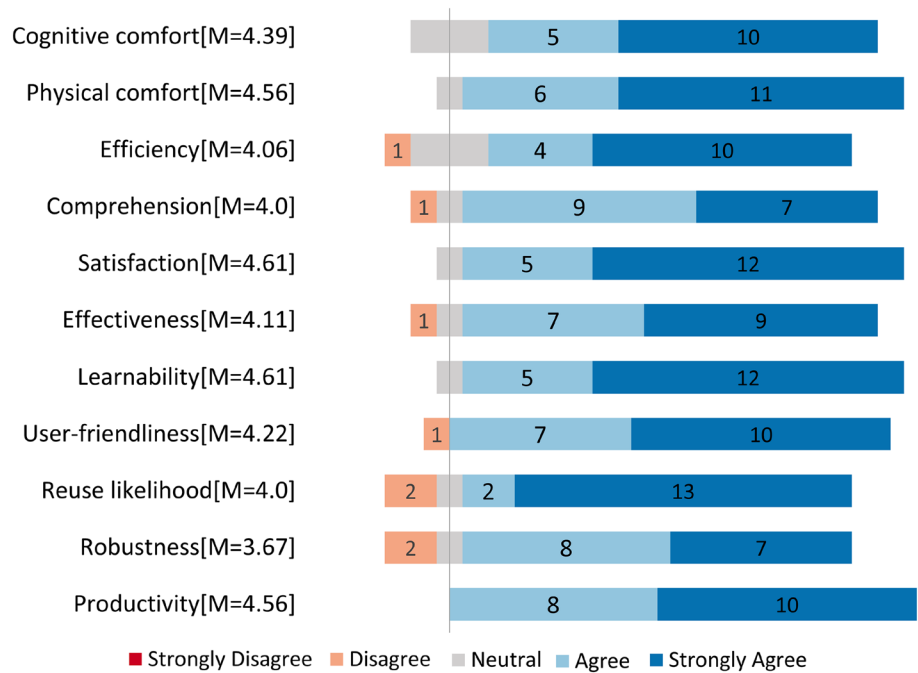
and perceptions as they engaged with the refined version of ContextMate. This approach contributes to the robustness of our evaluation by providing valuable continuity in user experience insights.

5.2 Procedure and tasks

As users can work with multiple applications, instances, and datasets simultaneously, we have identified three distinct use case scenarios: (1) Single or multiple applications, (2) Single or multiple instances, and 3) Single or multiple datasets. This leads to a possibility of $2^3 = 8$ different scenarios.

In the simplest and least common scenario, a data analyst works on a single dataset loaded in a specific application instance. In this case, ContextMate consults the application to create a context based on the data schema and augments the user's natural language query before feeding it to the LLM for generating a response. Whereas, in the most

Fig. 5 Post-experiment questionnaire results: Question types together with the mean values, and the rating Distribution



complex scenario, Where, user works on multiple datasets, in multiple instances of different applications. ContextMate utilizes the UI interaction priority list generated by the service and the LLM to resolve the user's query with the most relevant application instance, while the LLM contextualizes the user's query by considering the data schema of all instances.

The primary objectives guiding our experimental task design encompass the comprehensive assessment of ContextMate's effectiveness across diverse scenarios and its performance evaluation across various applications and instances. In consideration of the participants' limited availability, we requested that they complete nine tasks grouped into three phases, meticulously crafted to capture nuanced aspects of the system's functionality.

1. Phase 1: Employing the same dataset across single instances of each application enables a focused evaluation of ContextMate's capacity to contextualize the desired application based on interaction traces, showcasing its proficiency in handling similar datasets across different tools.
2. Phase 2: Introducing different datasets across single instances of each application explores the system's adaptability to varied data schemas, assessing its com-

petence in analyzing distinct datasets within the same application context.

3. Phase 3: Directing participants to work with different datasets across multiple instances of each application presents a more intricate scenario. This setup facilitates the evaluation of the system's performance in a multi-tool, multi-instance environment, offering insights into its versatility and robustness.

ContextMate Prototype supports six pre-defined data analysis APIs, participants were instructed to choose three distinct tasks of their preference in each phase and allowed to utilize any tables from the NL4DV dataset. This approach facilitated the correlation between the results of participants' queries. It is noteworthy that due to the quiet working environment of the office, only one participant opted for the verbal mode for inquiring the natural language queries.

We administered a post-experiment questionnaire to the participants to elicit their feedback on the following aspects of ContextMate:

1. Cognitive comfort: To what extent did the use of ContextMate require cognitive effort?
2. Physical comfort: To what extent did the use of ContextMate require physical effort?

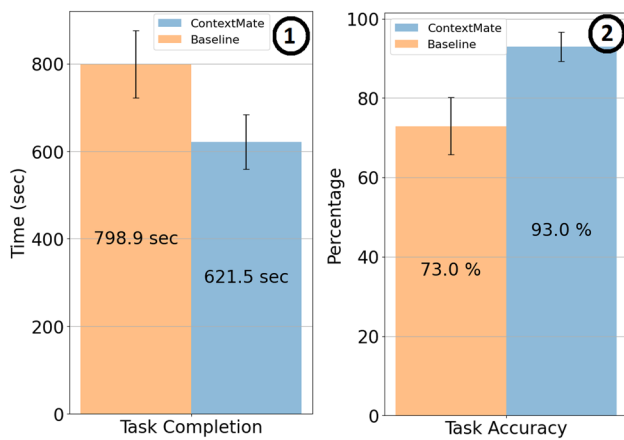


Fig. 6 Task completion analysis: evaluating completion time and accuracy of tasks

3. Efficiency: How would you rate the pace of the interaction with ContextMate?
4. Comprehension: To what extent do you believe that ContextMate understood your intentions?
5. Satisfaction: How satisfied were you with the final results generated by ContextMate?
6. Effectiveness: To what degree did ContextMate enhance your comprehension of its context-aware abilities?
7. Learnability: How effortless was it to acquire proficiency in using ContextMate?
8. User-friendliness: How effortless was it to produce context using ContextMate?
9. Reuse likelihood: How probable are you to employ ContextMate again in the future?
10. Robustness: How robust is ContextMate in handling diverse user queries without encountering errors or failures?
11. Productivity: How did ContextMate affect your overall productivity?

5.3 Results and analysis

5.3.1 Subjective satisfaction

In terms of subjective satisfaction, the participants demonstrated an encouraging outlook towards the system. The experience of using ContextMate received high ratings from users ($\mu = 4.25$, $\sigma = 0.299$). Figure 5 presents the feedback obtained from users regarding ContextMate. Among the eighteen users, fifteen agreed that ContextMate provided cognitive ease, indicating a smooth and effortless user experience. Additionally, seventeen users acknowledged feeling physical comfort while using ContextMate. All users

expressed the belief that ContextMate had improved their productivity, indicating its effectiveness in enhancing task efficiency. Almost all of the participants expressed satisfaction with the learn-ability, user-friendliness, and comprehension of ContextMate.

However, the feedback also highlighted areas for improvement, particularly in terms of robustness. Participants rated the robustness of ContextMate relatively low. During post-experiment discussions, P1-P3 and P7 mentioned that due to the latency of the underlying LLM, they may not prefer using the system as they were already proficient in existing data analysis tools. These findings shed light on the overall satisfaction of users with ContextMate, with positive ratings for cognitive ease, physical comfort, learn-ability, satisfaction, and improved productivity. However, the feedback emphasizes the need to address issues related to robustness and latency to further enhance the system's performance and user experience.

