WaitGPT: Monitoring and Steering Conversational LLM Agent in Data Analysis with On-the-Fly Code Visualization

Liwenhan Xie[*]
liwenhan.xie@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR, China

Chengbo Zheng
cb.zheng@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR, China

Haijun Xia
haijunxia@ucsd.edu
University of California San Diego
La Jolla, CA, USA

Huamin Qu
huamin@ust.hk
Hong Kong University of Science and Technology
Hong Kong SAR, China

Chen Zhu-Tian
ztchen@umn.edu
University of Minnesota
Minneapolis, MN, USA

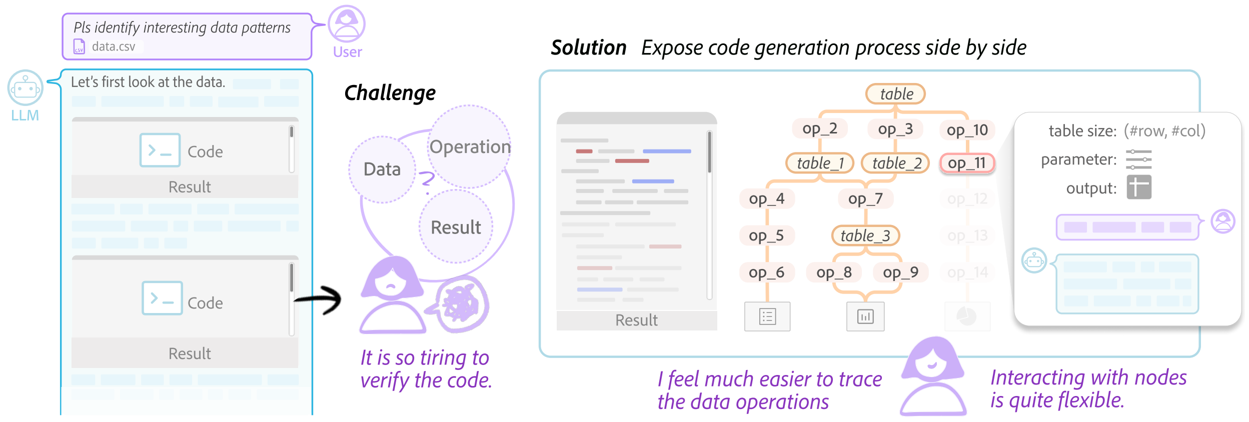

Pls identify interesting data patterns
data.csv
User
LLM
Let's first look at the data.
Code
Result
Code
Result
Challenge
Data
Operation
Result
It is so tiring to verify the code.
Solution Expose code generation process side by side
Result
table
op_2
op_3
op_10
table_1
table_2
op_11
op_4
op_7
op_12
op_5
table_3
op_13
op_6
op_8
op_9
op_14
table size: (#row, #col)
parameter:
output:
I feel much easier to trace the data operations
Interacting with nodes is quite flexible.

Figure 1: Monitoring and steering LLM-powered data analysis tools with WaitGPT: Beyond viewing the raw code, users can inspect data operations with a transformable representation generated on the fly and participate in data analysis proactively.

ABSTRACT

Large language models (LLMs) support data analysis through conversational user interfaces, as exemplified in OpenAI's ChatGPT (formally known as Advanced Data Analysis or Code Interpreter). Essentially, LLMs produce code for accomplishing diverse analysis tasks. However, presenting raw code can obscure the logic and hinder user verification. To empower users with enhanced comprehension and augmented control over analysis conducted by LLMs, we propose a novel approach to transform LLM-generated code into an interactive visual representation. In the approach, users are provided with a clear, step-by-step visualization of the LLM-generated code in real time, allowing them to understand, verify, and modify individual data operations in the analysis. Our design decisions are informed by a formative study (N=8) probing into user practice and challenges. We further developed a prototype named WaitGPT and conducted a user study (N=12) to evaluate its usability and effectiveness. The findings from the user study reveal that WaitGPT facilitates monitoring and steering of data analysis performed by LLMs, enabling participants to enhance error detection and increase their overall confidence in the results.

[*]This study was partially conducted during an academic visit to Harvard University.

CCS CONCEPTS

• **Human-centered computing → Natural language interfaces**; *Graphical user interfaces*; *Information visualization.*

KEYWORDS

Conversational Data Analysis, LLM Agent, Human-AI Interaction, Generative AI, Code Verification, Visual Programming

ACM Reference Format:
Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. WaitGPT: Monitoring and Steering Conversational LLM Agent in Data Analysis with On-the-Fly Code Visualization. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3654777.3676374

1 INTRODUCTION

Large language models (LLMs) have significantly lowered the entry point for data analysis, empowering users without strong programming skills to engage in sophisticated analytical tasks [8, 12, 23]. Instead of writing scripts or using complex software, people can directly talk to conversational LLM agents. Examples of emerging LLM-powered data analysis services or tools include ChatGPT Plus [47], Gemini Advanced [17], and CodeActAgent [65]. Generally, these tools follow a planning framework, where the LLM agent proposes a plan to divide the task, then generates code to process data and continues the process based on the execution result.

Despite their potential, real-world deployment of LLM-powered data analysis tools has exposed reliability concerns, including hallucinations [6, 33], subtle bugs [69, 73], and mismatch between LLM's understanding of the tasks and under-articulated user intents [32, 64]. Such shortcomings necessitate human oversight to verify and correct the data analysis process [9, 19, 46]. Current tools often present raw data analysis code, shifting the user's focus to low-level details instead of the high-level data analysis process. According to our interview with ChatGPT users, individuals, especially those with limited coding skills, struggle to comprehensively review the code produced by LLMs, thereby risking undetected errors and potentially incorrect results. Moreover, rectifying code through conversation can turn into a cumbersome exchange, adding to the inefficiency and frustration.

Our goal is to make the data analysis process conducted by LLMs easier to understand and navigate for users, in line with current research on designing UIs featuring generative AIs (e.g., [52, 57]). Specifically, we aim to support real-time monitoring and proactive intervention (steering) at any point. Compared with existing approaches targeting a traditional data analysis pipeline (e.g., [31, 55]), this scenario features conversational interaction and on-demand generation of unfamiliar code to the users, where the code streams in. Informed by a formative study involving 8 users experienced in LLM-powered data analysis, we propose a workflow that identifies data operations within the generated code and maps them to visual, interactive primitives on the fly (Figure. 1). These primitives collectively offer an overview of the data analysis process, and surface the details of each data operation and their internal runtime states in an intuitive, syntax-independent format. Furthermore, users can refine each operation by interacting directly with these primitives without regenerating the entire analysis code. Through this approach, we augment traditional conversational user interfaces (CUIs) with interactive visualization, transforming users from passive recipients of information into active participants in the data analysis task.

We have designed and implemented WaitGPT, a prototype system that converts the data analysis code generated by an LLM into a visual diagram that consists of nodes representing key data operations, composing an overview step by step. This diagram progressively evolves along with the code generation process. Furthermore, WaitGPT executes the underlying code line by line and updates the visual diagram to reflect the code's intermediate state during runtime. Users can interact with these nodes to modify or adjust the operations, thereby refining the data analysis process. Execution results are maintained and preserved within a sandbox environment, enabling the system to resume or rerun the analysis code after modifications, without the need to regenerate the entire code. A user study with 12 participants reported an enhanced experience, noting the ease of spotting errors, increased agency, and heightened confidence in the results produced by the LLM.

In summary, our contributions are three-fold.

- A formative study (N=8) that summarizes practices, challenges, and expectations in conducting data analysis with LLM agents based on conversation.
- A novel design that facilitates monitoring and steering LLM-generated data analysis script featuring interactive visualizations. We implement a prototype system named WaitGPT and evaluate its usability (N=12).
- Discussions and implications on user interface design of LLM agents for data analysis tasks.

2 BACKGROUND & RELATED WORK

Here, we review NLI-based data analysis tools, visualization techniques for data processing scripts, and user interface design for human-LLM interactions, which are closely related to our study.

2.1 Demystifying NLI-based Data Analysis

NLI-based data analysis tools interpret users' instructions in natural language and automatically perform analytic tasks. Existing tools often assemble atomic data operations based on a clear categorization of analytical tasks [53, 75]. To support more flexible user tasks, there has been surging interest in applying LLMs to translate NL-based user intents into data-related operations or directly synthesize visualization programs (e.g., [34, 35, 60]).

However, it remains unrealistic to expect completely correct outputs for reasons like language ambiguity and algorithmic or model accuracy [14, 15, 44]. This issue becomes more pronounced when integrating LLMs into data analysis tools, given their black-box nature. This characteristic calls for rigorous inspection and verification strategies, as highlighted in prior research [9, 18, 49]. Example errors include wrong column selection, data mapping, data transformation, etc. In response to the challenge, XNLI [14] provides a standalone interface that shows one user query to the key aspects in a finite set of the traditional NLI pipeline, i.e., attributes, tasks, and visual encodings. With LLMs, Huang et al. [25] converted the data transformation program into a flowchart using intermediate tables as nodes. Under a spreadsheet-based interface, Liu et al. [33] proposed *grounded abstraction matching* (GAM) that explains LLM-generated code to end users in natural language. ColDeco [15] further augments GAM with two complementary views of intermediate results, highlighting how the operation changes the result.

Our work applies to analytic tasks that are more open-ended and concern complex data operations, which is under-examined [23]. Most relevant to our interest in a conversational interface, Gu et al. [19] added a side panel that profiles intermediate data to facilitate retrospective examination of the synthesized code. Kazemitabaar et al. [28] proposed to afford editable assumptions, execution plans, and code in LLM response for close verification and steering. We complemented their design by proposing a transformable representation of the code, aiming to lower the abstraction level of the code and enhance user engagement during the interaction.

2.2 Sense-making of Data Processing Code

Simplifying data processing code can support learning [31], collaborative work [50], and quality control [56, 72]. To give a comprehensive view, prior research has condensed the operations into descriptive narratives [14, 33] or schematic diagrams [25, 51]. In addition, many works focused on visualizing interim results through animation (e.g., [21, 29, 50]) or a timeline representation (e.g., [2, 36, 45]). For instance, Datamation [50] visually maps and links each step of the data process to the underlying dataset, providing more context for the audience. Smallset Timeline [36] intelligently selects samples affected by the operation and encodes the changes on a table along the timeline.

To enhance understanding of atomic data operations, many works investigated step-wise examination of the underlying data. This can be achieved by revealing the connections and discrepancies between the input and output states. Pandas Tutor [31] highlights selected rows and links their new position with arrows. SOMNUS [72] presents 23 static glyphs for data transformation operations in table, column, and row granularity, respectively. To bridge the mental map between data transform specifications and results, some works allow interactive inspection [27, 55, 56]. For instance, Unravel [55] automatically transforms individual data operations into summary boxes with key parameters and the table size, which serves as an intermediate layer for users to modify and access runtime execution results.

