Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition

Majeed Kazemitabaar
University of Toronto
Toronto, Ontario, Canada
majeed@dgp.toronto.edu

Jack Williams
Microsoft Research
Cambridge, UK
jack.williams@microsoft.com

Ian Drosos
Microsoft Research
Cambridge, UK
t-iandrosos@microsoft.com

Tovi Grossman
University of Toronto
Toronto, Ontario, Canada
tovi@dgp.toronto.edu

Austin Z. Henley
Microsoft Research
Redmond, Washington, USA
austinhenley@microsoft.com

Carina Negreanu
Microsoft Research
Cambridge, UK
cnegreanu@microsoft.com

Advait Sarkar
Microsoft Research
Cambridge, UK
advait@microsoft.com

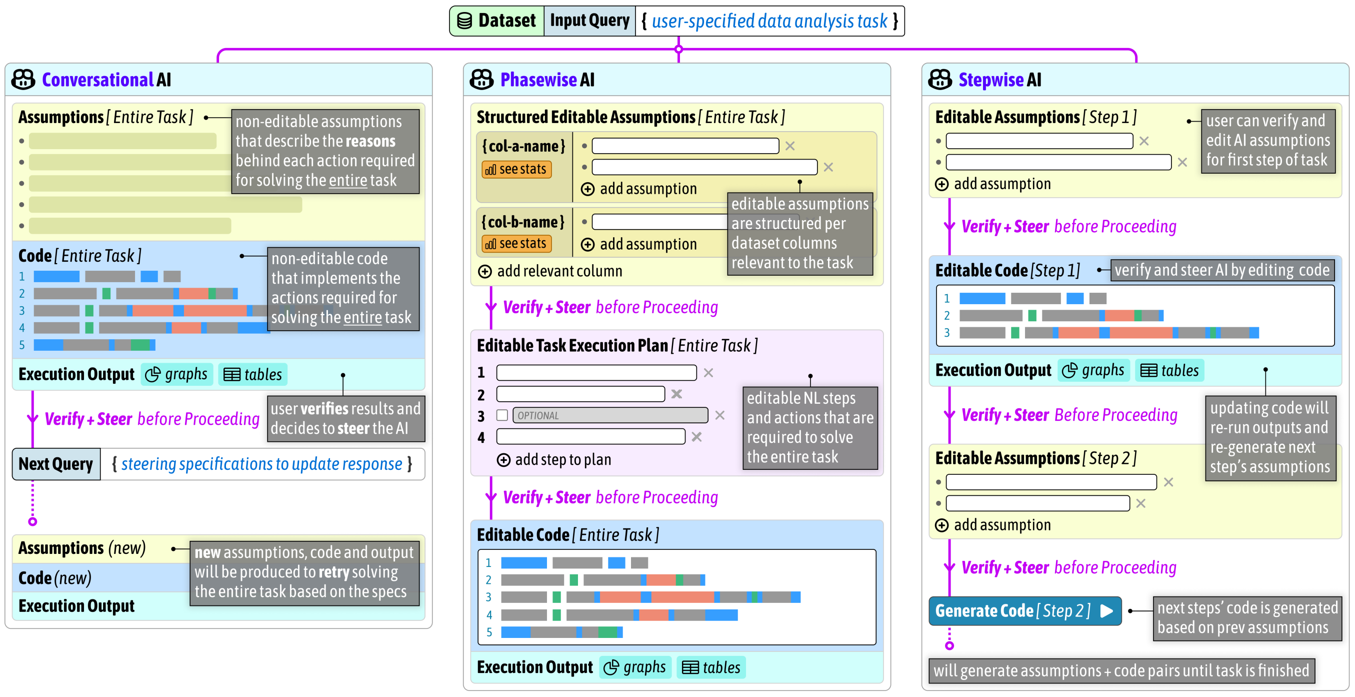

Dataset
Input Query
{ user-specified data analysis task }
Conversational AI
Assumptions [Entire Task]
non-editable assumptions that describe the reasons behind each action required for solving the entire task
Code [Entire Task]
non-editable code that implements the actions required for solving the entire task
Execution Output
graphs
tables
Verify + Steer before Proceeding
user verifies results and decides to steer the AI
Next Query
{ steering specifications to update response }
Assumptions (new)
Code (new)
Execution Output
new assumptions, code and output will be produced to retry solving the entire task based on the specs
Phasewise AI
Structured Editable Assumptions [Entire Task]
{ col-a-name }
see stats
add assumption
{ col-b-name }
see stats
add assumption
editable assumptions are structured per dataset columns relevant to the task
add relevant column
Verify + Steer before Proceeding
Editable Task Execution Plan [Entire Task]
1
2
3
OPTIONAL
4
add step to plan
editable NL steps and actions that are required to solve the entire task
Verify + Steer before Proceeding
Editable Code [Entire Task]
Execution Output
graphs
tables
Stepwise AI
Editable Assumptions [Step 1]
add assumption
user can verify and edit AI assumptions for first step of task
Verify + Steer before Proceeding
Editable Code [Step 1]
verify and steer AI by editing code
Execution Output
graphs
tables
Verify + Steer Before Proceeding
updating code will re-run outputs and re-generate next step's assumptions
Editable Assumptions [Step 2]
add assumption
Verify + Steer before Proceeding
Generate Code [Step 2]
next steps' code is generated based on prev assumptions
will generate assumptions + code pairs until task is finished

Figure 1: An illustration of the three decomposition approaches that we developed for solving data analysis tasks using AI: (A) Conversational approach solves the entire task without any user intervention but allows submitting follow-up prompts for further steering. (B) Stepwise approach provides intervention points at each *step* of solving the task by presenting pairs of editable assumptions followed by corresponding code at each step. (C) Phasewise approach provides intervention points at each *phase* of solving the entire task with three editable components: editable assumptions of the entire task structured around relevant columns of the dataset, editable task execution plan, and corresponding code.

Abstract

LLM-powered tools like ChatGPT Data Analysis, have the potential to help users tackle the challenging task of data analysis programming, which requires expertise in data processing, programming, and statistics. However, our formative study (n=15) uncovered serious challenges in verifying AI-generated results and steering the AI (i.e., guiding the AI system to produce the desired output). We developed two contrasting approaches to address these challenges. The first (**Stepwise**) decomposes the problem into step-by-step subgoals with pairs of editable assumptions and code until task completion, while the second (**Phasewise**) decomposes the entire problem into three editable, logical phases: structured input/output assumptions, execution plan, and code. A controlled, within-subjects experiment (n=18) compared these systems against a conversational baseline. Users reported significantly greater control with the Stepwise and Phasewise systems, and found intervention, correction, and verification easier, compared to the baseline. The results suggest design guidelines and trade-offs for AI-assisted data analysis tools.

CCS Concepts

• **Human-centered computing → Natural language interfaces**; **Interactive systems and tools**; **Empirical studies in HCI**.