5.3.2 Quantitative comparison

A quantitative comparison was performed to assess the task accuracy and completion time between ContextMate, and the baseline system. Figure 6(1) presents the results of task accuracy, calculated as the average proportion of completed tasks at the individual level, along with a 93% confidence interval. The average task accuracy achieved with ContextMate ($\mu = 0.93$, $\sigma = 0.036$) was significantly higher than that of the baseline system ($\mu = 0.73$, $\sigma = 0.05$), indicating the superior performance of ContextMate in terms of task accuracy.

Regarding task completion time, Fig. 6(2) illustrates the comparison between ContextMate and the baseline system. Participants took less time, on average, to complete the tasks using ContextMate ($\mu = 621.5$, $\sigma = 64.31$) compared to the baseline system ($\mu = 798.9$, $\sigma = 79.6$). The observed difference in completion time suggests that ContextMate offers a more efficient and time-saving approach for task completion in the context of data analysis.

In order to comprehensively evaluate the performance of ContextMate, we also conducted a Wilcoxon signed-rank test to compare both task accuracy and task completion time with the Baseline system. The obtained p-values of 0.00029 and 0.00019 respectively suggest statistically significant differences between ContextMate and the Baseline system in both aspects.

Regarding task accuracy, the significant p-value of 0.00029 indicates that ContextMate achieved higher accuracy compared to the Baseline system. This finding reinforces the superiority of ContextMate in accurately fulfilling user tasks in data analysis. The observed difference is statistically significant, providing strong evidence

to support the claim that ContextMate outperforms the Baseline system in terms of task accuracy.

Similarly, in terms of task completion time, the Wilcoxon signed-rank test yielded a significant p-value of 0.00019. This indicates that ContextMate significantly outperforms the Baseline system by reducing task completion time. The lower p-value strengthens the confidence in the observed difference, confirming the effectiveness of ContextMate in enhancing efficiency and reducing the time required to complete data analysis tasks.

Taken together, these results highlight the superior performance of ContextMate over the Baseline system in terms of both task completion time and task accuracy. The statistically significant differences observed in both measures demonstrate the efficacy of ContextMate in improving the overall efficiency and accuracy of data analysis tasks. The utilization of UI Interaction traces and data schema in constructing contextual information effectively supplements important information that may be missing from user textual queries. This enables ContextMate to generate more comprehensive and thorough prompts for the LLM with lower cognitive and physical efforts. As a result, ContextMate facilitates the faster, more accurate, and more natural mapping of user intent by building contexts.

6 Discussion

The findings from this study suggest that ContextMate is a straightforward yet effective approach to generating contextual information for pre-trained LLMs in data analysis domain. User study indicates that seamless integration of generated context with the user natural language query significantly decrease user fatigue and lower the likelihood of errors, thus boosting user productivity by only emphasizing the descriptiveness of the query.

While concurrently working on multiple datasets, ContextMate dynamically generates the appropriate context and enhances user queries with k-shot pre-crafted examples related to the task, allowing for consultation with LLMs as needed. Our user study showed that generating contextual information enhances accessibility, increases satisfaction, improves comprehension, and accelerates data analysis capabilities for users. Specifically, we pinpointed multiple elements of the context generation process that led to these performance gains. The results of our study show that the main performance gains are attributed to three factors: (1) the seamless generation of context for the user interface environment, (2) the selection of the most relevant API from the context-augmented user query, and (3) the demonstration of a few-shot examples for the desired pattern of actionable code, which further enhances these trends. Our findings suggest broader implications for context generation

and augmentation of user queries for LLM, and also provide directions for future research.

All participants were able to successfully complete all nine tasks within the allotted time frame while using ContextMate. However, when working with the baseline version, participants encountered both mental and physical challenges in constructing the context for the LLM.

In the first phase, the participants were instructed to import data of their choice into all three applications, while ensuring that the same data was used in each application. Our objective was to check the effectiveness of ContextMate in the contextualization of the desired application from the application's interaction traces, as all applications has the same dataset. For instance, P1 imported "**olympic_medals**" table from NL4DV dataset, which had fields such as "Name, Sex, Age, Country, Sport, Year, Bronze Medals, Gold Medals, Silver Medals, **Total Medals**" in all three applications, and interacted with **MATLAB** and asked to "**Draw a bar chart of total medals won by the United States**". Our system was able to comprehend the query and add the context that the user wanted to plot the bar chart using **MATLAB**. Similarly, when the same query was submitted after interaction with the **MS Excel**, ContextMate was able to accurately incorporate the user's intent and display the same bar chart using **MS Excel**. In another scenario, when the user opened the "**olympic_medals.csv**" file in **Notepad** and repeated the same procedure, ContextMate consulted the LLM with Notepad at the top of the priority list. The LLM contextualized the user's intent by realizing that the user wanted to plot using Notepad data. After verifying that **Notepad** already contained the required data, the LLM generated Python code to read the data and plotted the bar chart using Python's matplotlib library. These results demonstrate the effectiveness and flexibility of ContextMate in facilitating the analysis of the same dataset in multiple applications.

In the second phase, the participants were instructed to interact with distinct tables in each application while being limited to opening only one instance of each application. P2, imported "**superstore**" table, which had 27 columns in **MS Excel**. "**movies-w-year**" table, which had 10 columns in **MATLAB** and opened "**colleges.csv**", which had 14 columns in the **Notepad**. P2 interacted with **MATLAB** by manually running some plotting queries, and then asked ContextMate to "**Show a list of the top ten universities sorted by their average faculty salary**". Although the UI interaction list showed **MATLAB** at the top of list, ContextMate consulted the LLM and selected "**colleges.csv**" from **Notepad**, as it contained a relevant column for "**Average Faculty Salary**" that schematically matched the query. In the third phase of the study, participants were instructed to open multiple instances of each application and load different tables in each instance. Participants were limited to ask data analysis queries, including

analysis, format, filter, sort, plot, and formulas. As results shows, ContextMate's average success rate was 93.0% in complex scenario.

Instances of failure were reported when participants were working with datasets that were schematically similar, which was addressed by ContextMate with the incorporation of human feedback. Another significant cause of failure was related to underspecified natural language queries from the users (Maddigan et al. 2023), which could be improved by implementing abstraction matching techniques (Liu et al. 2023a). At times, ContextMate misconstrue the distinction between filter and plot tools, yet the user-centric design of ContextMate readily rectified this issue either by rephrasing the query or by giving priority to the intended tools through the user interface. During the user study, P1 loaded the same dataset in both Excel and MATLAB instances, and even though they interacted solely with the Excel instance, their query explicitly mentioned MATLAB which was not initially taken into account by our prototype. Consequently, the system prioritized the latest interaction application, which was Excel in this case. However, we have since corrected this behavior and repeated that task.

In General, ContextMate was able to identify the appropriate instance and tool by considering the schema of the table and the user's interaction with the application instances. This ensured that the user's requests were fulfilled successfully, resulting in reduced anxiety as the system was comprehensible and their intentions were fully honored.

In conclusion, based on the follow-up survey, all 18 participants provided favorable feedback on the usability of ContextMate. Specifically, the system was described as "Intuitive" (P1, P2, P11) and "Reliable" (P3, P13), the user experience was "fluent" (P6, P14) and "User-friendly" (P7, P8, P15, P18), and it was suggested that it has "potential for integration into a variety of situations" (P1-3, P10).