WaitGPT addresses a new problem: sense-making of data processing code produced by an LLM agent. Compared to previous approaches that deal with complete and static scripts, the code is generated in a streaming manner, which may present challenges for users in terms of following the LLM's response during the generation process. In addition, some tools (e.g., [55, 63]) require coding proficiency while some have a rigid functionality (e.g., [14, 72]). However, in our scenario, end-users, including data analysts, laypeople, etc., talk to an LLM agent for various data analysis tasks. We prioritize intuitive visualization designs for immediate understanding and rapid verification, keeping users engaged and undistracted during the active code generation phase. General code debugging, however, is beyond our scope.

2.3 Advancing UIs for Human-LLM Interaction

Amidst the wave of LLMs, the HCI community has been advancing user interface design to enhance control over LLMs, moving beyond a standard chatbot framework or basic API invocations.

Similar to our motivation to facilitate easier comprehension and verification of the generated content, some works seek to bridge the gulf of envisioning in human-LLM interactions [57, 59]. For example, Graphlogue [26] converts linear text into a diagram that encodes logical structure on the fly to assist information-seeking tasks. Zhu-Tian et al. [76] foreshadows LLM-generated code incrementally and instantly during prompt crafting. Sensecape [58] empowers users with a multilevel abstraction of existing conversation and supports information foraging and sense-making. We attend to an emerging scenario of conversational data analysis with LLMs, where we present novel features like on-the-fly visualization as code streams in, code scrolly-telling, and snippet navigation.

Another stream of research explores novel interaction designs with LLMs that surpass the conventional single-text prompt, where more dynamic and progressive workflows and interaction modalities are promoted. For instance, Wu et al. [68] introduced the concept of AI Chains, where users specify how the output of one step becomes the input for the next, resulting in cumulative gains per step. Many works targeted specific application domains, including writing [10], graphics design [37], programming [1], etc. Relevant to our interest in granular control of LLM-generated code, Low-code LLM [4] allows users to edit the tentative workflow synthesized by a planning LLM, thereby providing control over the generated code. DynaVis [61] leverages LLM to synthesize UI widgets to edit data visualizations dynamically. Bearing a similar idea, our work supports user interactions with the intermediate visualization to drill down or refine the code in place for more intuitive and granular control with LLMs.

3 FORMATIVE STUDY

We conducted a formative study (N=8) to better understand the glitches in LLM-powered data analysis tools and inform the design considerations for contextualized support.

3.1 Setup

Recruitment & Screening. We posted recruitment advertisements on social media and university forums. Candidate participants were required to complete a questionnaire about their demographic information and relevant experience. We selected volunteers who are more experienced with data analysis and familiar with LLM-powered data analysis tools.

Protocol. The study consisted of a contextual inquiry (20∼40 min) and a structured interview (15 min). First, we asked participants to show their interaction history with LLM agents in data analysis tasks. If their original dataset is available, they will also walk the moderator through the data analysis procedure while thinking aloud. For five participants with the original dataset at hand, we asked them to replicate one analysis session directly while thinking aloud. The interview ended with a list of questions regarding the overall experience. Each participant is compensated with $12/hour.

Participants. We recruited 8 participants in total (P1–P8), with 3 females and 5 males, aged from 20 to 30. Specifically, there are 6 postgraduate students, 1 undergraduate student (P3), and 1 data journalist (P4). All are familiar with the data analysis mode (formally named as "Advanced Data Analysis" or "Code Interpreter") embedded in OpenAI's ChatGPT [47] and had at least 5 sessions.

Analysis. All interviews were video-recorded and transcribed into text. Following thematic analysis [3], the first author applied inductive and deductive approaches and derived initial categorized codes and themes. The first three authors reviewed transcripts and important screenshots based on weekly meetings to agree on the final themes after iterations.

3.2 Findings

Here, we summarize the key findings from the interview study.

Table 1: Common issues in the code generated by OpenAI's ChatGPT for data analysis tasks.

Issue Type	Detailed Behaviors of an LLM Agent
Incomplete workflow	Misses some important steps, e.g., not excluding empty value when computing means.
Non-existing symbols	Invoke a function, configure a parameter, or use a variable that is not defined.
Data transform failure	Fails to handle edge data value, e.g., accessing an attribute that does not exist in all data items.
Wrong columns	Selects the wrong column(s).
Unreasonable values	Sets parameter to an inappropriate value, e.g., using an overly high threshold for outliers.

3.2.1 Why do people turn to LLM-powered tools for data analysis? Participants recognized the versatility of conversational LLM agents for data analysis as a significant advantage. They have utilized it for a diversity of data-intensive tasks, including exploratory data analysis (4/8), data wrangling (4/8), confirmatory data analysis (2/8), data profiling (2/8), and data retrieval (1/8). In addition, participants appreciated its flexibility in open-ended data analysis. "*Compared with software with rigid functionalities, I enjoy the freedom here [in ChatGPT]. I can ask for an explanation based on the result, request recommendations for the next step, or insert irrelevant questions.*" (P6) Another strength of an LLM-powered data analysis tool is its low-code or no-code environment, where end users only need to describe the tasks and obtain a well-organized response in the form of code or report. For instance, P4, who works in investigative data journalism [54] and regularly cleans and organizes datasets from various sources, stated "*Having code generated from scratch saves days of my work*". This feature was particularly valued by participants who were not proficient in coding (2/8). "*I no longer need to care about detailed operations and learn the APIs.*" (P2)

3.2.2 How do people work with LLM-powered tools in data analysis? We categorize participants' workflows into three phases: code generation, post-verification, and iterative refinement.

By default, ChatGPT collapses the code and communicates the progress in percentage only. Correspondingly, participants (7/8) hardly toggled the code panel during the generation phase but distracted themselves by turning to personal matters or engaging in related side tasks like reviewing previous conversations.

Upon completion of the code generation, every participant consistently reviewed the textual response and, if available, the visualizations to grasp the analysis's implications. Verifying the code's reliability was a common concern, with most (6/8) participants inspecting the generated script, especially when the data insights were important. They would look into the entire data processing pipeline and specific parameters of individual operands. P4 sometimes posed a validation question to verify the code's correctness, such as requesting the mean value to see if it aligned with his prior knowledge. When the generated code was inconsistent with expectations, participants (6/8) attempted to recalibrate the agent's direction through refined prompts. P2 mentioned a special strategy: "*I try really hard to decompose the task into actionable items so that it won't be too challenging for ChatGPT.*" Notably, some participants (3/8) regenerated the response instead of starting a new conversation. "*I am afraid to break the analysis flow with additional requirements on a small step.*" (P3) For open-ended tasks, after obtaining initial results, participants may further drill down through conversation (3/8) or turn to a local coding environment (2/8), depending on the trade-off between coding and prompting. "*With the code, I can easily reuse it on a (computational) notebook.*" (P1)

3.2.3 What hinders human-LLM collaboration in data analysis tasks? Three themes emerge regarding glitches for users to participate in data analysis assisted by LLM agents actively.

◇ Disrupted workflow negatively impacts user engagement. As code generation and execution are sometimes long-winded, it interrupts the analysis flow. Most participants (7/8) would shift focus during the process instead of monitoring the generated code closely, for code is not as intuitive or accessible as natural language. "*I feel exhausted when reading the code, so I'd rather leave it alone.*" (P1) Without timely intervention, tiny errors in the code may propagate and invalidate the analysis result, precipitating a need to revisit and revise the work. This leads to heightened frustration and a considerable waste of time, as finishing one exploratory data analysis task generally takes half to three minutes. To avoid such prolonged dialogue exchanges, P3 explicitly requested the agent to ask for permission before generating and executing, explaining that "*(In this way,) I can at least take control over the direction*". (P5)

◇ Verifying raw code is mentally demanding. While LLMs may provide clear annotations to explain each step, many participants (7/8) still found verifying the generated code challenging.

On the one hand, reviewing the code snippet is inherently laborious and counter-intuitive, particularly when deciphering code from an external source, which can be mentally taxing. After all, LLMs may not follow the coding styles the participants are comfortable with. "*It [LLM] sometimes uses much-advanced syntax, so I ask it to write code like a freshman.*" (P5) Besides, LLMs may employ unfamiliar packages. "*I don't even know what the function parameter is about, let alone correct it.*" (P3)

On the other hand, LLMs may introduce various unexpected errors in the code that require careful inspection, as evidenced in the literature [9, 14, 18]. Table 1 lists example issues. P6 noted LLM hallucinations: "*At first look, the logic was awfully smooth, yet the parameter was a synthesized constant. It's very tricky (to identify the issue).*" Some participants (3/8) were concerned about the finding's reliability but frustrated with limited approaches. "*I am not sure if the conclusion is correct. I have a tight time budget, so I check the major steps and cross my fingers for no other issues.*" (P8)

◇ Iterations can be extensively back-and-forth. To fix identified issues, users need to formulate instructions regarding what is wrong and how to correct the errors and then wait for another generation-execution-report cycle. Unfortunately, this process can become time-consuming due to its trial-and-error nature and requires substantial effort to communicate the nuances of the desired analysis effectively. Therefore, many participants (6/8) were reluctant to embrace the conversational workflow fully. For minor issues like refining operational details, some participants (5/8) preferred to copy-paste the code to a local environment and make adaptations.

"*It is more convenient to reuse the code than telling ChatGPT specifically what to do.*" (P7) For major changes like adding a new processing step, they were more willing to communicate with the LLM agent since writing code becomes tedious. Still, after several trials, they would turn to the local environment when losing patience.

3.3 Design Considerations

Informed by the formative study, we draw the following design considerations (DC) to guide our conception of an alternative interaction design for LLM-powered data analysis tools. **Our design goal is to support monitoring and steering LLM-synthesized data analysis with interactive visual scaffolding**.

DC1. Abstract code stream into key data operations for a focused verification. In the context of LLM-based data analysis, a primary challenge emerges due to the often extensive and complex nature of the generated code. However, users usually prefer understanding the analysis process itself over the complex details of the code. Echoing a previous study [19], participants expressed the need to access data operations, determinant parameters, and their outcomes. To address this challenge, we propose to simplify the information to digest for verifying the data analysis procedure conducted by LLMs. By extracting the layered information concerning individual data operations from the code, such as the parametric specifications and execution results, we aim to refocus users' attention on the analysis process itself, sparing them from the overwhelming task of understanding the raw code.

DC2. Scaffold data operations and execution results through straightforward visualization generated on the fly. Despite the abstraction, users, particularly those with limited programming expertise, may still find it challenging to interpret the raw, syntax-heavy output produced by LLMs. Drawing inspiration from previous works in code visualization [42, 62], we adopt visual representations that abstract away from specific code syntax to facilitate quick comprehension of the data analysis process. Thus, the visual representation should also expose this information, including the data state before and after each operation. Moreover, this process should be executed on the fly along the code generation process, ensuring a seamless experience for the user aligning to their sensemaking process. It is also critical to establish a connection between the code and its visual representation. This will allow users to see the direct impact of their instructions on the data and to navigate the analysis workflow more effectively.