Keywords

Data Analysis, Data Science Assistant, Human-AI Interaction, AI Agents, Generative AI, Large Language Models, Copilot

ACM Reference Format:
Majeed Kazemitabaar, Jack Williams, Ian Drosos, Tovi Grossman, Austin Z. Henley, Carina Negreanu, and Advait Sarkar. 2024. Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition. 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, 19 pages. https://doi.org/10.1145/3654777.3676345

1 Introduction

Data science often involves large datasets, source code, domain expertise, and unwritten assumptions [68]. The process of extracting insights from data [94] for decision making and knowledge discovery [20] has several documented challenges [8, 68]. Data scientists spend considerable time inspecting data, writing single-use scripts, "gluing together" data sources, cleaning messy data, and documenting their efforts [8, 47, 48, 68]. In fact, data scientists describe the need to "have a conversation" with their data to understand it [68].

Recent advancements in AI and particularly the natural language processing and code generation capabilities of Large Language Models (LLMs) have shown promise to facilitate data science tasks. Specifically, chain-of-thought prompting [96] and ReAct prompting [100] have emerged as implementation techniques for *task decomposition*, generating *"reasoning"* traces, followed by *"acting"* traces where the LLM can invoke external agents. In data science, such agents might read from datasets and execute code. For example, ChatGPT Data Analysis supports uploading CSV files, after which its language model and code execution agents can be used to clean data, display visualizations, and answer questions about data through its iterative chat interface [70].

However, a study of data scientists using ChatGPT *without* code execution functionality found that participants were unaware of the AI's assumptions when solving the task, found verifying the correctness of results tedious, were overwhelmed by long responses, and could not effectively steer the AI when it made mistakes [10].

To better understand user behavior, needs, and challenges when performing exploratory data analysis with a conversational AI tool, we conducted a formative study involving 15 participants (Section 3). The study identified *steering* and *verification* as the primary limitations of conversational AI tools. *Steering* refers to the user's interaction with the AI to guide its output from an initial state to a desired outcome. *Verification* refers to the user's interaction to understand the AI's output, check its correctness, ensure no incorrect assumptions were used, and decide on further refinement. The study also highlighted the need for new affordances that decompose and display the AI's chain-of-thought reasoning as structured and interactive assumptions, enabling users to modify them at any time, even retroactively.

Based on this core requirement we developed two systems that make "decomposition" a focal point in the interface, not just an implementation detail (Section 4). First, the Phasewise system decomposes the problem into three editable phases (assumptions, planning, and code) with increasing levels of specificity. Second, the Stepwise system decomposes the task into separate steps (visually similar to a computational notebook), displaying editable assumptions and their corresponding code one step at a time until the task is complete.

Both approaches use the LLM to decompose the task into parts, and help the user focus on one part at a time (as a metacognitive aid [89]). This enables finer-grained steering than standard conversational prompting [103], and progressive disclosure to reduce information overload [69]. We introduce the idea of using the LLM to generate *editable assumptions* about the input and desired output, based on the task query and data. This also provides a structure for verifying that the AI correctly interpreted the user's intent and translated it into a valid plan. However, when decomposing a task, there is an important space of trade-offs: how much information to display, when to display it, and how many intervention points to provide with how much control. Our two systems occupy different points in this space, and through our user study, we evaluated the trade-offs and identified different situations in which each approach might be beneficial (Sections 5 and 6).

We conducted a controlled, within-subjects experiment (n=18) comparing the Stepwise and Phasewise tools on task decomposition and prompting strategies that support steering, verification, and the user's perceived utility of the tools. We also developed a baseline tool, called Conversational, similar to ChatGPT with code execution. Users reported significantly greater control with the two new systems, and found intervention, correction, and verification easier, compared to the baseline.

This paper makes the following contributions:

- A formative study that identifies the limitations of *"conversational"* AI tools in terms of steering them and verifying their output.
- A novel approach to improve steering and verification using editable AI assumptions, progressive disclosure, and non-linear conversations to promote data exploration and verification. We describe two implementations of this approach, each balancing information overload and the degree of user control differently (Section 4).
- A within-subjects experiment in which we compared the two systems with a Conversational baseline system (Section 5), finding that while there was no difference in task success or completion time, participants felt significantly more in control with the Phasewise and Stepwise systems compared to the baseline (Section 6).

2 Related Work

2.1 AI-assisted Data Analysis

Previous work has considered how data transformation scripts can be synthesized from demonstrations (e.g., Wrangler [32, 41]). This follows an influential line of research that synthesizes programs from examples in data wrangling contexts (e.g., [30]), which may include natural language [31]. These can be constrained to use specific APIs such as `pandas`, using generator-based synthesis (e.g., AutoPandas [5]). Scripts can also be synthesized based on heuristics of data quality improvement (e.g, CoWrangler [9]), and data preparation heuristics can also be learned from corpora (e.g., Auto-Suggest [99]).

More recently, a number of commercialized LLM supported data analysis tools have become available. These enable data scientists to access AI-powered chat assistants within their notebook (such as Anaconda [2], Databricks [14], and Jupyter AI [38]), and other alternate data-science environments (e.g., DataChat AI [15], SQL and file editors for Databricks, etc.). The semantic abilities of LLMs, coupled with a chat interface, allows conversational interaction with data, follow-up questions, and highly contextualized responses. Consequently, research has investigated the chatbot paradigm for AI assistance in data analysis and visualization in detail [18, 28, 35, 43, 84, 105].

Thus, early work on data wrangling script synthesis can be contrasted with current LLM-powered data analysis tools both in terms of the complexity of tasks being tackled, and the interaction modality (i.e., from demonstration, examples, and direct manipulation, to naturalistic language prompts). In turn, this also means that generation mistakes become more common, due to underspecification of natural language, assumptions that the AI is making but the user is not aware of, etc. This creates new metacognitive demands for the user to verify the AI's responses and then steer the AI if incorrect. In our work, we try to provide new interaction modalities with LLMs for data analysis tasks to increase the transparency of the AI and the assumptions that it is making.

McNutt et al. [65] present a design space for AI code assistance in computational notebooks, which are commonly used for data analysis. They find that AI assistants can vary in the gestures they provide for the user to initiate a model response, and options that the user has to verify and refine the output. They also consider the relationship between the assistant interface and other interface components, such as code context, specialization, provenance, and customization.

Though not specifically tackling data analysis, Sarkar et al. [81] studied the experiences of programmers using LLM assistance for writing code. They found that LLM assistance was most useful in rewriting boilerplate code and in API discovery, but also brought new challenges for debugging and code inspection. Sarkar et al. also identify prompt formulation as a major challenge. Fiannaca et al. [23] describe methods for how this can be improved by leveraging semantically meaningful structure within prompts to assist programmers.

Similarly, Vaithilingam et al. [91] found that while programmers preferred LLM-assisted programming to unassisted programming, there were no consistent improvements in task time or success rate, due to productivity benefits being opposed by new challenges in debugging and comprehension of AI-generated code. A detailed telemetric study of GitHub Copilot usage by Mozannar et al. [67] similarly found that the "verifying suggestion" state is the most time consuming.

There have been other studies of AI-assisted programming [4, 19, 40, 56]. Many of these point to steering and verification as general challenges with all LLM-assisted programming, which also apply in the specific case of programming data analysis scripts with LLM assistance.

However, it is worth noting that data analysis does have particularities in comparison to "general" programming tasks, e.g., data analysis programming tends to be more exploratory and open-ended, and the activity of analyst sensemaking and insight generation is more important than providing code as a finished product [22, 27, 42, 47, 54, 61, 73, 76, 77]. The implication of this is that the need for rapid steering and verification is more acute in data analysis programming, since the effectiveness of the process depends on rapid exploration of the program space.