In underscoring the versatility of the LLM, our user study revealed a crucial aspect. Despite diverse grammatical structures and language usage among participants, the LLM consistently demonstrated a high level of comprehension, accurately discerning users' intentions. It adeptly provided contextually relevant resolutions, showcasing remarkable adaptability to diverse linguistic idiosyncrasies. Notably, this adaptability excelled in handling oral inputs, effectively interpreting queries despite spoken language challenges. Recognizing the complexity of various user inputs, our findings affirm that the LLM's advanced NLP capabilities adeptly address these challenges. The system's robust performance in our user study attests to its effective processing and response to a wide array of linguistic inputs, ensuring accessibility and efficiency for users with different linguistic styles and preferences.

In summary, conversational capabilities of chat based pre-trained LLM (Diederich et al. 2022; OpenAI 2023) are

noteworthy due to its ability to gradually build context and enrich it with additional prompt exchanges. This feature offers significant advantages in interpreting user intent, particularly in relation to fine-tuning data analysis and enhancing overall performance. However, the process of building context through interaction with the LLMs can be time-consuming and resource-intensive for users. In this research we proposed ContextMate, it generates the user mental model, by utilizing the user interaction traces while performing data analysis tasks. User study in real-life settings supports that, if user query augmented with the precisely generated context feed to LLM it quickly understands the user intent and generates the desired response. This approach can be valuable for both saving time and reducing the user's efforts.

7 Limitation and future work

While our research on ContextMate has yielded promising results, we leverage the LLMs for generation of the actionable code for the downstream tasks by utilizing few-shot prompting techniques to feed context information. Where the practical application of LLMs still poses certain limitations. The relatively high response latency of LLMs can have a significant impact on the ContextMate's response time, which can have a detrimental effect on the user experience and ultimately compromise its overall quality. Additionally, the token limit of LLMs can restrict the number of examples in prompts, which can impede their ability to concurrently handle multiplex queries effectively. To address these limitations, fine-tuning the models in future work could be a possible solution. Another possible avenue is to investigate the use of lightweight models for our analysis tasks, similar to the FLAME model (Joshi et al. 2023) for formula authoring assistance. This could help us achieve competitive performance with substantially smaller model sizes and less training data, making it easier to deploy our models in real-world settings.

In addition, there exists a limitation regarding the stochastic nature of LLM's outputs. Nonetheless, the pipeline design of our prototype, which includes pre-crafted prompts with few-shot examples, setting the LLM temperature close to zero for actionable code generation, and setting it to 0.5 for the analysis reports, has mitigated this issue to some extent. To overcome these limitations, we will stay abreast of the latest advancements in LLM research and strive to use more efficient, accurate, and practical LLMs in our system.

Generalization is a critical aspect to consider for ContextMate's real-world applicability. Although ContextMate demonstrates its capacity to acclimate to varied environments. Despite these strengths, challenges in generalization must be acknowledged, given the diversity in data

analysis tools, UI variations, differences in functionalities, data structures and user preferences, requiring tailored adaptations. To address this, future iterations could explore strategies like customizing prompt designs for varied software interfaces, integrating continuous user feedback mechanisms, and ensuring compatibility with a broader spectrum of data schemas.

Our present methodology predominantly utilizes user interaction traces within data analysis applications for context generation; however, a potential avenue for improvement involves exploring alternative modalities, such as the integration of daily activities tracking systems. This exploration aims to offer a more comprehensive understanding of user intentions, enriching the context generation process and enhancing ContextMate's capability to discern user behavior intricacies.

In future research, it would be intriguing to investigate whether the integration of gaze-assisted (Karaman et al. 2018; Ding et al. 2023) user intention can enhance the context information, ultimately leading to an improvement in the performance of data analysis tasks.

8 Conclusion

This paper introduces ContextMate, a new and efficient approach that empowers end-users to generate contextual information on the fly using natural language for carrying out data analysis tasks. In order to address the problem of

incorporating task-specific context into natural language queries for the LLM, ContextMate utilizes user interaction with data analysis applications to generate context. By leveraging the LLM, the system correlates user queries with the schema of imported data, using the best-matching application as a candidate for building the context. This contextual information is then integrated with the most suitable pre-crafted prompts, enabling LLM to provide actionable code for the desired task. In order to enhance the alignment between user intent and generated context, we adopt human feedback to fine-tune the behavior of the proposed framework.

By conducting a user study with 18 participants, across 7 distinct data analysis tasks, using 3 popular analysis tools in real-life environment, we have demonstrated the effectiveness of ContextMate, achieving an average success rate of 93.0% in building right contextual information. ContextMate surpassed the baseline with respect to user satisfaction, reduced cognitive effort, precision, and efficiency, emphasizing the efficacy of integrating in-situ contextual information for data analysis tasks. Furthermore, the participants noted that the smooth generation and incorporation of contextual data enabled them to focus exclusively on the cognitive aspect of data analysis queries, leading to enhanced productivity.

Our research advances the development of context-aware applications within data analysis contexts, opening up avenues for additional investigations into extracting the intentions of desktop office users for their everyday tasks.

Appendix 1 Source Code of Background Service

```

131         'pid': pid,
132         'name': process_name,
133         'title': window.title,
134         'is_focused': False,
135     })
136
137     # Update the previous state for the next iteration
138     prev_window_states[pid] = window.isMaximized
139
140     return interacting_apps
141
142 if __name__ == "__main__":
143     asyncio.run(monitor_windows())

```

Listing 1: Python code for the background service that generates a priority list of active data analysis applications


```

65         break
66
67     if not updated:
68         foreground_app['total_active_time'] = 0.0
69         foreground_app['last_update_time'] = current_time
70         active_apps_priority_list.insert(0, foreground_app)
71
72     for app in active_apps_priority_list:
73         if app.get('last_update_time') is not None and (foreground_app is None or app['pid'] !=
74         ↪ foreground_app['pid'] or app['title'] != foreground_app['title']):
75             app['last_update_time'] = current_time
76
77 async def get_interacting_applications():
78     global active_apps_priority_list
79
80     interacting_apps = []
81
82     # Get a list of all running processes
83     all_processes = psutil.process_iter(['pid', 'name'])
84     process_names = {process.info['pid']: process.info['name'].lower() for process in all_processes}
85
86     # Filter out processes not in target apps
87     process_names = {pid: name for pid, name in process_names.items() if name in target_apps}
88
89     # Dictionary to store the previous state of each window
90     prev_window_states = {}
91     prev_active_window = None
92
93     # Get the currently focused (active) window in a separate thread
94     def get_active_window():
95         nonlocal prev_active_window
96         prev_active_window = win32gui.GetForegroundWindow()
97
98     active_window_thread = threading.Thread(target=get_active_window)
99     active_window_thread.start()
100    active_window_thread.join()
101
102    # Check for changes in window focus for target apps
103    if prev_active_window:
104        _, process_id = win32process.GetWindowThreadProcessId(prev_active_window)
105        if process_id in process_names:
106            process_name = process_names[process_id]
107
108            for index, app in enumerate(active_apps_priority_list):
109                if index != 0 and (process_id == app['pid'] and win32gui.GetWindowText(
110                ↪ prev_active_window) == app['title']):
111                    app['last_update_time'] = time.time()
112                    break
113
114            interacting_apps.append({
115                'pid': process_id,
116                'name': process_name,
117                'title': win32gui.GetWindowText(prev_active_window),
118                'is_focused': True,
119            })
120
121    # Iterate through open windows of target apps
122    for window in gw.getAllWindows():
123        hwnd = window.hwnd # Get the window handle
124        _, pid = win32process.GetWindowThreadProcessId(hwnd)
125
126        if pid in process_names:
127            process_name = process_names[pid]
128
129            # Check if the window state has changed
130            if pid in prev_window_states and prev_window_states[pid] != window.isMaximized:
131                interacting_apps.append({