DC3. Support interrogation to the LLM and iterative code generation in the visualization. An outstanding issue of LLM-powered data analysis in a conversational interface is the tediousness of articulating refinement intents and uncertainties in LLMs' follow-up responses. To overcome this, the visual representations should simplify articulating these intents by providing mechanisms to modify the data analysis process at a granular level. Users should be able to interact with individual steps (data operations) of the generated analysis, allowing them to make precise adjustments without the need to rewrite large portions of code or restart the conversation. This granular control empowers users to fine-tune the analysis, accurately reflects their intentions, and streamlines the iterative refinement process.

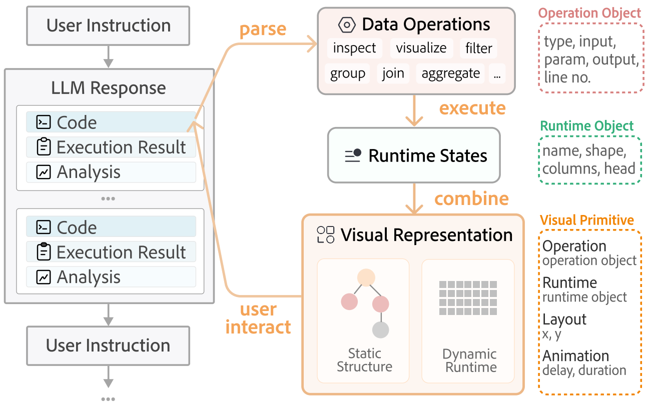


Figure 2: We propose a workflow that identifies data operations within the generated code and maps them to visual, interactive primitives on the fly. These primitives collectively offer an overview of the data analysis process.

DC4. Embed visualization seamlessly into the conversational user interface (CUI). As conversational data analysis normally takes place in a CUI [9, 19], we tailor the design to common design patterns of web CUIs in a non-intrusive manner. For instance, the visualization should be stably revealed during the progressive generation, following the same vertical order as the code. It should offer a lightweight complementary view of the code section in the LLM's response (see Figure. 2) and afford a level of visual guidance for the code dependency between conversational threads.

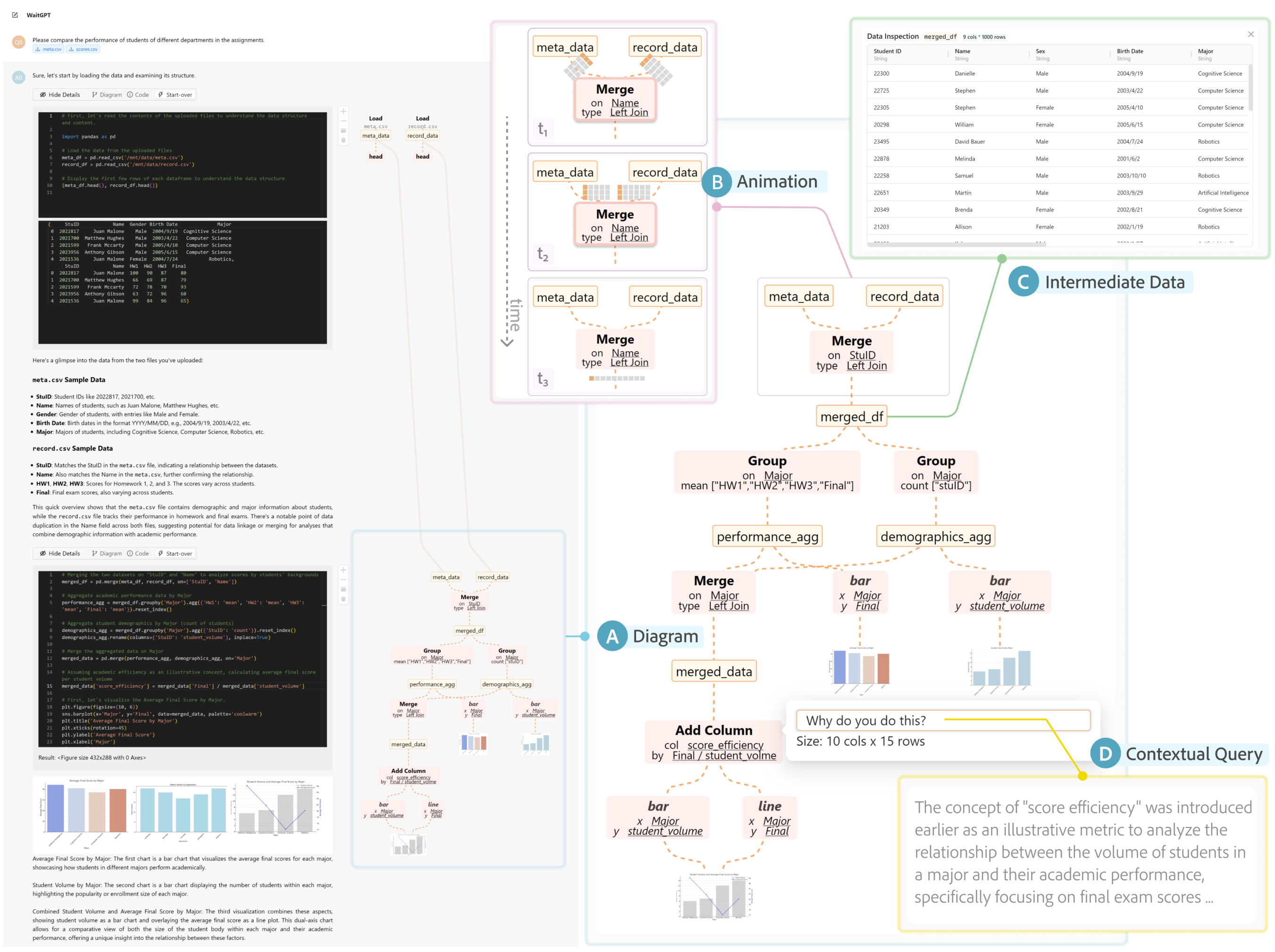
4 WAITGPT: USAGE SCENARIO

Informed by the formative study and design considerations, we propose dynamically visualizing the code generation process to help users steer a conversational LLM agent during the data analysis process. This is achieved through a workflow that identifies data operations within the generated code and maps them to visual primitives on the fly (see Figure. 2). These visual primitives not only illustrate the static aspects of data operations but also display the runtime states of the underlying data (i.e., tables) both before and after these operations. Moreover, they provide users with rich interaction possibilities, allowing them to refine the data operations without regenerating the code entirely.

We instantiate this idea with a prototype system, WaitGPT, which enables users to proactively guide the data analysis process with an LLM agent, making interventions akin to saying, *"Wait, GPT, there is something wrong..."* This section walks through WaitGPT using a hypothetical use case, demonstrating its capacity to transform the user's interaction with LLMs in data analysis tasks.

Usage Scenario. Zoey, a college lecturer, would like to review her students' performance across assignments to inform future teaching strategies. She opened WaitGPT, an LLM-powered conversational tool for data analysis that she was familiar with.

WaitGPT's interface resembles a chat box, allowing users to upload spreadsheets and inquire about the data in natural language (Figure. 3). Upon uploading two spreadsheets — one detailing student profiles and the other their individual assignment scores —



Please compare the performance of students of different departments in the assignments.

Sure, let's start by loading the data and examining its structure.

```
# First, let's read the contents of the uploaded files to understand the data structure
and content.

import pandas as pd

# Load the data from the uploaded files
meta_df = pd.read_csv('/mnt/data/meta.csv')
record_df = pd.read_csv('/mnt/data/record.csv')

# Display the first few rows of each dataframe to understand the data structure
(meta_df.head(), record_df.head())
```

```
(     StuID           Name  Gender Birth Date              Major
 0  2022817     Juan Malone    Male  2004/9/19  Cognitive Science
 1  2021700  Matthew Hughes    Male  2003/4/22   Computer Science
 2  2021599   Frank Mccarty    Male  2005/4/10   Computer Science
 3  2023956  Anthony Gibson    Male  2005/6/15   Computer Science
 4  2021536     Juan Malone  Female  2004/7/24           Robotics,
     StuID           Name  HW1  HW2  HW3  Final
 0  2022817     Juan Malone  100   90   87     80
 1  2021700  Matthew Hughes   66   69   87     79
 2  2021599   Frank Mccarty   72   78   70     93
 3  2023956  Anthony Gibson   63   72   96     60
 4  2021536     Juan Malone   99   84   96     65)
```

Here's a glimpse into the data from the two files you've uploaded:

meta.csv Sample Data

- **StuID**: Student IDs like 2022817, 2021700, etc.
- **Name**: Names of students, such as Juan Malone, Matthew Hughes, etc.
- **Gender**: Gender of students, with entries like Male and Female.
- **Birth Date**: Birth dates in the format YYYY/MM/DD, e.g., 2004/9/19, 2003/4/22, etc.
- **Major**: Majors of students, including Cognitive Science, Computer Science, Robotics, etc.

record.csv Sample Data

- **StuID**: Matches the StuID in the meta.csv file, indicating a relationship between the datasets.
- **Name**: Also matches the Name in the meta.csv, further confirming the relationship.
- **HW1**, **HW2**, **HW3**: Scores for Homework 1, 2, and 3. The scores vary across students.
- **Final**: Final exam scores, also varying across students.

This quick overview shows that the meta.csv file contains demographic and major information about students, while the record.csv file tracks their performance in homework and final exams. There's a notable point of data duplication in the Name field across both files, suggesting potential for data linkage or merging for analyses that combine demographic information with academic performance.

```
# Merging the two datasets on "StuID" and "Name" to analyze scores by students' backgrounds
merged_df = pd.merge(meta_df, record_df, on=['StuID', 'Name'])

# Aggregate academic performance data by Major
performance_agg = merged_df.groupby('Major').agg({'HW1': 'mean', 'HW2': 'mean', 'HW3':
'mean', 'Final': 'mean'}).reset_index()

# Aggregate student demographics by Major (count of students)
demographics_agg = merged_df.groupby('Major').agg({'StuID': 'count'}).reset_index()
demographics_agg.rename(columns={'StuID': 'student_volume'}, inplace=True)

# Merge the aggregated data on Major
merged_data = pd.merge(performance_agg, demographics_agg, on='Major')

# Assuming academic efficiency as an illustrative concept, calculating average final score
per student volume
merged_data['score_efficiency'] = merged_data['Final'] / merged_data['student_volume']

# First, let's visualize the Average Final Score by Major.
plt.figure(figsize=(10, 6))
sns.barplot(x='Major', y='Final', data=merged_data, palette='coolwarm')
plt.title('Average Final Score by Major')
plt.xticks(rotation=45)
plt.ylabel('Average Final Score')
plt.xlabel('Major')
```

Result: <Figure size 432x288 with 0 Axes>

Average Final Score by Major: The first chart is a bar chart that visualizes the average final scores for each major, showcasing how students in different majors perform academically.