2.2 Verifying LLM Outputs and their Reliability

Data science is a challenging yet important function within software teams. Previous research has focused on how data scientists engage in collaborative sensemaking, and make choices about how to communicate and report results [11, 50, 51, 71, 93, 104]. They have found that data scientists need support in managing these complex collaborative workflows [46, 48, 95]. Consequently, research has explored how data scientists can manage, visualize, and trace the evolution of their analysis process [33, 45, 74, 97, 98].

Working with an AI assistant may have important differences from a human team. Trust in AI systems is developed differently from trust in human collaborators and is mediated by the conceptual metaphors used to convey them [37, 49]. Trust, communication, and perception management in human collaborations may result in a lack of code verification behaviours, or selective sharing [17, 61, 65, 71, 93]. This raises the importance of additional tools for verifying AI generated output, for instance "co-audit" tools [25].

Previous research has noted the importance of explainability in AI to support user tasks and decision-making [52, 57, 58]. There are difficulties in applying traditional explainability techniques to large language models due to their large number of parameters, training sets, and complex and open-ended space of inputs and outputs [19, 79, 88, 92]. Research has also explored the challenges non-experts face in prompting and conversational strategies for explainability [3, 55, 103].

Furthermore, previous research on AI coding assistants emphasizes that engaging users in verifying the *process* of how the AI generates code can maximize user experience, efficiency, and the predictability of obtaining a helpful response [44]. For instance, CodeHelp [59] incorporates a *Sufficiency Check* step that engages users in refining the AI's understanding of the task by prompting them to clarify uncertainties or provide missing context.

Gu et al. [29] created a design probe similar to ChatGPT Data Analysis [70] with an added sidebar for inspecting intermediary variables. They conducted a study with 22 participants using the design probe to understand common behaviors for verifying the AI's response to a natural language query and dataset. They found two main behaviors within the verification workflow: *procedural-oriented* and *data-oriented* which in many cases were tightly coupled and participants frequently switched between an intermediary variable and the code that outputted it. In contrast with how data analysts verify their work in any tool-assisted (non-AI) data analysis workflows, there is now a much bigger demand for verification when users "offload" an entire data analysis task to an LLM.

In our work, we explicitly ask the AI to show its assumptions and reasoning in a structured (and editable) way, paired with their corresponding actions, so that users could focus on them and make decisions. We also include features such as "side queries" that allow users to pose exploratory questions, build up assumptions, and then add those assumptions to the AI generation workflow.

2.3 Steering LLMs

Currently, most commercial LLM tools (e.g., ChatGPT Data Analysis [70] or ChatGPT with Noteable [1]) use a turn-based conversational method, where the AI attempts to solve problems with minimal intervention points. Typically, users can only steer the AI after it has generated an entire solution, using follow-up prompts, which limits steering control. To address such limitations, Masson et al. [63] propose principles of direct manipulation [36] for steering LLMs in other contexts: continuous representation of objects of interest, physical actions to localize prompt effects, and reusable prompts. Furthermore, research on the metacognitive demands of generative AI identifies decomposition and structured generation as potential aids [89]. Suh et al. [87] explore hierarchical text generation at different abstraction levels to assist with sensemaking and managing information overload from large text quantities. They also introduce *structured generation* [86], where user's prompt is first used to generate dimensions that make the model's responses vary, and then responses are generated according to those dimensions.

Specifically in the context of AI-assisted programming, Liu and Sarkar et al. [60] introduce "grounded abstraction matching," allowing users to steer LLMs by editing natural language utterances grounded in each step of AI-generated code for data analysis in spreadsheets. Similarly, Tian et al. developed Steps, which lets users edit step-by-step explanations of AI-generated SQL code from natural language queries [90]. CoLadder [101] aids experienced programmers in externalizing their problem-solving intentions flexibly, enhancing their ability to evaluate and modify code across various abstraction levels, from goal to final code implementation. These methods enable users to edit natural language prompts grounded in each step of AI-generated code, providing an accessible abstraction level for reading, verifying, and editing.

However, these approaches hide the AI's reasoning and decomposition process, leaving users without insight into the "how" and "why" behind the generated code. Users are left to manually infer the AI's reasoning from the output and determine explicit actions to edit and refine the grounded utterances. While this might be an acceptable trade-off in systems that generate short programs (e.g., typical spreadsheet formulas or SQL queries), it is unclear how this approach would extend to longer and more complex data analysis scripts. Our work expands this design space by not only displaying the AI's assumptions in a structured way but also enable users to directly edit these assumptions as a novel method of steering the AI to control its output.

3 Formative Study

To explore the challenges of data analysis with conversational AI assistants, we conducted a formative study with 15 participants (12 male, 3 female, 0 non-binary) using the Noteable plugin for ChatGPT [1]. At the time of the study, Noteable was the only publicly available tool offering features similar to ChatGPT Data Analysis (formerly Code Interpreter). With Noteable, participants could upload datasets to a Noteable project and enter a natural language (NL) descriptions of their data analysis task in ChatGPT. In response, ChatGPT would generate code cells in the Noteable project, which Noteable would execute. ChatGPT then displayed Noteable's output including any tables or visualizations, and generated an interpretation of the results. ChatGPT would then continue generating code if the task was incomplete, or asked users for additional information if required.

Participants (F1-F15), who were recruited from our research institute, regularly performed data analysis tasks using computational notebooks and Python data science libraries. Each participant was assigned to one of four tasks commonly performed by data scientists: data cleaning, merging and plotting, extracting insights, or training an ML model (see Table 1).

Study sessions lasted approximately 60 minutes and were conducted in-person. Screen activity was recorded. Participants were asked to think aloud [24], and audio data was recorded and transcribed. Participant consent was obtained prior to the study and participants were each compensated with a GBP £25 Amazon gift card. The study protocol was approved by our institution's ethics and compliance review board.

3.1 Results

We analyzed the interactions participants had with ChatGPT and the Noteable plugin from 301 total prompts. Participants used a mix of different actions which included (1) directing the AI to perform a data analysis task, (2) exploring the dataset, (3) requesting suggested

Table 1: Tasks used in the formative study in which users used ChatGPT with the Noteable plugin.

Task	Dataset	Task Description	Assigned Participants
Data Cleaning	`food-choices.csv`	Fix columns with inconsistent formatting and prepare dataset for analysis.	F5, F8, F9
Merging and Plotting	`country-happiness.csv`	Merge the five happiness datasets on the country column and visualize the top countries with the moist changes in happiness score from 2015 to 2019.	F10, F11, F12, F13
Extracting Insights	`airbnb-nyc.csv`	Extract high-level spatial and temporal insights about price, availability, and distribution.	F2, F3, F7, F15
Training Model	`heart-disease.csv`	Train an explainable ML model to predict presence of heart disease and report the key factors contributing to the presence or absence of heart disease in patients.	F1, F4, F6, F14

methods or approaches to accomplish the task, (4) steering and repairing the AI process in how it should accomplish the task, and (5) performing verification on the results of the task (either with or without the AI).