```

```

1 import asyncio, threading, psutil, win32gui, win32process, time
2 import pygetwindow as gw
3
4 # Define the target applications
5 target_apps = ['excel.exe', 'matlab.exe', 'notepad.exe']
6
7 # List to store the history of active applications
8 active_apps_priority_list = []
9
10 async def monitor_windows():
11     while True:
12         # Check and remove closed applications from history
13         await remove_closed_applications()
14
15         interacting_apps = await get_interacting_applications()
16
17         # Update and print the history of active applications
18         if interacting_apps: # Check if there is any active application
19             update_active_apps_history(interacting_apps[0]) # Update history with the active app
20
21         print("Priority List of active applications :")
22         for app in active_apps_priority_list:
23             print(app)
24
25         # Sleep for a short duration before checking again
26         await asyncio.sleep(1)
27
28 async def remove_closed_applications():
29     global active_apps_priority_list
30
31     # Get current open windows
32     current_windows = set()
33     for window in gw.getAllWindows():
34         _, pid = win32process.GetWindowThreadProcessId(window.hWnd)
35         window_title = window.title
36         current_windows.add((pid, window_title.lower()))
37
38     # Remove entries from history if their PID and title are not in current windows
39     active_apps_priority_list = [app for app in active_apps_priority_list if (app['pid'], app['title'].lower()) in
40         ↪ current_windows]
41
42 def update_active_apps_history(foreground_app):
43     global active_apps_priority_list
44     current_time = time.time()
45
46     if foreground_app and foreground_app['name'] in target_apps:
47         # Define a unique key for each application instance (considering both pid and title)
48         new_active_key = (foreground_app['pid'], foreground_app['title'])
49         # Update or add the foreground window in the history
50         updated = False
51
52         for app in active_apps_priority_list:
53             if (app['pid'], app['title']) == new_active_key:
54                 # Make a copy of the app
55                 app_copy = app.copy()
56
57                 # Remove the original app from the history
58                 active_apps_priority_list.remove(app)
59
60                 # Add the copied app back into the history at the beginning of the list
61                 app_copy['total_active_time'] += current_time - app.get('last_update_time', current_time)
62                 app_copy['last_update_time'] = current_time
63                 active_apps_priority_list.insert(0, app_copy)
64
65         updated = True

```

Listing 1: Python code for the background service that generates a priority list of active data analysis applications

Appendix 2 APIs Prompts

1. Filter (MATLAB/Text-Editor)	
Prefix	<p>I want you to act as SQLite3 query writer</p> <p>Table MyDataTable, columns = ['Product Category] as str', '[Region] as str', '[Year] as int', '[Sales] as str', '[Revenue] as float', '[Profit Margin] as float', '[Sales Channel] as str', '[Payment Method] as str', '[Advertising Spend] as int', '[Customer Segment] as str', '[Customer Lifetime Value] as int', '[Sales Rep] as str']</p> <p>Here are some Examples</p> <p>Q: What are the Total Sales in the Table.</p> <p>Ans: SELECT SUM([Sales]) FROM MyDataTable;</p> <p>Q: What is the highest Sales Channel in Table.</p> <p>Ans: SELECT [Sales Channel] FROM MyDataTable ORDER BY [Sales] DESC LIMIT 1;</p> <p>Q: The total sales in the Sales Table from the region North and south.</p> <p>Ans: SELECT SUM([Sales]) FROM MyDataTable WHERE [Region] = 'South' OR [Region] = 'North'</p> <p>Q: In which year are the lowest Sales.</p> <p>Ans: SELECT [Year], MIN([Sales]) FROM MyDataTable GROUP BY [Year] ORDER BY MIN([Sales]) ASC LIMIT 1;</p> <p>Q: Select all rows where Sales Rep is Sarah.</p> <p>Ans: SELECT * FROM MyDataTable WHERE UPPER([Sales Rep]) LIKE "%SARAH";</p>
Table Schema	<p>Table MyDataTable2, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float']</p>
User Query	<p>Select only Canadian "Male" Players with less than 25 years</p>
Suffix	<p>Ans:</p>
LLM Response	<p>SELECT AVG([Expenditure]) FROM olympic.medals</p>

Summary of Olympic Medal Table:

The Olympic Medal Table contains data about athletes' names, sex, age, country, sport, year, and the number of bronze, gold, silver, and total medals won by each athlete. The table is used to analyze and compare the performance of athletes from different countries, sports, and years in terms of medal count.

Key Findings:

1. The total number of athletes in the table is 8668.
2. The average age of athletes in the table is 26.75.
3. The country with the highest number of gold medals is 'United States'.
4. The sport with the highest number of gold medals is 'Swimming'.
5. The year with the highest number of gold medals is 2016.
6. The country with the highest total number of medals is 'United States', 'Russia'.
7. The sport with the highest total number of medals is 'Swimming', 'Gymnastics'.
8. The year with the highest total number of medals is 2016, 2000.
9. The average number of bronze medals won by athletes is 0.39.
10. The average number of gold medals won by athletes is 0.32.
11. The average number of silver medals won by athletes is 0.36.
12. The country with the highest average age of athletes is 'Canada'.
13. The sport with the highest average age of athletes is 'Equestrianism'.
14. The year with the highest average age of athletes is 2008.
15. The country with the highest number of female athletes is .