Student Volume by Major: The second chart is a bar chart displaying the number of students within each major, highlighting the popularity or enrollment size of each major.

Combined Student Volume and Average Final Score by Major: The third visualization combines these aspects, showing student volume as a bar chart and overlaying the average final score as a line plot. This dual-axis chart allows for a comparative view of both the size of the student body within each major and their academic performance, offering a unique insight into the relationship between these factors.

Data Inspection merged_df 9 cols * 1000 rows

Student ID String	Name String	Sex String	Birth Date String	Major String
22300	Danielle	Male	2004/9/19	Cognitive Science
22725	Stephen	Male	2003/4/22	Computer Science
22305	Stephen	Female	2005/4/10	Computer Science
20298	William	Female	2005/6/15	Computer Science
23495	David Bauer	Male	2004/7/24	Robotics
22878	Melinda	Male	2001/6/2	Computer Science
22258	Samuel	Male	2003/10/10	Robotics
22651	Martin	Male	2003/9/29	Artificial Intelligence
20349	Brenda	Female	2002/8/21	Cognitive Science
21203	Allison	Female	2002/1/19	Robotics

The concept of "score efficiency" was introduced earlier as an illustrative metric to analyze the relationship between the volume of students in a major and their academic performance, specifically focusing on final exam scores ...

Figure 3: A screenshot of the WaitGPT user interface. (A) An enlarged view of the flow diagram representing the code. (B) An illustration of the "table glyphs" that flow along the edge showing table dependency and changes during code generation. (C) Inspecting intermediate data by toggling the interactive table panel. (D) Interrogating LLM based on an operation.

Zoey asks WaitGPT to compare the performance of students with different backgrounds. In response, WaitGPT outlines a plan to meet her requirements, then crafts a code snippet to conduct analysis. An external executor executes this code snippet to yield results.

Unlike similar tools, WaitGPT visualizes the data analysis process instead of just presenting raw code and textual execution results (Figure. 3 A). It dynamically extracts data operations and presents them as nodes within a diagram illustrating the data flow. For instance, a "join" operation node would display as "merge". And the node shows the tables being joined, the type of join (e.g., left join, cross join, etc.), and the indexing column used for the join. These blocks are linked based on dependencies and posited from left to right to reflect the procedural order. Notably, WaitGPT breaks down the analysis script into executable blocks that are executed immediately instead of executing until the entire code snippet is ready. This allows for a progressive understanding and debugging process, enabling users to see the effects of each operation in real time. The tool also visualizes the runtime state of data tables (e.g., the number of data entries/columns, selected columns) as part of the diagram. Specifically, the runtime state of each table is visualized as glyphs, which move along the linked edges between operation objects.

Through the visual representation, Zoey quickly spots a flaw in the diagram—the row number reduces (Figure. 3 B). Rather than requiring rewriting the original query and regenerating the entire data analysis code, WaitGPT enables users to refine specific operations directly within the visualizations. Users can directly update its parameters, inquire about details, and indicate refinement intents through natural language. Thus, Zoey adjusts the join parameters to student IDs, and then clicks on the re-run button to execute the updated code. While the analysis goes on, Zoey inspects the table. She requests the LLM to clean the data. The diagram updates, reflecting the corrected scores after the agent integrates a data validation operation. Now Zoey is ready to analyze the reliable data, her teaching plans are secure on a foundation of accuracy.

5 WAITGPT: SYSTEM DESIGN

The design of WaitGPT consists of three major components: abstracting the code to data operation chains, visualizing these chains, and providing interactions to steer the analysis process.

5.1 Abstracting Code to Operation Chains

Based on the interview, we identified three types of information indispensable for code comprehension: table variables, data operations, and execution results. In addition, different data operations encapsulate dedicated semantics and independent parameters. Therefore, we opt to abstract a data analysis process into a graph structure, chaining its nodes with an input-output relationship as follows (**DC1**). The input of each data operation is table(s), whereas the output can be the updated table, new table(s), other derived values/visualizations, or none.

∗ *Table node*: A table node corresponds to a variable for an underlying table in the code, such as a `dataframe` in the Pandas package. It can be either loaded from a data file or dynamically generated during code execution as an interim variable.
∗ *Operation node*: An operation node ties to an atomic data operation. It surfaces the detailed parameters of an operation object, e.g.,, Select, Filter, and Sort.
∗ *Result node*: A result node is associated with an execution result, such as printed values or data visualization.

Additionally, the relationship between these nodes can be one of the following:

∗ *Input*: From table node(s) to an operation node. It means the data operation is based on the input table(s).
∗ *Assignment*: From an operation node to a new table node. It means a new table-typed variable is yielded from the operation.
∗ *Result generation*: From an operation node to a result node. It means the operation outputs some visible results.
∗ *Operation chain*: From an operation node to an operation node. It means a table undergoes the two operations sequentially.

Extracting the Nodes through Static Analysis. To extract these nodes and relationships, we perform static analysis on the abstract syntax tree (AST) of the generated code, where we apply heuristics informed by patterns of data analysis scripts and functional interface design of relevant packages. WaitGPT currently can parse atomic operations including Load Data, Inspect, Select, Filter, Sort, Transform, Group, Aggregate, Merge, Add Column, and Visualize, based on the `Pandas`, `Matplotlib`, and `Seaborn` packages, which are the default choices of ChatGPT and widely adopted [6]. For instance, `merge_df = df[["attr_1", "attr_2"]].sort()` will be converted into two operation objects: Select and Sort. To bind the table targets to the operations, we maintain a global variable of existing table variables. This is because a table variable can only be created by being loaded from external sources (files, database, etc.) or generated as the output of prior operations.

5.2 Visualizing Data Operation Chains

Our goal is to transform the LLM-generated code into easily interpretable visualizations, facilitating user inspection of the data analysis process (**DC2**). To this end, we have developed a suite of visual primitives, which present the details of each operation and their internal runtime states. These primitives are chained together, collectively offering an overview of the data analysis process.

Visual primitives for the static code. We utilize a diagram to represent the graph-based data processing procedure for individual code snippets. The table node, operation node, and result node are visualized as blocks, color-encoded in yellow, pink, and white. A node-style visualization is chosen for its familiarity to general users (**DC2**) and flexibility in displaying layered information, expanding with the code stream, and implying the operation order (**DC4**). As LLMs sometimes synthesize long variable names for clarification, we considered a rectangular block beneficial for encapsulating this information. For simplicity, a table node only shows the corresponding variable name, and a result node shows a thumbnail. For an operation node, we use a bold font style to prioritize the communication of its type (e.g., filter, group, etc.). And we visually differentiate its parameters' names and values through typography.

An operation chain spans from top to bottom, following its procedure. For a table node, there can be multiple associated operation chains. These chains are aligned from left to right with respect to the execution order. A code snippet depends on preexisting code as the runtime environment is shared throughout a conversation. Therefore, a table node may trace back to previous snippets. To reflect such a relationship, a copy is made in such a situation, which is linked to its previous occurrence with a cross-conversation curve.

Visual primitives for the runtime states. The diagram is further enriched by visual glyphs that encode the runtime status of table variables. A table glyph takes a common visual representation for tables—a 2D matrix. The number of matrix columns is the same as the column number of the table. The number of matrix rows per column is proportional to the number of table rows to roughly indicate changes in data size and scale to different data sizes. To access precise information about the runtime states, one may interact with the associated operation node for details. Through chained operations, the size of a table can be updated.

5.3 Steering the Data Analysis of LLMs

The diagram goes beyond merely a visual representation of the data analysis process. It also acts as an interactive scaffold for users to steer data analysis code generated by LLMs, enabling real-time inspection, retrospective examination, and granular refinement (**DC3**). This section introduces interactions supported in WaitGPT.

5.3.1 *Real-Time Inspection on the underlying code.*

During code generation, only the diagram is shown to reduce the cognitive load of end users. However, they may still toggle on the code panel and juxtapose the diagram side-by-side. When a data operation is being activated, i.e., the external executor has just run the code, it will be added to the diagram, potentially introducing a new table node or a result node. Meanwhile, relevant table glyphs also appear and gradually flow from the previous node to the current node. Figure.4 showcases an example of the dynamic process.

5.3.2 *Retrospective Investigation on the analysis process.*

After the code and diagram are completely generated, users may perform a retrospective examination to verify the procedure and investigate potential issues. To evaluate the analysis flow, users may replay

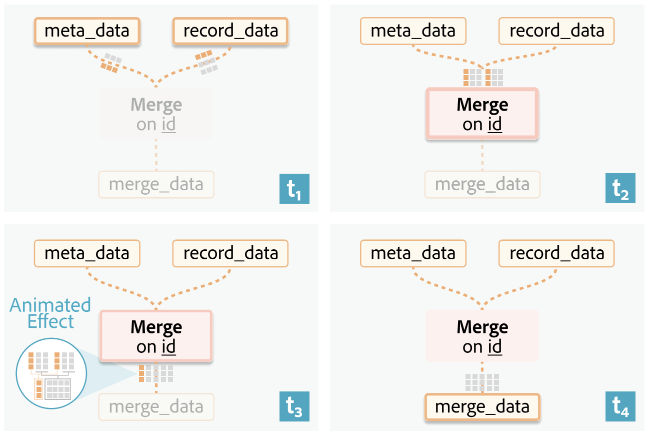
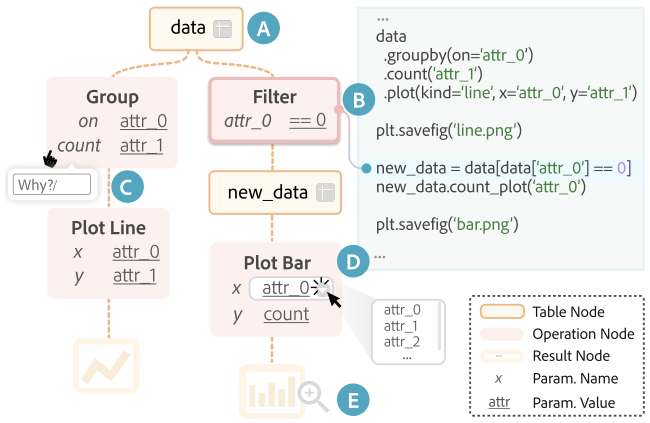


Figure 4: An illustration of how the diagram grows with animated table glyphs during the code generation process.



```
...
data
  .groupby(on='attr_0')
  .count('attr_1')
  .plot(kind='line', x='attr_0', y='attr_1')

plt.savefig('line.png')

new_data = data[data['attr_0'] == 0]
new_data.count_plot('attr_0')

plt.savefig('bar.png')
...
```

Figure 5: The visualization offers multiple interactions for inspecting and refining the underlying data analysis. Users can: (A) toggle a table node to view the underlying data; (B) hover over a node to highlight its corresponding code; (C) modify a data operation using natural language; (D) directly manipulate the parameters of a node; and (E) view the resulting visualizations from the analysis.

the animation showing diagram expansion or utilize scrolly-telling, where they can take control over the animation progress using scroll-based interactions (**DC4**). If the code panel is toggled on, the corresponding line(s) of code will be highlighted for activated nodes upon re-play or the user's mouse hover events (see Figure. 5 A). This feature bridges the visual representation and the textual code, visual navigation and troubleshooting. Essentially, nodes in a diagram are visually displayed in the simplest way to support fast comprehension. To access details about the underlying data tables in the runtime context, users may click on a node of interest and review an additional panel (see Figure. 5 B). The thumbnail of a visualization result node is expandable (see Figure. 5 E).