Steering: Participants steered the AI's actions and methods using their NL prompts (106 prompts, 35%). Many of the steering prompts (n=34) were for performing data wrangling (cleaning and manipulation) tasks on specific columns of the dataset. Similarly, some prompts (n=20) were used to explicitly add, remove, or change code produced in previous steps (e.g., "`exclude the ones that are purely categorical`" (F5)). For repairing mistakes the AI made or any miscommunications between human and AI, participants frequently corrected an assumption the AI had made (36 prompts). For example, after ChatGPT generated data analysis code, F8 prompted ChatGPT that they wanted code that could "`map each row to multiple classes and not just one closest class`" instead.

Data exploration: We identified 76 instances (25%) in which participants wanted to inspect the data frames loaded into the notebook using natural language filters such as displaying "`which country names are inconsistent`" (F12), and "`unique values in GPA column`" (F8). Sometimes these explorations were in the form of visualizations, (e.g., requesting a "`histogram of cholesterol levels`" (F6)).

Verification: Although the most common behavior for validating the AI's process was reading the AI-generated code and inspecting the output, we also categorized 57 prompts (19%) as assisting with verification, such as: "`The USA is missing from all these heat maps, is it also missing from the CSV files or not?`" (F11).

Code or Logic Explanation: In 21 prompts (7%), participants used the main thread of the conversation to ask the AI for explanations about code they did not understand, an algorithm that was used, how something was computed, or help interpreting the results.

Furthermore, our results indicate that each participant engaged in linear conversations consisting, on average, 20 AI-generated messages (SD=5). They experienced lengthy responses upon each interaction point, averaging 24 lines of AI-generated code (SD=21, Max=152) and 134 words of the AI's interpretation of the output (SD=104, Max=717). Participants often lost track of the long conversation history and struggled with finding, verifying, and fixing accumulated assumptions. As a result, they requested an "undo button" to fix accumulated assumptions made by the AI. Without this feature, they tried workarounds, asking the AI to "`undo the last step`" (F9) or to "`ignore the previous data cleansing steps and do it from scratch`" (F5).

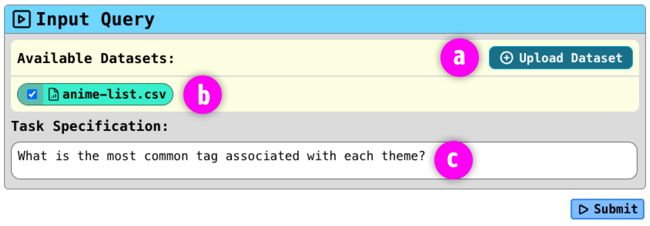
3.2 Design Goals and Rationale

Our formative study highlighted *steering* and *verification* as the most common user interactions. Based on our findings, along with relevant prior work and how such AI tools use chain-of-thought prompting for task decomposition and execution, we established the following design goals to enhance user control over the AI-assisted data analysis process.

First, participants struggled to understand the AI's reasoning. They often tried to manually infer the underlying assumptions from its generated code, verify them, and then correct them with follow-up prompts. This was evident from the 36 prompts that they used to explicitly fix incorrect assumptions made by the AI. Therefore, ***DG.1*** *proposes visually separating each different assumption from its corresponding actions (code) and allowing users to directly edit and update them.*

Second, participants were overwhelmed by long responses and lost track of the long conversation history. Consistent with previous studies [10], ***DG.2*** *recommends adding intervention points in the AI's responses to help users focus on smaller information chunks. Additionally, steering operations should update only relevant sections at each intervention point, rather than adding new outputs to the main thread.*

Lastly, participants frequently used prompts for data exploration, result verification, and code explanation. While these were useful, they often derailed the main conversation thread from solving the task. To address this, ***DG.3*** *suggests enabling side conversations and other methods to assist users in verifying assumptions without cluttering the main thread.*



Input Query

Available Datasets:

anime-list.csv

Task Specification:

What is the most common tag associated with each theme?

Figure 2: The starting input for all systems, which includes a button to upload datasets (a), selectable datasets to be included in the analysis (b), and a text input for entering the natural language description that specifies the data analysis task (c).

4 System Design

We address design goal **DG.1** through *interactive task decomposition*. This involves: (1) prompting the LLM to generate its chain-of-thought reasoning as NL assumptions and corresponding actions about the input task and dataset; and (2) rendering the LLM's output as structured, editable UI components, allowing users to refine the AI's proposed plan. We refer to these as *editable assumptions* that represent the AI's reasoning based on the task and dataset (e.g., pattern of values in a dataset column.)

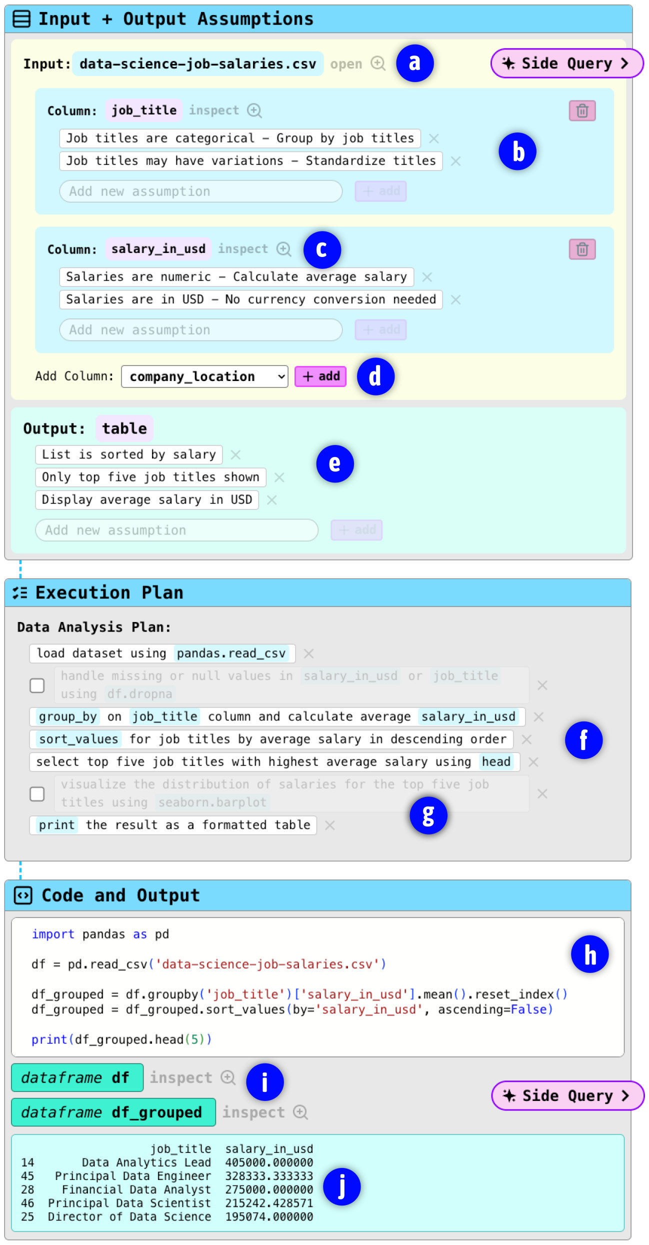
In addressing **DG.2** to provide proper intervention points, we encounter trade-offs in balancing the number, amount of information presented, and the degree of control provided at each intervention point. This led us to develop two alternative approaches. The **Phasewise** system, which gives users greater control over the entire analysis plan from the outset, but with fewer intervention points, and requires the user to understand more information at each step. Conversely, the **Stepwise** allows more focused control by decomposing the task into step-by-step subgoals, increasing intervention opportunities, reducing the information overload per step, but with less structured control over the entire task.