2. Analysis

Prefix	<p>I want you to act as a Data Analyst Writing Summary and Key Finding using SQLite3 query writer</p> <p>Table MyDataTable, columns = ['[Product Category] as str', '[Region] as str', '[Year] as int', '[Sales] as str', '[Revenue] as float', '[Profit Margin] as float', '[Sales Channel] as str', '[Payment Method] as str', '[Advertising Spend] as int', '[Customer Segment] as str', '[Customer Lifetime Value] as int', '[Sales Rep] as str']</p> <p>Summary of Sales Table:</p> <p>Sales Table contains data about product categories, regions, years, sales, revenue, profit margin, sales channel, payment method, advertising spend, customer segment, customer lifetime value, and sales rep. The table is used to analyze and compare the performance of different sales channels, payment methods, customer segments, and sales reps. The data can also be used to identify trends in sales, revenue, profit margin, and customer lifetime value.</p> <p>Key Findings:</p> <ol style="list-style-type: none"> Total Sales in the Sales Table are {TotalSales__} and Revenue is {Revenue__} and Profit margin is {ProfitMargin__}. TotalSales__ = SELECT SUM([Sales]) FROM MyDataTable; Revenue__ = SELECT SUM([Revenue]) FROM MyDataTable; ProfitMargin__ = SELECT AVG([Profit Margin]) FROM MyDataTable; The highest Sales Channel in Sales Table is {TopSalesChannel__} and Payment Method is {TopPaymentMethod__}. TopSalesChannel__ = SELECT [Sales Channel] FROM MyDataTable ORDER BY [Sales] DESC LIMIT 1; TopPaymentMethod__ = SELECT [Payment Method] FROM MyDataTable ORDER BY [Sales] DESC LIMIT 1; The total sales in the Sales Table from the region {Region__} are {TotalSalesFromRegion__}. Region__ = SELECT [Region] FROM MyDataTable ORDER BY [Sales] DESC LIMIT 2; TotalSalesFromRegion__ = SELECT SUM([Sales]) FROM MyDataTable WHERE [Region] = (SELECT [Region] FROM MyDataTable ORDER BY [Sales] DESC LIMIT 2); The most common Product Category in Sales Table is {MostCommonProductCategory__}. MostCommonProductCategory__ = SELECT [Product Category], COUNT([Product Category]) FROM MyDataTable GROUP BY ([Product Category]) ORDER BY COUNT([Product Category]) DESC LIMIT 2;
Table Schema	Table MyDataTable2, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float']
User Query	Analyze the table with at least 15 Key findings?
Suffix	Write the Summary and Generate template for analysis of MyDataTable2, Example Template Format must have blanked entries in the Python String Formatter and SQLite3 queries.
LLM Response	<p>Summary of Olympic Medal Table: The Olympic Medal Table contains data about athletes' names, sex, age, country, sport, year, and the number of bronze, gold, silver, and total medals won. The table is used to analyze and compare the performance of athletes from different countries, sports, and years in terms of medal count. The data can also be used to identify trends in medal distribution and the success of individual athletes.</p> <p>Key Findings:</p> <p>TotalAthletes__ = SELECT COUNT(DISTINCT [Name]) FROM MyDataTable2;</p> <p>The total number of athletes in the table is {TotalAthletes__}.</p> <p>TotalMedals__ = SELECT SUM([Total Medal]) FROM MyDataTable2;</p> <p>The total number of medals won in the table is {TotalMedals__}.</p> <p>TopGoldCountry__ = SELECT [Country] FROM MyDataTable2 WHERE [Gold Medal] = (SELECT MAX([Gold Medal]) FROM MyDataTable2);</p> <p>The country with the highest number of gold medals is {TopGoldCountry__}.</p> <p>TopGoldSport__ = SELECT [Sport] FROM MyDataTable2 WHERE [Gold Medal] = (SELECT MAX([Gold Medal]) FROM MyDataTable2);</p> <p>The sport with the highest number of gold medals is {TopGoldSport__}.</p> <p>AverageAge__ = SELECT AVG([Age]) FROM MyDataTable2;</p> <p>The average age of athletes in the table is {AverageAge__}.</p>

3. Plot

Prefix	<pre> 1 I want you to act as Python Data Visualization Analyst using matplotlib 2 Table MyDataTable, columns = ['[Product Category] as str', '[Region] as str', '[Year] as int', '[Sales] as str', '[Revenue] as float', '[Profit Margin] as float', '[Sales Channel] as str', '[Payment Method] as str', '[Advertising Spend] as int', '[Customer Segment] as str', '[Customer Lifetime Value] as int', '[Sales Rep] as str'] 3 Here are some Examples 4 1. Line chart showing Revenue by Year: 5 fig, ax = plt.subplots(figsize=(10, 6)) 6 grouped = df_plot.groupby('Year')['Revenue'].sum() 7 ax.plot(grouped.index, grouped.values) 8 ax.set_xlabel('Year') 9 ax.set_ylabel('Revenue') 10 ax.set_title('Revenue by Year') 11 plt.xticks(rotation=45) 12 ax.spines['top'].set_visible(False) 13 ax.spines['right'].set_visible(False) 14 plt.show() 15 2. Stacked bar chart showing Revenue and Advertising Spend by Product Category: 16 fig, ax = plt.subplots(figsize=(10, 6)) 17 grouped = df_plot.groupby('Product Category')[['Revenue', 'Advertising Spend']].sum() 18 grouped.plot(kind='bar', stacked=True, ax=ax) 19 ax.set_xlabel('Product Category') 20 ax.set_ylabel('Amount') 21 ax.set_title('Revenue and Advertising Spend by Product Category') 22 plt.xticks(rotation=45) 23 ax.spines['top'].set_visible(False) 24 ax.spines['right'].set_visible(False) 25 plt.show() 26 3. Scatter plot showing Revenue vs. Advertising Spend by Sales Channel: 27 fig, ax = plt.subplots(figsize=(10, 6)) 28 grouped = df_plot[['Sales Channel', 'Revenue', 'Advertising Spend']].groupby('Sales Channel') 29 for name, group in grouped: 30 ax.scatter(group['Advertising Spend'], group['Revenue'], label=name) 31 ax.set_xlabel('Advertising Spend') 32 ax.set_ylabel('Revenue') 33 ax.set_title('Revenue vs. Advertising Spend by Sales Channel') 34 ax.legend() 35 ax.spines['top'].set_visible(False) 36 ax.spines['right'].set_visible(False) 37 plt.show() </pre>
Table Schema	<pre> Table MyDataTable2, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float'] </pre>
User Query	Plot bar chart of Normalized data Worldwide Gross, Production Budget ratings Vs Genre
Suffix	<pre> 1 Label the x and y axes appropriately. 2 Add a title. Set the fig supitle as empty. 3 import pandas as pd 4 import matplotlib.pyplot as plt 5 df_plot=pd.read_csv('MyDataTable2.csv') 6 fig, ax=plt.subplots(1,1,figsize=(10,4)) </pre>
LLM Response	<pre> grouped = df_plot.groupby('Country')[['Gold Medal', 'Silver Medal', 'Bronze Medal']].sum(). ↪ nlargest(10, 'Gold Medal') grouped.plot(kind='bar', ax=ax) ax.set_xlabel('Country') ax.set_ylabel('Number of Medals') ax.set_title('Gold, Silver, and Bronze Medals vs Top 10 Countries') ax.set_xticklabels(grouped.index, rotation=45) ax.spines['top'].set_visible(False) ax.spines['right'].set_visible(False) fig.suptitle('') </pre>

4. Format

Prefix

```

1 Act as Python Expert programmer that automates MS Excel using xlwings Version 0.29.1
2 "Sales" worksheet has columns = ['[Product Category] as str', '[Region] as str', '[Year] as int',
   '[Sales] as str', '[Revenue] as float', '[Profit Margin] as float', '[Sales Channel] as str',
   '[Payment Method] as str', '[Advertising Spend] as int', '[Customer Segment] as str',
   '[Customer Lifetime Value] as int', '[Sales Rep] as str']'.
3 Here's are a few examples:
4 1. Format the years greater than 2010
5 import xlwings as xw
6 from xlwings.constants import FormatConditionType, FormatConditionOperator
7 active_wb = xw.books.active, active_ws = active_wb.sheets.active
8 year_col = next((active_ws.range(f"{cell.address.split('$')[1]}1").expand('down') for cell in
   ↳ active_ws.range("1:1").expand('right') if cell.value == "Year"), False)
9 if year_col:
10     col = year_col.address.split('$')[1]
11     exp = f'=${col}2 > 2010'
12     data_range = active_ws.range("A2").expand()
13     data_range.api.FormatConditions.Add(FormatConditionType.xlExpression, "", exp)
14     data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Interior.
   ↳ ColorIndex = 6
15     data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Font.Bold =
   ↳ True
16 else:
17     print("Column header 'Year' not found in worksheet.")
18 2. Highlight the Profit Margin less than 0.1
19 import xlwings as xw
20 from xlwings.constants import FormatConditionType, FormatConditionOperator
21 active_wb = xw.books.active, active_ws = active_wb.sheets.active
22 profit_margin_col = next((active_ws.range(f"{cell.address.split('$')[1]}1").expand('down') for
   ↳ cell in active_ws.range("1:1").expand('right') if cell.value == "Profit Margin"),
   ↳ False)
23 if profit_margin_col:
24     col = profit_margin_col.address.split('$')[1]
25     exp = f'=${col}2 < 0.1'
26     data_range = active_ws.range("A2").expand()
27     data_range.api.FormatConditions.Add(FormatConditionType.xlExpression, "", exp)
28     data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Interior.
   ↳ ColorIndex = 3
29     data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Font.Bold =
   ↳ True
30 else:
31     print("Column header 'Profit Margin' not found in worksheet.")