5.3.3 Granular Refinement. The diagram offers new interaction modes for granular refinement through direct manipulation and contextual interrogation. Instead of regenerating the entire analysis, which may involve multiple code snippets, users can steer the data analysis at a finer granularity within the visualization (**DC3**). Users may directly manipulate the operation objects based on their visual representation and update the underlying code (see Figure. 5 D). The fields of parameters in operation nodes are editable input forms, allowing fine-grain updates.

Similar to the concept of interrogative debugging [30], users can select specific operation nodes within the diagram and then request explanations or suggest revisions to the LLM by focusing on a particular node, which offers a targeted context for verification and refinement (Figure. 5 C). This provides an alternative mode to the common practice of selecting code or table cells and posting queries to LLMs [43]. Inspired by the regeneration practice of participants in the formative study, the query is independent of the main conversation and, thus, will not affect the memory of LLM agents. The LLM's suggestion of code update will directly apply to the code panel, and the previous version will be commented out for comparison. When satisfied with the refinement, users can re-run the code snippet to attain updated analysis from LLM agents based on the new execution results.

6 IMPLEMENTATION

WaitGPT is a web-based app implemented in the React [40] framework based on TypeScript. We apply the Monaco Editor [41] to display the code with standard syntax highlighting. We adopt the OpenAI's API, with the `gpt-4-0125-preview` model. To manage user-uploaded files, parse LLM-synthesized code into an abstract syntax tree, and obtain its execution result, we also host a back-end server implemented in Python with Flask [48]. The LLM prompts applied in WaitGPT generally follow the guidance of OpenAI with little engineering effort. Our implementation integrates three key mechanisms as follows.

Session Management. In addition to the conversation history for each session, WaitGPT maintains other contexts to support diagram generation on the fly and granular refinement. The associated contexts include a sandbox environment for file storage and code execution, a global record of table variables, and specifications of the diagram for each data analysis code snippet. In addition to the parsed parameters, the runtime status of target tables, and rendering configurations, the specification of a data operation node in a diagram also records conversation logs with the LLMs based on the code to support iterative refinement.

When a user sends a query, the LLM will respond with textual contents or a function call to the pre-declared Python executable. For code-based response, WaitGPT first decides whether it is about data analysis and then activates the automatic parser. The runtime context for each code snippet is cloned from the main process and cached for potential rework, thus enabling flexible user interruptions and refinement at any point. We enhance user navigation by prompting LLM to summarize the main task and build a minimap for existing data analysis snippets.

Sandbox Execution. Before running the code in a sandbox environment, WaitGPT refactors the method chain into separate standalone statements. Therefore, based on the identified targets (i.e., table variables) of data operations, the static parser inserts printing statements based on templates to retrieve the intermediate status of the table, including the number of rows, the number of columns, and column names. The table status is then bonded to the corresponding data operation object. As a note, we opt to insert code to the LLM-generated script in a post hoc manner to reduce dependency on specific versions. An alternative approach is to inject logging facilities into the standard libraries [50, 55].

Rendering. The rendering of the flow diagram comprises two steps. Once the static analyzer extracts new data operation objects, they will be added to the diagram using a graph layout algorithm and maintain inactivated status. When the runtime information is bound to the operation object, its animated effect is pushed to a queue to play sequentially, where the corresponding node will be activated and the table glyph will flow from the prior node to the current node.

7 USER EVALUATION

We evaluate WaitGPT through an in-lab user study with 12 participants of various backgrounds and data analysis expertise. Specifically, we are interested in the following research questions.

- How effectively does WaitGPT facilitate intermediate verification during the generation process of LLM agents?
- How effectively does WaitGPT support retrospective verification after data analysis tasks are completed?
- To what extent does WaitGPT support the granular refinement of generated code snippets?
- How do users perceive the usefulness of WaitGPT in their daily data analysis tasks?

7.1 Participants

We recruited 12 participants (10 males, 2 females; ages 23—30, M = 26.33, SD = 2.15) through social media and word-of-mouth. They were postgraduate students with diverse backgrounds in databases, machine learning, visual analytics, industrial engineering, computational sociology, and HCI. According to their self-rating based on a 5-point Likert scale (1: lowest extent, 5: greatest extent), participants were generally adept at data analysis (M = 3.67, SD = 1.37) and familiar with the Pandas syntax used in WaitGPT (M = 3.5, SD = 1.38). They were experienced with LLM-powered chatbots (M = 3.75, SD = 1.06). Specifically, 5/12 participants leveraged ChatGPT to analyze more than 20 datasets, whereas 4/12 analyzed less than 5 datasets on ChatGPT.

7.2 Protocol

Tasks. There are three tasks in total. Task A is based on the *Employee* dataset[1] with six analysis tasks (A1–A6). Task B is based on the *Flight* dataset[2] with four tasks (B1–B4). For Tasks A & B, the participants are required to address individual questions by interacting with LLMs and decide if the LLM-generated code is error-free. To cover representative cases, we included both confirmation and exploratory tasks on two tabular datasets and replicated 4 known errors made by LLMs [19]. In addition, we prepared dedicated prompts for the participants to ensure that the first LLM-generated content was identical in each task. These prompts are grounded in the ARCADE [74] and Text2Analysis [23] datasets. Each data analysis task is independent of the other, including common data insight types [13], e.g., rank, distribution, outlier, etc. Task C is based on the synthesized dataset used in the usage scenario (see Sec. 4), where participants were asked to explore the dataset freely. We also offer a list of self-curated queries for their reference.

Baseline and Apparatus. We removed the extended view of the diagram as the baseline system, namely Baseline. Baseline retains essential functionalities of ChatGPT that the participants are familiar with. The code snippet offers by-line textual comments explaining each step for user verification and has standard syntax highlighting for Python. Meanwhile, Baseline shares the same visual appearance as WaitGPT. This ensures that any differences in user interaction can be attributed to the diagram's presence or absence rather than other factors like aesthetics or layout. Participants joined the study in person and finished their tasks on standardized desktop devices to eliminate hardware variability as a confounding factor.

Procedure. We opted for a counterbalanced within-subjects design to compare WaitGPT and Baseline. There are two groups (I, II) that participants were randomly assigned to. In Group I, participants finish A1-3 & B1-2 in Baseline, and A4-6 & B3-4 in WaitGPT. Conversely, in Group II, participants finish A1-3 & B1-2 in WaitGPT, and A4-6 & B3-4 in Baseline. This approach allowed each participant to experience both conditions while performing a balanced set of tasks across the two systems.

The user study begins with a presentation of the visualization and interaction design, where participants can ask for details (5 min). Then, the participant should work on Task A1-6 (15-30 min), Task B1-4 (10-20 min), and Task C (5-15 min) sequentially. The study ends with a semi-structured interview (10-15 min) and a questionnaire (5 min). A facilitator conducted one-on-one sessions with each participant, closely observing and taking notes of participant behaviors. The post-study interview was audio-recorded for later analysis. Participants were compensated with $12 per hour.

7.3 Measures

We adopted the NASA-TLX [22] questionnaire to measure the perceived cognitive load in steering LLM-synthesized data analysis. We developed a questionnaire based on a 7-point Likert scale to evaluate the usefulness of WaitGPT. For each pre-recorded query, the facilitator records (1) the time cost that the participant discerns issues in the result since response generation, (2) the time cost that the participant makes a judgment on the correctness, (3) whether the data has been examined, and (4) whether the code panel is expanded when viewing diagrams only.

7.4 Results

To compare Baseline and WaitGPT, we analyze task correctness for Task A & B and the subjective ratings of the participants. We further report insights from the interview,

[1]https://www.kaggle.com/datasets/soorajgupta7/corporate-compensation-insights
[2]https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction

Table 2: The success rate (%) and average duration (seconds) in WaitGPT and Baseline for Task A & Task B (N=6/condition). The failure column describes the mistake made by LLMs in the task. #Line: No. lines in the code snippet; #Char: No. characters. #Df: No. table nodes in the data operation chains, #Op: No. operation nodes, #Res: No. result nodes. "(Value)": standard deviance.

Task	Failure	#Line	#Char	#Df	#Op	#Res	Success (%)		Average Duration (s)	
							WaitGPT	Baseline	WaitGPT	Baseline
A1	Sort on string	14	474	2	5	0	83 (0.41)	33 (0.52)	65.83 (45.32)	136.67 (88.69)
A2	Miss a group condition	5	233	2	3	0	50 (0.55)	50 (0.55)	88.33 (40.21)	102.50 (68.68)
A3	NA	47	1,836	5	10	1	100 (0.00)	100 (0.00)	154.00 (103.00)	151.67 (143.69)
A4	Miss a filter condition	10	509	3	4	0	67 (0.52)	83 (0.41)	86.67 (18.62)	92.50 (34.31)
A5	Miss dropping duplicates	24	780	1	5	1	50 (0.06)	50 (0.55)	92.50 (58.37)	87.50 (27.34)
A6	NA	21	817	1	3	1	100 (0.10)	100 (0.00)	144.17 (69.02)	95.83 (22.45)
B1	NA	29	1,167	4	7	1	100 (0.00)	83 (0.41)	141.67 (49.97)	160.00 (82.16)
B2	Miss dropping duplicates	25	1,287	5	6	1	67 (0.52)	50 (0.55)	242.50 (167.74)	221.67 (159.80)
B3	NA	25	1,262	4	6	1	100 (0.00)	100 (0.00)	185.00 (113.31)	176.67 (64.94)
B4	Wrong aggregation logic	10	654	4	6	0	83 (0.41)	83 (0.41)	212.50 (145.87)	138.50 (40.71)

7.4.1 Task Correctness. Table 2 lists detailed configurations and participant performance in Task A1-6 and B1-4. In general, the success rates in the WaitGPT condition are no less than the Baseline condition, except for A4. A4 asked for 10 employees with the highest salary currently, whereas LLM did not filter out those on leave. Many participants did not notice this problem in the response. As for the duration, the two conditions had similar time costs (≤ 10s) for Task A3-5 and B3. And WaitGPT took less time in Task A1-2 and Task B. However, multiple factors are attributed to the total duration, as seen in the relatively large standard deviation values. For instance, we did not consider expertise in data analysis when assigning participants to different groups. When the participant chose to inspect the code after viewing the diagram, there was an additional time cost to browse the code.