Similarly, to address **DG.3**, we balanced the amount of information displayed for verification and decision making at each intervention point. The Phasewise system aids AI-based information retrieval by displaying relevant dataset columns, allowing inspection of column statistics and AI assumptions, as well as distinguishing required and suggested steps in the execution plan. To support user-led exploration in both systems, we allocated a "sidebar" on the ride side of the screen where users can: (a) select portions of the AI-generated code and ask questions about them; (b) ask natural language queries for data exploration; and (c) generate code from natural language description for the user to manually incorporate into the AI-generated code.

In the following, sections we outline the core features shared between the Phasewise and Stepwise systems, and explain how they differ. Lastly, we describe the Conversational system, serving as a baseline similar to ChatGPT's Advanced Data Analysis plugin for our user evaluation.

4.1 Core System Features

4.1.1 Task Input. Data analysis begins with dataset(s) and a task specification. *Dataset Input:* Data is loaded using the *Input Query*



Input + Output Assumptions

Input: data-science-job-salaries.csv

Column: job_title
- Job titles are categorical - Group by job titles
- Job titles may have variations - Standardize titles

Column: salary_in_usd
- Salaries are numeric - Calculate average salary
- Salaries are in USD - No currency conversion needed

Add Column: company_location

Output: table
- List is sorted by salary
- Only top five job titles shown
- Display average salary in USD

Execution Plan

Data Analysis Plan:
- load dataset using pandas.read_csv
- [ ] handle missing or null values in salary_in_usd or job_title using df.dropna
- group_by on job_title column and calculate average salary_in_usd
- sort_values for job titles by average salary in descending order
- select top five job titles with highest average salary using head
- [ ] visualize the distribution of salaries for the top five job titles using seaborn.barplot
- print the result as a formatted table

Code and Output

```python
import pandas as pd

df = pd.read_csv('data-science-job-salaries.csv')

df_grouped = df.groupby('job_title')['salary_in_usd'].mean().reset_index()
df_grouped = df_grouped.sort_values(by='salary_in_usd', ascending=False)

print(df_grouped.head(5))
```

dataframe **df** inspect

dataframe **df_grouped** inspect

```
                  job_title  salary_in_usd
14       Data Analytics Lead  405000.000000
45   Principal Data Engineer  328333.333333
28    Financial Data Analyst  275000.000000
46  Principal Data Scientist  215242.428571
25  Director of Data Science  195074.000000
```

Figure 3: Overview of the Phasewise system's task flow, which decomposes tasks into three stages. *Input + Output Assumptions*, allows users to upload a dataset (a), manage column-based AI assumptions (b), inspect column-based descriptive statistics (c), add columns missed by the AI (d), and edit assumptions about the task's output (e). *Execution Plan* contains the AI's editable natural language plan for solving the task (f), which includes user-selectable optional steps (g). *Code and Output* contains AI generated code for solving the task and includes an code editor (h), intermediary variable inspector (i), and the code output (j). Section 4.1.2 details these features.

interface (Figure 2, a). Users can then select one or more datasets relevant to the task (Figure 2, b). The Python server generates a summary of each selected dataset with sample values for all columns. This summary is passed to the LLM to build its initial set of assumptions for the data analysis task.

Task Specification Input: After selecting relevant datasets, users specify their data analysis task in the text field and press submit to start the data analysis process (Figure 2, c).

4.1.2 Task Decomposition: Phasewise System. Using the summary of the dataset and the user-specified task description, the Phasewise system decomposes the task into three phases: Input and Output Assumptions, Execution Plan, and Code and Output.

***A)** Input and Output Assumptions:* After loading a dataset (Figure 3, a), this component displays all the columns that the AI found to be relevant to the task, and for each column it displays several editable assumptions regarding the task (Figure 3, b). Assumptions can pertain to data type, uniformity, units, sorting order, etc. Users can delete columns they find unrelated to the task, or add columns that the AI incorrectly did not select (Figure 3, d). Within each column, users can edit, add, or remove assumptions for that column (Figure 3, b). For each column, users can "inspect" descriptive statistics (Figure 3, c), including a frequency table of sample values for categorical columns. Additionally, the entire dataset can be viewed by clicking on the "open" button a, with the selected columns highlighted to help the user leverage the columns to build up assumptions. Finally, the task's output assumptions can be viewed and changed to edit, add, or remove assumptions to steer the final output (Figure 3, e).

***B)** Execution Plan:* Using the assumptions, including edits, the system generates a list of natural language steps for solving the task (Figure 3, f). Steps are editable and the user may add or remove steps. The model is also prompted to include optional steps that are rendered as selectable steps with a checkbox (Figure 3, g). After the user is satisfied with the plan, they can proceed to generating and running the code.

***C)** Code and Output:* Here the AI generates code to solve the task based on the previous two components. The code is immediately executed and displayed in an editor to allow modification and re-execution (Figure 3, h). Users can inspect the dataframe and variables used in the code execution (Figure 3, i), and see the code output (Figure 3, j)

4.1.3 Task Decomposition: Stepwise System. Unlike the Phasewise system where each component reflects the entire task, the Stepwise system decomposes the task into subgoals, which have intermediate objectives. Each subgoal (except the first, which loads the dataset (Figure 4, a)), is represented as a pair of components: Subgoal Assumptions and Actions, and Subgoal Code and Output.

***A)** Subgoal Assumptions and Actions:* Each subgoal starts with a short description of its objective in natural language, followed by several assumptions and actions based on the dataset or previous steps (Figure 4, b). We designed LLM prompts so that each subgoal would focus on one specific objective such as pre-processing data, filtering columns, performing calculations, and displaying plots. Users may reorder assumptions and actions to change their priority, add or remove assumptions, and edit them directly. Once the user is satisfied with them, they can proceed to generate the subgoal code and output.

***B)** Subgoal Code and Output:* Similar to Code and Output in the Phasewise system, in this component, the system generates code to solve the task based on the previous assumptions and actions (Figure 4, c). The code is immediately executed and can be edited. Once executed, the system generates the next subgoal to allow the user to either reflect on the current subgoal or start working on the next. This process continues until the task is finished and the requirements have been satisfied, in which case the next *Subgoal Assumptions and Actions* will indicate completion. However, if the user still wants to continue, they can add assumptions and actions to continue.

4.1.4 Editable LLM Assumptions and Actions. We prompted the LLM to generate each assumption paired with its corresponding action in the format of `<assumption> - <action>`. We also prompted the LLM to enclose column names, variables, and keywords in backticks, which could be rendered into editable components highlighted with a different color. The aim of these interventions was to reduce information overload and improve the efficiency of the editing process.