```

4. Format (Continue)

Prefix	<pre> 1 3. Format North from Region 2 import xlwings as xw 3 from xlwings.constants import FormatConditionType, FormatConditionOperator 4 active_wb = xw.books.active, active_ws = active_wb.sheets.active 5 region_col = next((active_ws.range(f'{cell.address.split('\$')[1]}1').expand('down') for cell in 6 ↳ active_ws.range("1:1").expand('right') if cell.value == "Region"), False) 7 if region_col: 8 col = region_col.address.split('\$')[1] 9 exp = f'={\$col}2 = "North"' 10 data_range = active_ws.range("A2").expand() 11 data_range.api.FormatConditions.Add(FormatConditionType.xlExpression, "", exp) 12 data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Interior. 13 ↳ ColorIndex = 3 14 data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Font.Bold = 15 ↳ True 16 else: 17 print("Column header 'Region' not found in worksheet.") </pre>
Table Schema	<pre> Table MyDataTable2, columns = ['Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float'] </pre>
User Query	Format female players those play Gymnastics and from United States
Suffix	<pre> 1 import xlwings as xw 2 from xlwings.constants import FormatConditionType, FormatConditionOperator 3 active_wb = xw.books.active 4 active_ws = active_wb.sheets.active </pre>
LLM Response	<pre> sex_col = next((active_ws.range(f'{cell.address.split('\$')[1]}1').expand('down') for cell in ↳ active_ws.range("1:1").expand('right') if cell.value == "Sex"), False) sport_col = next((active_ws.range(f'{cell.address.split('\$')[1]}1').expand('down') for cell in ↳ active_ws.range("1:1").expand('right') if cell.value == "Sport"), False) country_col = next((active_ws.range(f'{cell.address.split('\$')[1]}1').expand('down') for cell in ↳ active_ws.range("1:1").expand('right') if cell.value == "Country"), False) if sex_col and sport_col and country_col: sex_col = sex_col.address.split('\$')[1] sport_col = sport_col.address.split('\$')[1] country_col = country_col.address.split('\$')[1] exp = f'={AND}({sex_col}2 = "F", {sport_col}2 = "Gymnastics", {country_col}2 = " ↳ United States")' data_range = active_ws.range("A2").expand() data_range.api.FormatConditions.Add(FormatConditionType.xlExpression, "", exp) data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Interior. ↳ ColorIndex = 4 data_range.api.FormatConditions(data_range.api.FormatConditions.Count).Font.Bold = ↳ True else: print("Column header 'Sex', 'Sport', or 'Country' not found in worksheet.") </pre>

5. Sort

Prefix	<pre> 1 Act as Python Expert programmer that automates MS Excel using xlwings Version 0.29.1 2 "Sales" worksheet has columns = ['[Product Category] as str', '[Region] as str', '[Year] as int', '[Sales] as str', '[Revenue] as float', '[Profit Margin] as float', '[Sales Channel] as str', '[Payment Method] as str', '[Advertising Spend] as int', '[Customer Segment] as str', '[Customer Lifetime Value] as int', '[Sales Rep] as str']'. 3 Here's are a few examples: 4 1. Sort selected column in ascending order 5 import xlwings as xw 6 active_wb = xw.books.active 7 active_ws = active_wb.sheets.active 8 selected_cell = st.session_state.selected_cell 9 column_to_sort = selected_cell.expand('down') 10 if column_to_sort: 11 data_range = active_ws.range("A2").expand() 12 data_range.api.Sort(Key1=column_to_sort.api, Order1=1, Header=2, Orientation=1) 13 else: 14 print("No Column selected.") 15 pass 16 2. Sort by Year 17 import xlwings as xw 18 active_wb = xw.books.active 19 active_ws = active_wb.sheets.active 20 column_to_sort = next((active_ws.range(f"{cell.address.split('\$')[1]}1").expand('down') for ↪ cell in active_ws.range("1:1").expand('right') if cell.value == "Year"), False) 21 if column_to_sort: 22 data_range = active_ws.range("A2").expand() 23 data_range.api.Sort(Key1=column_to_sort.api, Order1=1, Header=2, Orientation=1) 24 else: 25 print("Column header 'Year' not found in worksheet.") 26 pass </pre>
Table Schema	<pre> Table MyDataTable2, columns = ['[Name] as str', '[Sex] as str', '[Age] as float', '[Country] as str', '[Sport] as str', '[Year] as float', '[Bronze Medals] as float', '[Gold Medals] as float', '[Silver Medals] as float', '[Total Medals] as float'] </pre>
User Query	Sort by Sport in descending order
Suffix	<pre> 1 import xlwings as xw 2 active_wb = xw.books.active 3 active_ws = active_wb.sheets.active </pre>
LLM Response	<pre> column_to_sort = next((active_ws.range(f"{cell.address.split('\$')[1]}1").expand('down') for ↪ cell in active_ws.range("1:1").expand('right') if cell.value == "Sport"), False) if column_to_sort: data_range = active_ws.range("A2").expand() data_range.api.Sort(Key1=column_to_sort.api, Order1=2, Header=2, Orientation=1) else: print("Column header 'Sport' not found in worksheet.") </pre>

Fig. 7 Output Report for User Query: “Analyze the table with at least 15 Key findings”, While user was working on “Olympic_medal” Table from the NL4DV dataset

Fields	Mean	Median	Mode	Range	Min	Max	Sum	Count	Sample.Var	Std.Dev	Std.Err	Kurtosis	Skew
Age	26.75	26.0	26	46	15	61	231864	8668	26.38	5.14	0.06	2.15	0.91
Year	2008.63	2008.0	2016	16	2000	2016	17410780	8668	31.66	5.63	0.06	-1.31	-0.14
Bronze Medal	0.39	0.0	0	3	0	3	3422	8668	0.26	0.51	0.01	-1.05	0.64
Gold Medal	0.32	0.0	0	5	0	5	2778	8668	0.26	0.51	0.01	2.3	1.39
Silver Medal	0.36	0.0	0	3	0	3	3159	8668	0.26	0.51	0.01	-0.22	0.89
Total Medal	1.08	1.0	1	5	1	6	9359	8668	0.12	0.34	0.0	40.63	5.56

Acknowledgements This work is supported by the Natural Science Foundation of China under Grant No. 62132010, and by Beijing Key Lab of Networked Multimedia, the Institute for Guo Qiang, Tsinghua University, Institute for Artificial Intelligence, Tsinghua University (THUAI), and by 2025 Key Technological Innovation Program of Ningbo City under Grant No. 2022Z080. Additionally, we acknowledge the support provided by the Beijing Municipal Science & Technology Commission and the Administrative Commission of Zhongguancun Science Park under Grant No. Z221100006722018.