7.4.2 Subjective Ratings. As the questionnaires are based on an ordinal Likert scale and the sample size is relatively small, we performed the Wilcoxon signed-rank test to compare the subjective ratings between Baseline and WaitGPT.

⋄ On the cognitive load. In the NASA-TLX questionnaire, WaitGPT demonstrates lower cognitive demand to the participants. According to the statistical tests, there are highly significant differences (p<.001) in the mental and physical demand, performance, and affective states between the two conditions. The difference in the effort to accomplish self-performance level (p=.010) and the temporal demand (p=.050) is also significant.

⋄ On the usefulness. Figure. 6 compares the distribution of the user ratings on Baseline and WaitGPT based on our self-developed questionnaire. For each question, WaitGPT attains a higher median rating than Baseline at a confidence level of 99.5%, demonstrating its usefulness in demystifying the analysis (Q1-3), verifying or correcting the code (Q4-5), and engaging end-users (Q6). Notably, while participants varied in task performance, 10/12 people reported increased confidence in the correctness of the analysis result (Q1). Besides, based on a 7-point Likert scale (1: strongly disagree, 7: strongly agree), the participants considered it easy to comprehend the visualization design (Med=6.5, M=6.65, SD=.87) and interact with the diagram (Med=6.0, M=6.33, SD=.65).

7.4.3 General impressions. The participants were generally positive about WaitGPT and affirmed its support in monitoring and steering LLM-generated analysis.

⋄ Difference in the UX between conditions. Despite in-line explanations and meaningful variable names in the LLM-generated code, the participants found it mentally taxing to follow the source code and unguided in verification. The reasons include memory demand for excessively long content (8/12), limited runtime contexts (3/12), and unfamiliar coding styles (2/12). In comparison, participants (12/12) resonated with the ease of understanding and verifying the code in WaitGPT with a higher level abstraction. The diagram "*strips off unimportant details*" (P5) and offers an overview of the code. "*It [the diagram] has a clean structure and can serve as a navigation for the code.*" (P11) This also kept participants engaged during the code generation. "*I felt stressed viewing the code stream, but it's a pleasure to watch the diagram grow.*" (P4) The benefits of a visual summary were more apparent when the underlying code was long, as the diagram fit in the screen without the need to scroll vertically or horizontally (3/12). Lastly, many participants (8/12) were positive about the node-based interaction instead of sending a new chat. "*There's a chance that a new chat introduces new errors, so I prefer to change the code directly.*" (P9)

⋄ Perceived usefulness of the visualization. The current visual design was well-received by the participants (12/12). We categorize the perceived usefulness of the extended visualization and associated interactions into three dimensions.

First, the diagram offers an abstract layer to focus on high-level logic and task decomposition. As observed by P12, "*GPT outputs pretty code with mostly correct functional calls. This makes me lose caution for logical errors.*" P3 claimed that the visualization facilitated LLM alignment—"*I have a rough idea of how to process the data, and the diagram makes it easy to compare with my mind map.*"

Second, the visualization surfaces information at different layers, including the detailed parameters for data operations, profiles of the data table, and navigation back to the source code. For instance, the high accuracy rate for Task A1 was due to the convenience of inspecting data tables. "*It's great to access the table right away. It's [the diagram] like an information hub.*" (P10) P3 appreciated the typography applied in the operation nodes, as "*it separates the*

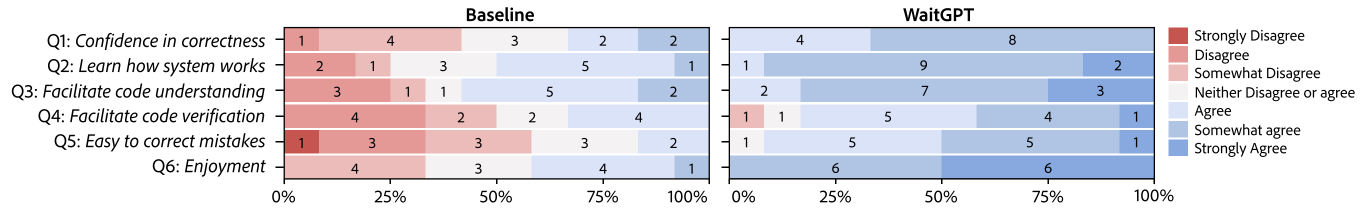


Figure 6: User ratings on the baseline (code-only interface) and WaitGPT.

variable names, operations, parameter names, and parameters". P7 noted that the table glyphs suggested the semantics of unfamiliar functions through the input-output trace.

Third, the node-based interactions offer a granular approach to interrogating or modifying the code. "*I prefer talking to nodes in the diagram because the context is preserved, so I don't need to type much. It's nice to have something to point to make things clearer.*" (P9)

Some participants (2/11) felt more comfortable manipulating the nodes than overwriting the code. "*Here [in the diagram], I don't need to care much about syntax but doing minimum updates.*" (P5) In addition, the context of a node-based interaction is constrained to the corresponding code section parallel to the entire conversation. "*I am happy to maintain a clean conversation thread.*" (P11)

7.4.4 Glitches in using WaitGPT. Despite the benefits mentioned, users encountered several glitches while using the prototype.

◇ Diverse needs for level of details. Participants had divergent perspectives on the current design of WaitGPT. For instance, P10 expressed the hope of showing relevant annotations directly on the operation nodes. For the table glyphs, a few participants (2/12) competent in data analysis criticized them as trivial. "*I'd prefer a small annotation showing the table dimensions.*" (P9) However, some participants (3/12) embraced the design and commented that its animation double encoded the program procedure, in addition to the implicit node layout from left to right—"*When the code has complex dependencies, I can follow the operations step by step with the table glyphs.*" (P2) To accommodate diverse needs, a customizable interface is anticipated for flexible user configuration.

◇ Concerns in the reliability & expressiveness. Participants with a computer science background (8/12) were generally interested in how the code was transformed into the diagram and expressed concerns about algorithmic failures (1/12) or potential information loss (2/12). Like what P12 asked: "*What if it [LLM] made errors in parameters not presented in the diagram?*" P6 recalled that he sometimes copied his code and prompted LLMs to use customized lambda functions for data transformation. However, in the current implementation, WaitGPT will only tag this as a "lambda function" without presenting more details due to the limit of current heuristics. As there are limited datasets on LLM-synthesized data analysis code at the moment, it remains challenging to systematically evaluate the coverage of our heuristics. To mitigate these concerns, future improvements may incorporate automatic verification of the parsing results and generative AI to surpass expressiveness limits.

7.4.5 Opportunities for Applications. The participants shared several creative ideas for extending WaitGPT. P8 wanted to transfer the underlying concept into a visualization authoring context, where the encoding specifications are procedural and atomized—"*After analyzing the data, I need to present it with high-quality visualizations, but tools like ChatGPT often fail my expectations.*" P7 saw the value of a diagram in communication, especially to an audience with limited technical backgrounds. He said: "*I can use the scroll-telling in my presentation to explain how the data has been transformed.*" P3 envisioned a visual programming paradigm in which the basic building blocks can be self-composed or reused to communicate intention in addition to textual prompts to LLMs.

8 DISCUSSION

In this section, we synthesize the implications and potential avenues for future research and reflect on the limitations.

8.1 Design Implications

Monitoring LLM agent through "visible hands". Despite recent progress, known issues like hallucinations in LLM agents warrant external steering. In WaitGPT, we abstract the LLM's generated content into high-level operations rather than raw text outputs, which align more closely with human cognitive processes. Our approach also enriches the design space of *AI resilient* interfaces [20]. Through static analysis, WaitGPT translates synthesized programs into abstracted operations. These abstracted operations are brought to life through dynamic visual representations, making it possible for end-users to monitor the actions of LLM agents, similar to watching "visible hands" in real-time. Future design may consider a similar mechanism of semantically rich representation and incremental update [76] in communicating agent actions.

Scrollytelling for LLM-generated content. WaitGPT incorporates a basic form of scrollytelling, guiding users through the code by highlighting the corresponding diagrams as they scroll through the generated content. By combining the flow diagram with a scroll-triggered revealing mechanism, this technique aligns naturally with the generating process of LLM-produced content, fostering a deeper engagement and understanding of the content. Looking ahead, we advocate developing automated streaming methods to create scrollytelling narratives for presenting LLM-generated content. This complements the animation in the steaming generation phase, allowing users to control their understanding speed rather than passively following a predefined playing timeline.

Addressing context composition in different task granularity. One interesting property of LLMs is that they can provide reasonably high-quality responses to a wide variety of user tasks [57]. Echoing our formative study, users may request background information or incorporate more contexts when analyzing data. They may start a sub-thread to test their assumptions [18]. The highly diverse and evolving nature of user tasks in LLM-powered data analysis necessitates the development of adaptive user interfaces. A more challenging direction is to generate visual representations for miscellaneous contexts in unpredictable LLM responses.

8.2 Future Works

Democratizing data consumption with verifiable generative AI. With nowadays generative AI, individuals without a programming background may easily create data visualizations for analysis or communication. However, such democratization comes with challenges, particularly in ensuring the accuracy and reliability of AI-generated content. There's a pressing need to navigate users to the potential inaccuracies and biases inherent in AI outputs [7, 28, 77]. We believe that the key to fully leveraging AI's capabilities in data consumption hinges on creating user interfaces that align with the expertise levels of the intended users. In addition, different data tasks raise different requirements warranting tailored supports, such as an emphasis on the authorial intent matching of encoding schemes in expressive visualization design (e.g., [61, 70]).

Introducing a "stop" mechanism in human-LLM agent interaction. While WaitGPT is based on a chatbot-like interface, such an interaction paradigm can apply to a standalone AI assistant integrated into data analysis software or notebook platforms [38]. Essentially, during the ongoing conversation with LLM agents, users may be overwhelmed by the token-based output and fail to prevent propagating errors in time. WaitGPT integrates proactive strategies to identify and rectify potential failures in AI-generated content. Similarly, future works may further enrich the design space of visual representations of LLM outputs [4, 20] for instant understanding and explore a low-cost approach to facilitate steering content generation based on intermediate outputs.

Exploiting interaction modalities in conversational data interface. First, beyond textual prompts with simple selections of data slices in ChatGPT, future systems may incorporate other input types like direct manipulation [37], demonstration [24], and reference [71]. Second, to navigate users in nuanced decisions with drill-down explorations [18, 19], it is promising to provide explanations on demand [39], or establish a tighter connection between code, data, textual analysis, and generated visualizations [5, 66]. Last, enabling users to directly reuse the generated code or interact with the resulting visualizations for further exploration [16, 67] could augment the flexibility of conversational data analysis tools.