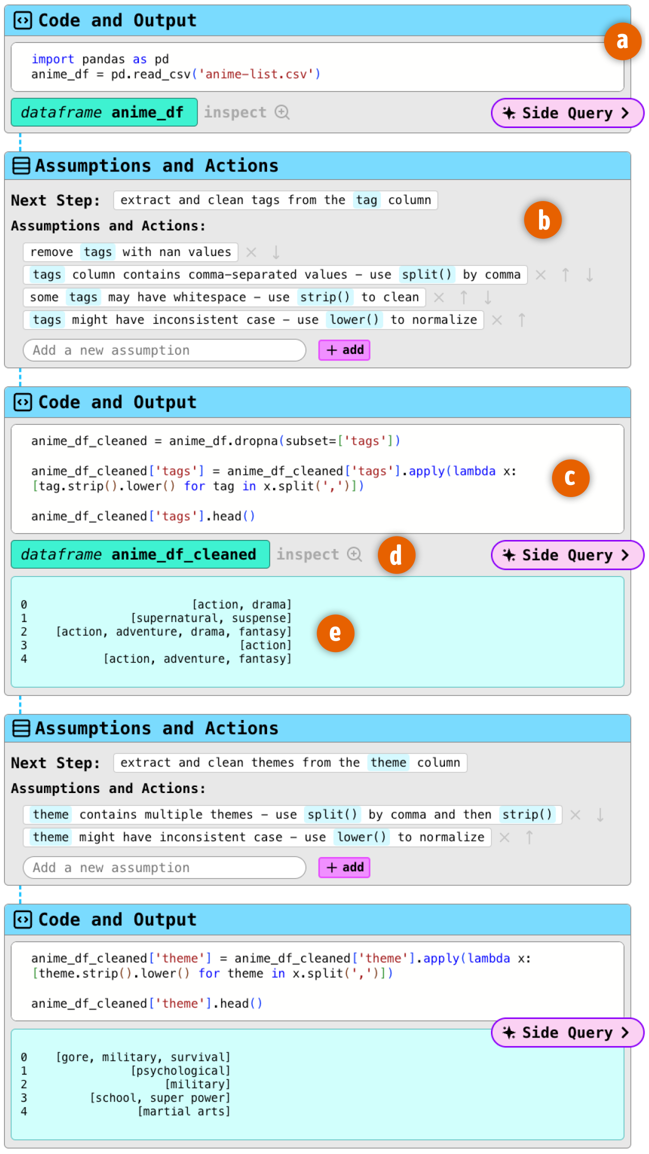
4.1.5 Code Execution and Intermediary Variables. The Python server runs the AI-generated code and returns any outputs, including text, visualization plots, or any errors. Any variables and dataframes created during execution are displayed as intermediary variables that the user may inspect.

Inspect Intermediary Variables: users can click on each intermediary variable to open a full-screen window for inspecting its values. For dataframes, the interface includes a string matching filter to assist users in finding specific values.

4.1.6 Managing Edits. Within each component, edits can be either pending or submitted. A submitted edit means that the edits have been applied to either generate the next component or regenerate downstream components, whereas a pending edit has not. Pending edits can be reverted using an undo button. However, once an edit is submitted, our system introduces a new branch to preserve the original, unedited version, while incorporating the edited version in the "main" branch. New branches are displayed in a tabbed ribbon at the top of the UI as shown in Figure 5. To allow iteration while minimizing proliferation of branches, new branches are not created when the user edits the last generated component in the stream of components. Branching allows users to keep track of previous edits and switch between edits as needed.

4.1.7 Side Conversations. We allocated space to the right of the main components for running side conversations with the system in three formats: *Ask Question*, *Generate Code*, and *Run Side Query*. These features are available in all editable code execution blocks, with the exception of *Run Side Query*, which is also accessible alongside the Input and Output Assumptions in the Phasewise AI system.

Ask Question: This allows users to ask questions about the generated code (See Figure 6). When a code editor is in focus or code is selected, the Ask Question button appears to the right of the editor. The user can provide a natural language query and the system



```
import pandas as pd
anime_df = pd.read_csv('anime-list.csv')
```

Next Step: extract and clean tags from the tag column

Assumptions and Actions:

- remove tags with nan values
- tags column contains comma-separated values - use split() by comma
- some tags may have whitespace - use strip() to clean
- tags might have inconsistent case - use lower() to normalize

```
anime_df_cleaned = anime_df.dropna(subset=['tags'])

anime_df_cleaned['tags'] = anime_df_cleaned['tags'].apply(lambda x:
[tag.strip().lower() for tag in x.split(',')])

anime_df_cleaned['tags'].head()
```

```
0                   [action, drama]
1          [supernatural, suspense]
2  [action, adventure, drama, fantasy]
3                          [action]
4        [action, adventure, fantasy]
```

Next Step: extract and clean themes from the theme column

Assumptions and Actions:

- theme contains multiple themes - use split() by comma and then strip()
- theme might have inconsistent case - use lower() to normalize

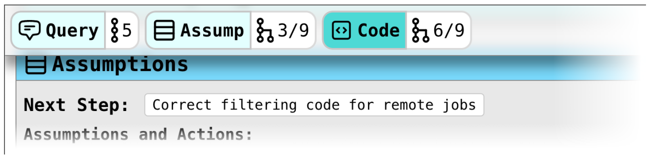
```
anime_df_cleaned['theme'] = anime_df_cleaned['theme'].apply(lambda x:
[theme.strip().lower() for theme in x.split(',')])

anime_df_cleaned['theme'].head()
```

```
0    [gore, military, survival]
1               [psychological]
2                    [military]
3         [school, super power]
4                [martial arts]
```

Figure 4: Overview of the STEPWISE system's task flow, which decomposes each task into subgoals containing two components. *Assumptions and Actions* includes the NL subgoal and editable AI assumptions b. *Code and Output* contains a code editor a c, a dataframe and variable inspector d, and the code output e. Section 4.1.3 details these features.

generates a response on the side. The selection allows users to ask targeted questions such as "what does this function do?"

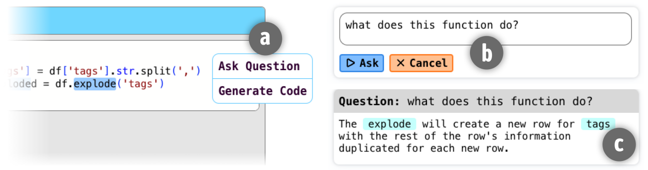
Generate Code: This feature generates code based on the selected code segment and the user's query. The user can inspect the generated code and, if it is found useful, insert it into the editor. Similar to the *Ask Question* feature, the selection here enables asking targeted questions, such as updating the code to exhibit a different behavior based on a natural language prompt.



Next Step: Correct filtering code for remote jobs

Assumptions and Actions:

Figure 5: The tabbed ribbon displays all the branches created after editing various nodes. Users can select a different tab to switch to that branch. Each tab indicates where the edit occurred and how much it has progressed (number of total nodes).



```
s'] = df['tags'].str.split(',')
oded = df.explode('tags')
```

what does this function do?

Question: what does this function do?

The explode will create a new row for tags with the rest of the row's information duplicated for each new row.

Figure 6: In the STEPWISE and PHASEWISE systems, users can select any code in the editor to ask questions a from the AI. This will create a question box to the right of the main components in which users can ask their clarification question b. The question box will then be replaced with the AI's response c, based on the query and the selected text.