Code availability The prompts substantiating the conclusions can be found in the appendix section of the manuscript. Additionally, the corresponding author will furnish the complete source code upon receiving a justified request.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

- Bach, J., Bolton, M.: A Context-Driven Approach to Automation in Testing. Technical report, Satisfice, Inc., Feb. 2016. Available at <https://shorturl.at/cORT9> (2016)
- Bazire, M., Brézillon, P.: Understanding context before using it. In: Modeling and Using Context: 5th International and Interdisciplinary Conference CONTEXT 2005, Paris, France, . Proceedings 5, pp. 29–40. Springer (2005)
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A.: Language models are few-shot learners. *Adv. Neural Inform. Process. Syst.* **33**, 1877–1901 (2020)
- Chase, H.: Langchain (2023). <https://python.langchain.com/en/latest/index.html>
- Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G.: Evaluating large language models trained on code. arXiv preprint <https://arxiv.org/abs/2107.03374> (2021)
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S.: Palm: Scaling language modeling with pathways. arXiv preprint <https://arxiv.org/abs/2204.02311> (2022)
- Christiano, P.F., Leike, J., Brown, T., Martic, M., Legg, S., Amodei, D.: Deep Reinforcement Learning from Human Preferences. Curran Associates, Inc. (2017). https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf
- Chung, H.W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S.: Scaling instruction-finetuned language models. arXiv preprint <https://arxiv.org/abs/2210.11416> (2022)
- Corporation, M.: Microsoft excel (2019 (16.0)). <https://office.microsoft.com/excel>
- Coutaz, J., Crowley, J.L., Dobson, S., Garlan, D.: Context is key. *Commun. of the ACM* **48**(3), 49–53 (2005)
- Diederich, S., Brendel, A.B., Morana, S., Kolbe, L.: On the design of and interaction with conversational agents: an organizing and assessing review of human-computer interaction research. *J. Assoc. Inf. Syst.* **23**(1), 96–138 (2022)
- Ding, J., Zhao, B., Huang, Y., Wang, Y., Shi, Y.: Gazereader: Detecting unknown word using webcam for english as a second language (esl) learners. In: Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–7
- Feng, Yingchaojie., Wang, Xingbo., Pan, Bo., Wong, Kam Kwai., Ren, Yi., Liu, Shi., Yan, Zihan., Ma, Yu xin., Qu, Huamin., Chen, Wei.: eng 2023/04/07 IEEE Trans Vis Comput Graph. 2023 Jan 26;PP. <https://doi.org/10.1109/TVCG.2023.3240003>
- Feng, Y., Wang, X., Pan, B., Wong, K.K., Ren, Y., Liu, S., Yan, Z., Ma, Y., Qu, H., Chen, W.: Explaining and diagnosing nli-based visual data analysis. *IEEE Trans. Vis. Comput. Graph.* (2023). <https://doi.org/10.1109/TVCG.2023.3240003>
- Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., Neubig, G.: Pal: Program-aided language models. arXiv preprint <https://arxiv.org/abs/2211.10435> (2022)
- Gauselmann, P., Runge, Y., Jilek, C., Frings, C., Maus, H., Tempel, T.: A relief from mental overload in a digitalized world: How context-sensitive user interfaces can enhance cognitive performance. *Int. J. Human-Comput. Interact.* **39**(1), 140–150 (2023)
- Greenberg, S.: Context as a dynamic construct. *Human-Comput. Interact.* **16**(2–4), 257–268 (2001)
- Hoque, E., Kavehzadeh, P., Masry, A.: Chart question answering: state of the art and future directions. *Comput. Graphics Forum* **41**(3), 555–572 (2022). <https://doi.org/10.1111/cgf.14573>
- Jiang, Z., Xu, F.F., Araki, J., Neubig, G.: How can we know what language models know? *Trans. Assoc. Comput. Ling.* **8**, 423–438 (2020)
- Joshi, H., Ebenezer, A., Cambronero, J., Gulwani, S., Kanade, A., Le, V., Radiček, I., Verbruggen, G.: Flame: A small language model for spreadsheet formulas. arXiv preprint <https://arxiv.org/abs/2301.13779> (2023)
- Karaman, Ç.Ç., Sezgin, T.M.: Gaze-based predictive user interfaces: visualizing user intentions in the presence of uncertainty. *Int. J. Human-Comput. Stud.* **111**, 78–91 (2018). <https://doi.org/10.1016/j.ijhcs.2017.11.005>
- Khatry, A., Cahoon, J., Henkel, J., Deep, S., Emani, V., Floratou, A., Gulwani, S., Le, V., Raza, M., Shi, S.: From words to code: Harnessing data for program synthesis from natural language. arXiv preprint <https://arxiv.org/abs/2305.01598> (2023)
- Kumar, S., Talukdar, P.: Reordering examples helps during priming-based few-shot learning. arXiv preprint <https://arxiv.org/abs/2106.01751> (2021)