8.3 Limitation

Threats to validity. The sample size in our formative and evaluation studies is relatively small and thus may not be representative of the broader population of data analysts and LLM users. In the evaluation study, both conditions were equipped with standard syntax highlight for Python language. However, without a careful visual design for key operations in the Baseline, participants may favor more on WaitGPT with its simplified information. Besides, participants were prompted to view the transformable representation of the data analysis script, which may not reflect their natural interaction patterns. The reported usability rating may also be subject to response bias [11] and participants' familiarity with the tasks. Future works may investigate how and how often users leverage this augmented view in their natural working space without explicit prompts to capture its real-world utility.

Scalability issues. In the framework, translating code into a flow diagram requires static analysis, which is dependent on the syntax. WaitGPT is currently tailored to Python language and libraries like `Pandas` and `Matplotlib` for tubular data. A potential solution to improve generalizability is to redesign LLM prompts to allow a mixed output stream of code and underlying operation objects, e.g., [28, 58]. However, the code stream visualization may not work for SQL-like languages with a reversed execution order compared to the procedure declaration. Second, the flow diagram assumes a linear structure in the code, targeting fluent interfaces [55]. Future works can incorporate control flows like loops and visual primitives for other data types. Last, the current glyph design may not scale to tables with over 20 columns. To address this, unused columns can be aggregated, or important ones can be hidden.

9 CONCLUSION

In this paper, we introduced WaitGPT, a novel interface design that transforms LLM-generated code into an accessible, interactive representation to address the reliability issues and user challenges in LLM-powered data analysis tools. Drawing from an interview study with general users (N=8) of ChatGPT, we gained insights into general perspectives on these nascent tools and glitches in disruptive workflow, code verification, and labor-intensive iterations. By translating stream-based code into a growing visualization of the key data operations and affording granular interactions, WaitGPT empowers users to monitor and steer data analysis performed by LLM agents. A user study (N=12) covering basic data analysis tasks demonstrated that WaitGPT could enhance error detection rate and improve overall confidence in the results.

Our work contributes to the field of human-AI collaboration in data analysis by demonstrating the effectiveness of transformable code representations in facilitating user understanding and engagement. As LLM applications in data analysis become more prevalent, prioritizing user experience and trust through accessible, interactive interfaces will be crucial in harnessing the potential of these powerful tools while ensuring their reliability and usability. We urge more exploration of novel human-LLM interaction paradigms and intuitive visual representation design for LLM responses.

ACKNOWLEDGMENTS

This research is supported by RGC GRF grant 16210321. The first author thanks Prof. Hanspeter Pfister for hosting the visit to the Harvard Visual Computing Group. We also thank the anonymous reviewers for their constructive feedback, the participants in the formative and user studies, and Zhan Wang, Leixian Shen, Xiaofu Jin, Shuchang Xu, Dr. Yanna Lin, Dr. Qingyu Guo, and Dr. Yun Wang for their valuable input.

REFERENCES

[1] Tyler Angert, Miroslav Suzara, Jenny Han, Christopher Pondoc, and Hariharan Subramonyam. 2023. Spellburst: A node-based interface for exploratory creative coding with natural language prompts. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. ACM, New York, NY, Article 100, 22 pages. https://doi.org/10.1145/3586183.3606719

[2] Christian Bors, Theresia Gschwandtner, and Silvia Miksch. 2019. Capturing and visualizing provenance from data wrangling. *IEEE Comput. Graph. Appl.* 39, 6 (2019), 61–75. https://doi.org/10.1109/MCG.2019.2941856

[3] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. In *APA handbook of research methods in psychology, Vol. 2. Research designs: Quantitative, qualitative, neuropsychological, and biological*. APA, Washington D.C. https://doi.org/10.1037/13620-004

[4] Yuzhe Cai, Shaoguang Mao, Wenshan Wu, Zehua Wang, Yaobo Liang, Tao Ge, Chenfei Wu, You Wang, Ting Song, Yan Xia, Nan Duan, and Furu Wei. 2024. Low-code LLM: Graphical user interface over large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*. 12–25. https://aclanthology.org/2024.naacl-demo.2

[5] Yining Cao, Jane L. E, Chen Zhu-Tian, and Haijun Xia. 2023. DataParticles: Block-based and language-oriented authoring of animated unit visualizations. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 808, 15 pages. https://doi.org/10.1145/3544548.3581472

[6] Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2024. VisEval: A benchmark for data visualization in the era of large language models. arXiv:2407.00981

[7] Yida Chen, Aoyu Wu, Catherine Yeh Trevor DePodesta, Kenneth Li, Nicholas Castillo Marin, Oam Patel, Jan Riecke, Shivam Raval, Olivia Seow, Martin Wattenberg, and Fernanda Viégas. 2024. Designing a dashboard for transparency and control of conversational AI. arXiv:2406.07882

[8] Liying Cheng, Xingxuan Li, and Lidong Bing. 2023. Is GPT-4 a good data analyst?. In *Findings of the Association for Computational Linguistics: EMNLP*. ACL, 9496–9514. https://doi.org/10.18653/v1/2023.findings-emnlp.637

[9] Bhavya Chopra, Ananya Singha, Anna Fariha, Sumit Gulwani, Chris Parnin, Ashish Tiwari, and Austin Z Henley. 2023. Conversational challenges in AI-powered data science: Obstacles, needs, and design opportunities. arXiv:2310.16164

[10] John Joon Young Chung, Wooseok Kim, Kang Min Yoo, Hwaran Lee, Eytan Adar, and Minsuk Chang. 2022. TaleBrush: Sketching stories with generative pretrained language models. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 209, 19 pages. https://doi.org/10.1145/3491102.3501819

[11] Nicola Dell, Vidya Vaidyanathan, Indrani Medhi, Edward Cutrell, and William Thies. 2012. "Yours is better!" Participant response bias in HCI. In *Proceedings of the sigchi conference on human factors in computing systems*. ACM, New York, NY, 1321–1330. https://doi.org/10.1145/2207676.2208589

[12] Victor Dibia. 2023. LIDA: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. ACL, Toronto, Canada, 113–126. https://doi.org/10.18653/v1/2023.acl-demo.11

[13] Rui Ding, Shi Han, Yong Xu, Haidong Zhang, and Dongmei Zhang. 2019. QuickInsights: Quick and automatic discovery of insights from multi-dimensional data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, New York, NY, 317–332. https://doi.org/10.1145/3299869.3314037

[14] Yingchaojie Feng, Xingbo Wang, Bo Pan, Kam Kwai Wong, Yi Ren, Shi Liu, Zihan Yan, Yuxin Ma, Huamin Qu, and Wei Chen. 2024. XNLI: Explaining and diagnosing NLI-based visual data analysis. *IEEE Trans. Vis. Comput. Graph.* 30, 7 (2024), 3813–3827. https://doi.org/10.1109/TVCG.2023.3240003

[15] Kasra Ferdowsi, Jack Williams, Ian Drosos, Andrew D Gordon, Carina Negreanu, Nadia Polikarpova, Advait Sarkar, and Benjamin Zorn. 2023. ColDeco: An end user spreadsheet inspection tool for AI-generated code. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, Piscataway, NJ, 82–91. https://doi.org/10.1109/VL-HCC57772.2023.00017

[16] Kiran Gadhave, Zach Cutler, and Alexander Lex. 2022. Reusing interactive analysis workflows. *Comput. Graphics Forum* 41, 3 (2022), 133–144. https://doi.org/10.1111/cgf.14528

[17] Google. 2024. *Gemini Advanced: Release updates*. Google. Retrieved June 1, 2024 from https://gemini.google.com/updates 2024.05.21: updates on data analysis features.

[18] Ken Gu, Madeleine Grunde-McLaughlin, Andrew M. McNutt, Jeffrey Heer, and Tim Althoff. 2024. How do data analysts respond to AI assistance? A wizard-of-oz study. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 1015, 22 pages. https://doi.org/10.1145/3613904.3641891

[19] Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M Drucker. 2024. How do analysts understand and verify AI-assisted data analyses?. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 748, 22 pages. https://doi.org/10.1145/3613904.3642497

[20] Ziwei Gu, Ian Arawjo, Kenneth Li, Jonathan K Kummerfeld, and Elena L Glassman. 2024. An AI-resilient text rendering technique for reading and skimming documents. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 898, 22 pages. https://doi.org/10.1145/3613904.3642699

[21] Yi Guo, Nan Cao, Xiaoyu Qi, Haoyang Li, Danqing Shi, Jing Zhang, Qing Chen, and Daniel Weiskopf. 2023. Urania: Visualizing data analysis pipelines for natural language-based data exploration. arXiv:2306.07760

[22] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[23] Xinyi He, Mengyu Zhou, Xinrun Xu, Xiaojun Ma, Rui Ding, Lun Du, Yan Gao, Ran Jia, Xu Chen, Shi Han, Zejian Yuan, and Dongmei Zhang. 2024. Text2Analysis: A benchmark of table question answering with advanced data analysis and unclear queries. *Proceedings of the Annual AAAI Conference on Artificial Intelligence (AAAI)* 38, 16 (2024), 18206–18215. https://doi.org/10.1609/aaai.v38i16.29779

[24] Yanwei Huang, Yurun Yang, Xinhuan Shu, Ran Chen, Di Weng, and Yingcai Wu. 2024. Table Illustrator: Puzzle-based interactive authoring of plain tables. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 186, 18 pages. https://doi.org/10.1145/3613904.3642415

[25] Yanwei Huang, Yunfan Zhou, Ran Chen, Changhao Pan, Xinhuan Shu, Di Weng, and Yingcai Wu. 2023. Interactive table synthesis with natural language. *IEEE Trans. Vis. Comput. Graph.* (2023). https://doi.org/10.1109/TVCG.2023.3329120 Early Access.