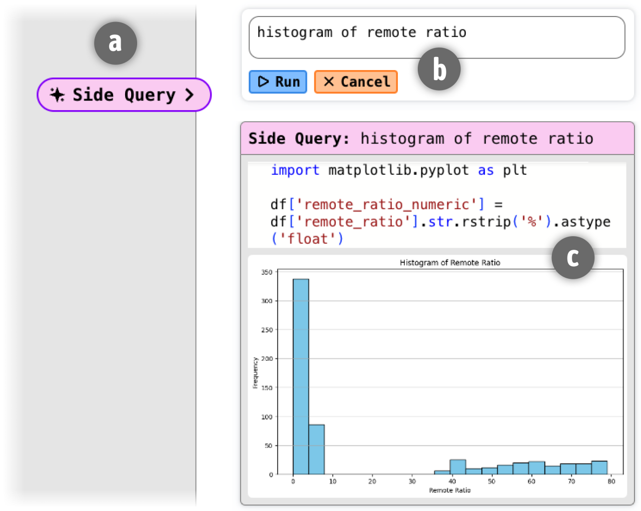
Run Side Query: This feature enables ancillary data analysis tasks using natural language queries. It enables further exploration of the dataset or any intermediary dataframes, and helps users validate and refine assumptions. By clicking on the Side Query button (Figure 7), users can ask natural language queries about the dataset or the current state of the code and variables. The system generates and executes code in the side panel, allowing users to view outputs such as visualizations, identifying outliers, and check the data's consistency.

4.2 CONVERSATIONAL Baseline System

We developed a CONVERSATIONAL system similar to ChatGPT's Advanced Data Analysis plugin as a baseline to compare with the PHASEWISE and STEPWISE systems. The CONVERSATIONAL system does not include any intervention points or editable assumptions, or any of the side conversation features (e.g. *Ask Question* or *Run Side Query*). It decomposes the task into a bullet point of non-editable, natural language assumptions and actions about the task, and then immediately generates and runs non-editable code that solves the entire task. To interact with this system, as with ChatGPT, the user needs to issue follow-up prompts. In this baseline system only the prompts (and follow-up prompts) are editable. For verification, users could read the code and inspect the intermediate variables, and for steering, they could ask follow-up questions in natural language.

4.3 System Implementation

All three variants are built as a web application and Python server stack. The web application is written in TypeScript and the React



histogram of remote ratio

Side Query: histogram of remote ratio

```
import matplotlib.pyplot as plt

df['remote_ratio_numeric'] =
df['remote_ratio'].str.rstrip('%').astype
('float')
```

Figure 7: The run Side Query button (a) is placed to the right of each code execution block in the Stepwise and Phasewise systems, and the input and output assumptions in the Phasewise system. This will open the question box (b) in which the user can specify a natural language data exploration query on any of the intermediary variables or the original dataset. The question box will then be replaced with the AI's response which includes the code and any outputs, including visualizations (c).

web framework to include the interface elements described in Section 4.1. The web application interacted with the Python server for uploading datasets, obtaining their descriptive summaries, running code, and retrieving their execution results. It also called the GPT-4 Turbo models from OpenAI. Each component in the Phasewise or Stepwise system is represented as a node in a tree data structure inside the application. This enables tracing the path from each node to the root node to prepare the context prompt for interacting with the LLM and generating the next component. It also provides state management for the edits that create branches and is used to render the tabbed ribbon interface. The Monaco Editor is used as the code editor in each of the code execution blocks and for syntax highlighting the non-editable code pieces. To enable code execution on the Python server, and to retrieve code execution outputs and intermediary variables, we used the IPython kernel. We used the `%matplotlib inline` command which returned all plots as base64 images that could be included as a response inside the REST APIs.

4.3.1 LLM Prompt Structures. We required complete control over the format of the LLM's output to allow reliable parsing and rendering of structured components. However, few-shot learning (e.g. providing specific input and output examples) would make the LLM overfit to the provided few-shot examples. Through informal experimentation with different prompts and models, we concluded that the GPT-4 and GPT-4 Turbo models are capable of following templates that only specify the format of the output with minimal specification of the content to be generated, with sufficient reliability for a practical evaluation. Figure 8 shows an example of the prompt used to select the columns relevant to the task and generate assumptions and actions about each column. Although the exact format and structure is explicitly provided, the values are not, which enables the system to work generally on a variety of input tasks and datasets.

```
[input-assumptions]:
  [dataset]: <dataset name>
  [relevant-columns]:
    [column]: <column name>
    [assumptions]:
      - <4-7 word column assumption> - <4-7 word required action>
      - <4-7 word column assumption> - <4-7 word required action>
      - ...
    [column]: <column name>
    [assumptions]:
      - <4-7 word column assumption> - <4-7 word required action>
      - <4-7 word column assumption> - <4-7 word required action>
      - ...
```

Figure 8: The prompt template that selects dataset columns relevant to the task and generates assumptions and actions for those columns in the Phasewise AI system.

5 User Evaluation

To evaluate and compare the Stepwise and Phasewise systems in enabling users to steer the AI and verify its responses, we conducted a within-subjects study. The study compared these systems with the Conversational baseline and involved 18 participants who used all three systems to complete six data analysis tasks, with two tasks per system. Datasets and tasks were designed to be sufficiently complex that the AI would not automatically produce correct solutions without user involvement. They required participants to carefully verify the AI's process and responses and steer the AI in addressing any issues.

The main focus of our exploratory study is on understanding the unique ways in which each system aids in steering and verification during the AI-assisted data analysis process. We also investigated the perceived utility of other various system components, and explored the usage patterns and user preferences that emerged with each system.

5.1 Participants

We recruited 18 participants (10 men, 8 women, 0 non-binary) from a large research university. Participants were pre-screened to ensure they were proficient in writing Python code, familiar with Python data science libraries, and experienced in regularly performing data analysis tasks. In terms of data analysis experience, five participants reported having 1–2 years, seven having 3–5, and six more than five years. The majority (14 participants) used Python daily, while the rest used it at least weekly. All reported familiarity with data science libraries like `numpy`, `matplotlib`, with 15 also familiar with `pandas`. Jupyter Notebooks were used daily by eight participants, weekly by six, monthly by two, and rarely by two. For English proficiency in technical contexts, 16 participants felt very comfortable, while two felt somewhat comfortable. In LLM usage for coding, seven reported using daily, seven weekly, and four using monthly or less.

5.2 Data Analysis Tasks

We designed six tasks derived from the ARCADE benchmark [102], which contains a diverse set of tasks from various datasets on Kaggle [39]. These tasks included a series of natural language (NL) queries written by professional data scientists with the intention of interacting with an AI assistant. We selected tasks to not require specific domain knowledge, targeting for participants to solve them within a 15-minute time frame. However, we also wanted to make sure that the tasks included additional complexities that would make it difficult for the AI to correctly solve them without proper user verification and intervention. Therefore, we selected and altered tasks and their corresponding datasets with some format inconsistencies and altered distributions. For instance, Task 2 (Table 2) requires splitting tags and themes with commas before grouping tags by themes. To increase complexity, we modified the tags and themes columns to have only the first theme or tag in capital case, with the rest in lowercase. See Table 2 for details of the six study tasks. We ran each of the six tasks (query + dataset) 10 times on all three systems, ensuring a consistent 100% failure rate.