- Lazaridou, A., Gribovskaya, E., Stokowiec, W., Grigorev, N.: Internet-augmented language models through few-shot prompting for open-domain question answering. arXiv preprint <https://arxiv.org/abs/2203.05115> (2022)
- Liang, Y., Wu, C., Song, T., Wu, W., Xia, Y., Liu, Y., Ou, Y., Lu, S., Ji, L., Mao, S.: Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. arXiv preprint <https://arxiv.org/abs/2303.16434> (2023)
- Liu, M.X., Sarkar, A., Negreanu, C., Zorn, B., Williams, J., Toronto, N., Gordon, A.D.: “what it wants me to say”: Bridging the abstraction gap between end-user programmers and code-generating large language models. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–31
- Liu, C., Han, Y., Jiang, R., Yuan, X.: ADVISor: automatic visualization answer for natural-language question on tabular data. Pac. Vis. Symp. (2021). <https://doi.org/10.1109/PacificVis52677.2021.00010>
- Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. ACM Comput. Surv. **55**(9), 1–35 (2023)
- Lu, Y., Bartolo, M., Moore, A., Riedel, S., Stenetorp, P.: Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. arXiv preprint <https://arxiv.org/abs/2104.08786> (2021)
- Luo, Y., Tang, N., Li, G., Chai, C., Li, W., Qin, X.: Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks (2021). [10.1145/3448016.3457261](https://doi.org/10.1145/3448016.3457261)
- Luo, Y., Tang, N., Li, G., Tang, J., Chai, C., Qin, X.: Natural language to visualization by neural machine translation. IEEE Trans. Vis. Comput. Graph. **28**(1), 217–226 (2022). <https://doi.org/10.1109/TVCG.2021.3114848>
- Luo, Yuyu, Tang, Nan, Li, Guoliang, Tang, Jiawei, Chai, Chengliang, Qin, Xuedi: Eng Research Support, Non-U.S. Gov’t 2021/11/17. IEEE Trans. Vis. Comput. Graph. **28**(1), 217–226 (2022). <https://doi.org/10.1109/TVCG.2021.3114848>
- Maddigan, P., Susnjak, T.: Chat2vis: Fine-tuning data visualisations using multilingual natural language text and pre-trained large language models. arXiv preprint <https://arxiv.org/abs/2303.14292> (2023)
- Maddigan, P., Susnjak, T.: Chat2vis: Generating data visualisations via natural language using chatgpt, codex and gpt-3 large language models. arXiv preprint <https://arxiv.org/abs/2302.02094> (2023)
- Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., Zettlemoyer, L.: Rethinking the role of demonstrations: What makes in-context learning work? arXiv preprint <https://arxiv.org/abs/2202.12837> (2022)
- Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W.: Webgpt: Browser-assisted question-answering with human feedback. arXiv preprint <https://arxiv.org/abs/2112.09332> (2021)
- Narechania, A., Srinivasan, A., Stasko, J.: NI4dv: a toolkit for generating analytic specifications for data visualization from natural language queries. IEEE Trans. Vis. Comput. Graph. **27**(2), 369–379 (2021). <https://doi.org/10.1109/TVCG.2020.3030378>
- Narechania, Arpit, Srinivasan, Arjun, Stasko, John: Eng research support , U.S. Gov’t, Non-P.H.S. 2020/10/14. IEEE Trans. Vis. Comput. Graph. **27**(2), 369–379 (2021). <https://doi.org/10.1109/TVCG.2020.3030378>
- Ni, A., Iyer, S., Radev, D., Stoyanov, V., Yih, W.-t., Wang, S.I., Lin, X.V.: Lever: Learning to verify language-to-code generation with execution. arXiv preprint <https://arxiv.org/abs/2302.08468> (2023)
- OpenAI: Code interpreter-chatgpt plugins (July 2023). <https://openai.com/blog/chatgpt-plugins>
- OpenAI: Gpt-4 technical report. arXiv preprint <https://arxiv.org/abs/2303.08774> (2023)
- (Open Source): Welcome to streamlit (2023 (1.18.1)). <https://github.com/streamlit/streamlit>
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A.: Training language models to follow instructions with human feedback. Adv. Neural Inform. Process. Syst. **35**, 27730–27744 (2022)
- Rath, A.S., Devaurs, D., Lindstaedt, S.N.: Uico (2009). <https://doi.org/10.1145/1552262.1552270>
- (Open Source): xlwings - make excel fly with python (2023 (0.29.1)).
- Setlur, V., Battersby, S.E., Tory, M., Gossweiler, R., Chang, A.X.: Eviza (2016). <https://doi.org/10.1145/2984511.2984588>
- Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N.A., Lewis, M.: Measuring and narrowing the compositionality gap in language models. arXiv preprint <https://arxiv.org/abs/2210.03350> (2022)
- Setlur, V., Tory, M.: How do you converse with an analytical chatbot? revisiting gricean maxims for designing analytical conversational behavior. In: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, pp. 1–17
- Shen, L., Shen, E., Luo, Y., Yang, X., Hu, X., Zhang, X., Tai, Z., Wang, J.: Towards natural language interfaces for data visualization: A survey. arXiv preprint <https://arxiv.org/abs/2109.03506> (2021)
- Srinivasa Ragavan, S., Hou, Z., Wang, Y., Gordon, A.D., Zhang, H., Zhang, D.: GridBook: Natural Language Formulas for the Spreadsheet Grid (2022). <https://doi.org/10.1145/3490099.3511161>
- Srinivasan, A., Nyapathy, N., Lee, B., Drucker, S.M., Stasko, J.: Collecting and Characterizing Natural Language Utterances for Specifying Data Visualizations (2021). <https://doi.org/10.1145/3411764.3445400>
- Stumpf, S., Bao, X., Dragunov, A., Dietterich, T.G., Herlocker, J., Johnsrude, K., Li, L., Shen, J.: The tasktracker system. In: PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, vol. 20, p. 1712. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999
- Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H.W., Chowdhery, A., Le, Q.V., Chi, E.H., Zhou, D.: Challenging big-bench tasks and whether chain-of-thought can solve them. arXiv preprint <https://arxiv.org/abs/2210.09261> (2022)
- Tang, J., Luo, Y., Ouzzani, M., Li, G., Chen, H.: Sevi: Speech-to-Visualization through Neural Machine Translation (2022). <https://doi.org/10.1145/3514221.3520150>
- Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V., Stojnic, R.: Galactica: A large language model for science. arXiv preprint <https://arxiv.org/abs/2211.09085> (2022)
- team, T.: Pandas for python (2023) <https://doi.org/10.5281/zenodo.10304236>
- The MathWorks, I.: Matlab engine for python (2022). https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html
- The MathWorks, I.: Matlab version: 9.13.0 (r2022b) (2022). <https://www.mathworks.com>
- Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y.: Lamda: Language models for dialog applications. arXiv preprint <https://arxiv.org/abs/2201.08239> (2022)
- Van Binsbergen, L.T., Verano Merino, M., Jeanjean, P., Van Der Storm, T., Combemale, B., Barais, O.: A principled approach to repl interpreters. In: Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, pp. 84–100 (2020)
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Adv. Neural Inform. Process. syst. **30** (2017)

- Vuong, T., Jacucci, G., Ruotsalo, T.: Watching inside the screen: digital activity monitoring for task recognition and proactive information retrieval. *Proceed. ACM Interact. Mob. Wear. Ubiquitous Technol.* **1**(3), 1–23 (2017)
- Wang, Yun., Hou, Zhitao., Shen, Leixian., Wu, Tongshuang., Wang, Jiaqi., Huang, He., Zhang, Haidong., Zhang, Dongmei. 2023 Eng 2022/10/06 *IEEE Trans Vis Comput Graph.* **29**(1):1222–1232. 10.1109/TVCG.2022.3209357
- Wang, Y., Hou, Z., Shen, L., Wu, T., Wang, J., Huang, H., Zhang, H., Zhang, D.: Towards natural language-based visualization authoring. *IEEE Trans. Vis. Comput. Graph.* **29**(1), 1222–1232 (2023). <https://doi.org/10.1109/TVCG.2022.3209357>
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D.: Emergent abilities of large language models. *arXiv preprint* <https://arxiv.org/abs/2206.07682> (2022)
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., Zhou, D.: Chain of thought prompting elicits reasoning in large language models. *arXiv preprint* <https://arxiv.org/abs/2201.11903> (2022)
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. *arXiv preprint* <https://arxiv.org/abs/2210.03629> (2022)
- Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., Chi, E.: Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint* <https://arxiv.org/abs/2205.10625> (2022)

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.



Aamir Khan Jadoon is a PhD candidate in Advanced Computing at Tsinghua University, specializing in Human-Computer Interaction, Collaborative Human-AI Systems, and Context-Aware Computing. With over 20 years of industry experience as a software analyst and programmer, he bridges the gap between academia and practical application, focusing on Adaptive AI for personalized task automation. Jadoon holds a Master's degree in Advanced Computing from Tsinghua

University.



Chun Yu, an Associate Professor at Tsinghua University since obtaining his Ph.D. in 2012, is a prominent figure in Human-Computer Interaction (HCI). His research focuses on computational models and algorithms for enhancing multi-modal user interfaces across mobile, large displays, and VR/AR platforms. With over 80 publications in leading HCI forums like ACM CHI, UIST, and Ubicomp, Chun Yu has been recognized with 14 awards, including multiple ACM CHI Honorable Mention Awards between 2013 and 2023, the Chinese CHI 2015 Best Paper Award, IEEE UIC 2014 Best Paper Award, and ACM MobileHCI 2013 Honorable Mention Award. His work, adopted by industry leaders and reaching over 600 million users, showcases the practical impact of his research on widely-used commercial products.



Yuanchun Shi received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University, China. She joined the faculty of Tsinghua University, in 1993, where she is currently a Changjiang Distinguished Professor with the Department of Computer Science and Technology. She has published more than 100 articles in top conferences and journals (CS Ranking, HCI), including over 30 CHI, UIST, Ubicomp, TOCHI, and IHCS papers over the most recent five years alone.

Her research interests include human-computer interaction, ubiquitous computing, and multimedia, which has won many academic awards, including two State Science and Technology Progress Awards, Faculty Awards, and many best paper and honorable mention awards from conferences, such as CHI and MobileHCI. Her recent HCI contributions include user behavior modeling, intelligent interaction techniques for mobile phones, large displays, and VR/AR headsets.