[26] Peiling Jiang, Jude Rayan, Steven P Dow, and Haijun Xia. 2023. Graphologue: Exploring large language model responses with interactive diagrams. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. ACM, New York, NY, Article 3, 20 pages. https://doi.org/10.1145/3586183.3606737

[27] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, 3363–3372. https://doi.org/10.1145/1978942.1979444

[28] Majeed Kazemitabaar, Jack Williams, Ian Drosos, Tovi Grossman, Austin Henley, Carina Negreanu, and Advait Sarkar. 2024. Improving steering and verification in AI-assisted data analysis with interactive task decomposition. arXiv:2407.02651

[29] Meraj Khan, Larry Xu, Arnab Nandi, and Joseph M Hellerstein. 2017. Data tweening: Incremental visualization of data transforms. *Proceedings of the VLDB Endowment* 10, 6 (2017), 661–672. https://doi.org/10.14778/3055330.3055333

[30] Amy J Ko and Brad A Myers. 2004. Designing the Whyline: A debugging interface for asking questions about program behavior. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, 151–158. https://doi.org/10.1145/985692.985712

[31] Sam Lau, Sean Kross, Eugene Wu, and Philip J Guo. 2023. Teaching data science by visualizing data table transformations: Pandas Tutor for Python, Tidy Data Tutor for R, and SQL Tutor. In *Proceedings of the International Workshop on Data Systems Education: Bridging Education Practice with Education Research*. ACM, New York, NY, 50–55. https://doi.org/10.1145/3596673.3596972

[32] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The dawn of natural language to SQL: Are we fully ready? arXiv:2406.01265

[33] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Ben Zorn, Jack Williams, Neil Toronto, and Andy Gordon. 2023. "What it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, 31 pages. https://doi.org/10.1145/3544548.3580817

[34] Shusen Liu, Haichao Miao, Zhimin Li, Matthew Olson, Valerio Pascucci, and Peer-Timo Bremer. 2024. AVA: Towards autonomous visualization agents through visual perception-driven decision-making. *Comput. Graphics Forum* 43, 3, Article e15093 (2024), 12 pages. https://doi.org/10.1111/cgf.15093

[35] Shang-Ching Liu, ShengKun Wang, Tsungyao Chang, Wenqi Lin, Chung-Wei Hsiung, Yi-Chen Hsieh, Yu-Ping Cheng, Sian-Hong Luo, and Jianwei Zhang. 2023. JarviX: A LLM no code platform for tabular data analysis and optimization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Industry Track*. ACL, 622–630. https://doi.org/10.18653/v1/2023.emnlp-industry.59

[36] Lydia R Lucchesi, Petra M Kuhnert, Jenny L Davis, and Lexing Xie. 2022. Smallset Timelines: A visual representation of data preprocessing decisions. In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency (FAccT)*. ACM, New York, NY, 1136–1153. https://doi.org/10.1145/3531146.3533175

[37] Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. 2024. DirectGPT: A direct manipulation interface to interact with large language models. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 975, 16 pages. https://doi.org/10.1145/3613904.3642462

[38] Andrew M Mcnutt, Chenglong Wang, Robert A Deline, and Steven M. Drucker. 2023. On the design of AI-powered code assistants for notebooks. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 434, 16 pages. https://doi.org/10.1145/3544548.3580940
[39] Sahar Mehrpour and Thomas D. Latoza. 2023. A survey of tool support for working with design decisions in code. *Comput. Surveys* 56, 2, Article 37 (2023), 37 pages. https://doi.org/10.1145/3607868
[40] Meta Open Source. 2024. *React*. https://react.dev/
[41] Microsoft. 2024. *Monaco Editor*. https://microsoft.github.io/monaco-editor/
[42] Brad A Myers. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1, 1 (1990), 97–123. https://doi.org/10.1016/S1045-926X(05)80036-9
[43] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to help with code understanding. In *Proceedings of the ACM International Conference on Software Engineering (ICSE)*. IEEE, Article 97, 13 pages. https://doi.org/10.1145/3597503.3639187
[44] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo Ramos. 2021. DIY: Assessing the correctness of natural language to SQL systems. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*. ACM, New York, NY, 597–607. https://doi.org/10.1145/3397481.3450667
[45] Christina Niederer, Holger Stitz, Reem Hourieh, Florian Grassinger, Wolfgang Aigner, and Marc Streit. 2017. TACO: Visualizing changes in tables over time. *IEEE Trans. Vis. Comput. Graph.* 24, 1 (2017), 677–686. https://doi.org/10.1109/TVCG.2017.2745298
[46] Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation?. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 49 pages. arXiv:2306.09896 Poster.
[47] OpenAI. 2024. *Data analysis with ChatGPT*. OpenAI. Retrieved June 1, 2024 from https://help.openai.com/en/articles/8437071-data-analysis-with-chatgpt
[48] Pallets. 2024. *Flask*. https://flask.palletsprojects.com/en/3.0.x/
[49] Luca Podo, Muhammad Ishmal, and Marco Angelini. 2024. Toward a structured theoretical framework for the evaluation of generative AI-based visualizations. In *Proceedings of the EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 6 pages. https://doi.org/10.2312/eurova.20241118
[50] Xiaoying Pu, Sean Kross, Jake M. Hofman, and Daniel G. Goldstein. 2021. Datamations: Animated explanations of data analysis pipelines. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 467, 14 pages. https://doi.org/10.1145/3411764.3445063
[51] Dhivyabharathi Ramasamy, Cristina Sarasua, Alberto Bacchelli, and Abraham Bernstein. 2023. Visualising data science workflows to support third-party notebook comprehension: An empirical study. *Empirical Software Engineering* 28, 3 (2023), 58. https://doi.org/10.1007/s10664-023-10289-9
[52] Hua Shen, Tiffany Knearem, Reshmi Ghosh, Kenan Alkiek, Kundan Krishna, Yachuan Liu, Ziqiao Ma, Savvas Petridis, Yi-Hao Peng, Li Qiwei, et al. 2024. Towards bidirectional human-AI alignment: A systematic review for clarifications, framework, and future directions. arXiv:2406.09264
[53] Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2022. Towards natural language interfaces for data visualization: A survey. *IEEE Trans. Vis. Comput. Graph.* 29, 6 (2022), 3121–3144. https://doi.org/10.1109/TVCG.2022.3148007
[54] Dilruba Showkat and Eric P. S. Baumer. 2021. Where do stories come from? Examining the exploration process in investigative data journalism. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 390 (2021), 31 pages. https://doi.org/10.1145/3479534
[55] Nischal Shrestha, Titus Barik, and Chris Parnin. 2021. Unravel: A fluent code explorer for data wrangling. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. ACM, New York, NY, 198–207. https://doi.org/10.1145/3472749.3474744
[56] Nischal Shrestha, Bhavya Chopra, Austin Z Henley, and Chris Parnin. 2023. Detangler: Helping data scientists explore, understand, and debug data wrangling pipelines. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, Piscataway, NJ, 189–198. https://doi.org/10.1109/VL-HCC57772.2023.00031
[57] Hariharan Subramonyam, Roy Pea, Christopher Lawrence Pondoc, Maneesh Agrawala, and Colleen Seifert. 2024. Bridging the gulf of envisioning: Cognitive design challenges in LLM interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 1039, 19 pages. https://doi.org/10.1145/3613904.3642754
[58] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling multilevel exploration and sensemaking with large language models. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. ACM, New York, NY, Article 1, 18 pages. https://doi.org/10.1145/3586183.3606756
[59] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The metacognitive demands and opportunities of generative AI. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 680, 24 pages. https://doi.org/10.1145/3613904.3642902
[60] Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. 2024. ChartGPT: Leveraging LLMs to generate charts from abstract natural language. *IEEE Trans. Vis. Comput. Graph.* (2024). https://doi.org/10.1109/TVCG.2024.3368621 Early Access.
[61] Priyan Vaithilingam, Elena L. Glassman, Jeevana Priya Inala, and Chenglong Wang. 2024. DynaVis: Dynamically synthesized UI widgets for visualization editing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 985, 17 pages. https://doi.org/10.1145/3613904.3642639
[62] Bret Victor. 2011. *Up and down the ladder of abstraction: A systematic approach to interactive visualization.* Retrieved April 1, 2024 from http://worrydream.com/LadderOfAbstraction/
[63] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M. Drucker. 2022. Diff in the loop: Supporting data comparison in exploratory data analysis. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 97, 10 pages. https://doi.org/10.1145/3491102.3502123
[64] April Y Wang, Ryan Mitts, Philip J Guo, and Parmit K Chilana. 2018. Mismatch of expectations: How modern learning resources fail conversational programmers. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 511, 13 pages. https://doi.org/10.1145/3173574.3174085
[65] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *Proceedings of the International Conference on Machine Learning (ICML)*. Article PMLR 235, 13 pages. arXiv:2402.01030
[66] Yun Wang, Leixian Shen, Zhengxin You, Xinhuan Shu, Bongshin Lee, John Thompson, Haidong Zhang, and Dongmei Zhang. 2024. WonderFlow: Narration-centric design of animated data videos. *IEEE Trans. Vis. Comput. Graph.* (2024), 17 pages. https://doi.org/10.1109/TVCG.2024.3411575 Early Access.
[67] Luoxuan Weng, Xingbo Wang, Junyu Lu, Yingchaojie Feng, Yihan Liu, and Wei Chen. 2024. InsightLens: Discovering and exploring insights from conversational contexts in large-language-model-powered data analysis. arXiv:2404.01644
[68] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. AI Chains: Transparent and controllable human-AI interaction by chaining large language model prompts. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 385, 22 pages. https://doi.org/10.1145/3491102.3517582
[69] Yang Wu, Yao Wan, Hongyu Zhang, Yulei Sui, Wucai Wei, Wei Zhao, Guandong Xu, and Hai Jin. 2024. Automated data visualization from natural language via large language models: An exploratory study. *Proceedings of the ACM on Management of Data* 2, 3, Article 115 (2024), 28 pages. https://doi.org/10.1145/3654992
[70] Liwenhan Xie, Xinhuan Shu, Jeon Cheol Su, Yun Wang, Siming Chen, and Huamin Qu. 2024. Creating emordle: Animating word cloud for emotion expression. *IEEE Trans. Vis. Comput. Graph.* 30, 8 (2024), 5198–5211. https://doi.org/10.1109/TVCG.2023.3286392
[71] Liwenhan Xie, Zhaoyu Zhou, Kerun Yu, Yun Wang, Huamin Qu, and Siming Chen. 2023. Wakey-Wakey: Animate text by mimicking characters in a GIF. In *Proceedings of the Symposium on User Interface Software and Technology (UIST)*. ACM, New York, NY, Article 98, 14 pages. https://doi.org/10.1145/3586183.3606813
[72] Kai Xiong, Siwei Fu, Guoming Ding, Zhongsu Luo, Rong Yu, Wei Chen, Hujun Bao, and Yingcai Wu. 2022. Visualizing the scripts of data wrangling with SOMNUS. *IEEE Trans. Vis. Comput. Graph.* 29, 6 (2022), 2950–2964. https://doi.org/10.1109/TVCG.2022.3144975
[73] Chenyang Yang, Shurui Zhou, Jin LC Guo, and Christian Kästner. 2021. Subtle bugs everywhere: Generating documentation for data wrangling code. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Piscataway, NJ, 304–316. https://doi.org/10.1109/ASE51524.2021.9678520
[74] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Oleksandr Polozov, and Charles Sutton. 2023. Natural language to code generation in interactive data science notebooks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL, Toronto, Canada, 126–173. https://doi.org/10.18653/v1/2023.acl-long.9
[75] Chen Zhu-Tian and Haijun Xia. 2022. CrossData: Leveraging text-data connections for authoring data documents. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, Article 95, 15 pages. https://doi.org/10.1145/3491102.3517485
[76] Chen Zhu-Tian, Zeyu Xiong, Xiaoshuo Yao, and Elena Glassman. 2024. Sketch then generate: Providing incremental user feedback and guiding LLM code generation through language-oriented code sketches. arXiv:2405.03998
[77] Chen Zhu-Tian, Chenyang Zhang, Qianwen Wang, Jakob Troidl, Simon Warchol, Johanna Beyer, Nils Gehlenborg, and Hanspeter Pfister. 2024. Beyond generating code: Evaluating GPT on a data visualization course. arXiv:2306.02914