5.3 Study Procedure

The order of the three Phasewise, Stepwise, and Conversational (baseline) systems was fully counterbalanced using a Latin square design across the 18 participants and tasks to minimize order effects, while tasks were fixed from T1 to T6. Participants spent approximately 50 minutes with each system. They received a 10-minute tutorial and a 5-minute warm-up task to familiarize themselves with the system. Then proceeded to the main study tasks, where they were given a dataset and a NL query. The lead author who conducted all experiments, explained the dataset and relevant columns for each task. Participants proceeded to execute the task and were asked to think aloud throughout the study [24].

Participants were made aware that identifying and correcting mistakes made by the AI was their responsibility. They were instructed to notify the experimenter once they believed they have achieved a correct result using the AI tool. The experimenter would then verify their solution against expected outcomes and provide up to two hints if necessary. These hints addressed AI mistakes correlating with each of the two issues for each task, as listed in Table 2, ensuring consistency across participants. Completion criteria for each task required resolving both issues listed in the table.

Following the completion of each task, participants were asked about their choice of method for steering the AI (e.g., editing the execution plan versus directly editing the code) and their verification processes. After completing two tasks under each condition, participants completed a questionnaire including Likert items about their ability to verify, intervene and steer the AI, sense of control, information overload, frustration levels, and the utility of specific features (exact questions included in Figure 9). Additionally, participants discussed their experience with each system in a 5-minute semi-structured interview.

The sessions were conducted in-person, lasting approximately 2.5 hours with a short break after using each system. Consent was obtained before running the study and each participant was compensated with a GBP £50 Amazon gift card. Our study protocol was reviewed and approved by our institution's ethics and compliance review board.

5.4 Data Collection and Analysis

We recorded the audio and screen activity during each session using MS Teams. Audio recordings were transcribed for analysis. User interactions and feature usage was also logged.

The think-aloud data was our main source of understanding how participants used the different systems and what they thought about them in comparison with each other. We transcribed the think-aloud data and post-condition interviews, and two researchers performed a negotiated, directed qualitative analysis. Ahead of the analysis, we identified a set of research themes concerning steering and verification during the AI-assisted data analysis process, and report our findings organised by these themes in Section 6. Because we were interested in specific themes *a priori*, and were not developing a reusable coding scheme, our analysis differs from the commonly applied inductive approach [7]. We did not develop a codebook, and this is not a situation in which it is appropriate to seek inter-rater reliability. Instead, we used a *"deductive"* coding approach focusing on steering and verification as the main themes. The two researchers iteratively discussed their interpretation of the findings and negotiated each disagreement until it was resolved [64].

Task completion was defined as achieving a solution that correctly resolved both issues indicated in Table 2 for each task within the 15-minute time frame. For tasks that were correctly completed, we recorded the number of hints provided during each task and calculated approximate time on task. Task completion time was an approximate of when participants started the task (clicking on the run query button) until they notified the experimenter that they finished the task, with no remaining issues. However, our analysis of task time is only indicative, as think-aloud protocols interfere with accurate timing.

We analyzed post-condition Likert responses to compare the three systems and determine any statistically significant differences using a Friedman Chi Square test on the responses for each question with the system type as the independent variable. When significant differences were found ($\alpha = 0.05$), a Wilcoxon signed-rank test identified pairwise significant comparisons. Since we made three comparisons (between each pair of systems), we applied a Bonferroni correction ($\alpha = 0.016$).

6 Results

In this section, we present a comparative analysis of the Stepwise and Phasewise AI tools versus the Conversational baseline. Our findings are derived from study observations, log data, participant (P1–P18) think-aloud data, post-condition surveys, and post-study interviews. In turn, we present the results regarding task completion (Section 6.1), steering and control (Section 6.2), and verification (Section 6.3).

Table 2: Final tasks used in the evaluation, including the exact queries for each task, the datasets involved, and issues the AI would encounter without user intervention.

Natural Language Query	Dataset	Issues
Task 1: Show me the top five highly rated products by Nivea	`big-basket-products.csv`	Not recognizing multiple sub-brands of `"Nivea"` (1) and not cleaning the `Rating` column to accurately extract numeric values (2).
Task 2: What is the most common tag associated with each theme?	`anime-list.csv`	Not correctly splitting `Themes` by comma (1) and overlooking the inconsistency in casing between `Tags` and `Themes` (2).
Task 3: Display the top 20 most popular drama names that have only one unique genre? Popularity is based on drama rating and votes.	`korean-drama.csv`	Not filtering `genres` labeled as `"Unknown"` (1) and extreme outliers in `votes` (2)
Task 4: What are the top ten positions (based on mean salary) for working remotely in US-based companies?	`data-science-job-salaries.csv`	Not cleaning `Country Code` (1) and not identifying remote companies using `Remote Ratio` (2).
Task 5: Show the top five movies with the highest percentage return on investment.	`bollywood-movies.csv`	Failed to (1) clean `budget` correctly, and (2) calculate missing `Revenue` values based on `India` and `Worldwide`.
Task 6: What were the top three lowest scoring matches? Sort in ascending order and show location, local and visitor team names.	`euroleague-basketball.csv`	Failed to select related columns for calculating scores (1), and not knowing how to aggregate scores by `Game` and `Round` (2).

6.1 Task Completion

Successful task completion was determined as solving the task with no remaining issues within 15 minutes. Of the 108 task episodes (18 participants × 6 tasks), only 7 were not completed successfully. P13 had three non-completed tasks, and P3, P8, P11, and P14 each recorded one non-completed task. The distribution of non-completed tasks per condition was as follows: Baseline: 1, Phasewise: 2, and Stepwise: 4. The incidence of task non-completion is too low to permit statistical comparison.

In 31 instances of the 108 task episodes, participants indicated task completion despite remaining issues, indicating insufficient verification. In such situations, the protocol was for the researcher to identify the remaining issue(s), requiring participants to steer the tool towards fixing the problem. A Friedman Chi Square test revealed no statistically significant differences in number of verification hints required across conditions ($F(2, 34) = 1.0$, $p = .606$), with 13 hints required for Baseline, 10 for Phasewise, and 8 for Stepwise.

Furthermore, a one-way ANOVA showed no significant differences in task completion time between conditions. The mean completion times across conditions indicated that tasks solved with the Baseline tool were finished slightly faster (M=543s, SD=220s), followed by the Stepwise tool (M=588s, SD=329s), whereas tasks finished with the Phasewise tool were solved slightly slower (M=658s, SD=240s).

Finally, post-condition questionnaires on ease of solving EDA tasks (Figure 9, Q1) or participants' sense of success (Figure 9, Q6) did not show any statistically significant differences across the three AI tools.

6.2 Steering and Control

Analysis of the post-condition questionnaires about control found that participants felt significantly more in control of the AI's analysis process when using the Stepwise and Phasewise systems compared to the Baseline (Stepwise-vs-Baseline: $p = .001$, $d = .42$; Phasewise-vs-Baseline: $p = .004$, $d = .42$). Participants also reported that the Phasewise and Stepwise systems were significantly easier to intervene and fix (Figure 9, Q3) whenever it was doing something wrong (Phasewise-vs-Baseline: $p = .012$, $d = .55$; Stepwise-vs-Baseline: $p = .011$, $d = 1.05$). However, no significant differences were found in the perceived ease of steering between the three systems (Figure 9, Q4).

In the remainder of this section, we explore themes identified within participants' workflows and their think-aloud data. This analysis reveals varied preferences among participants and offers

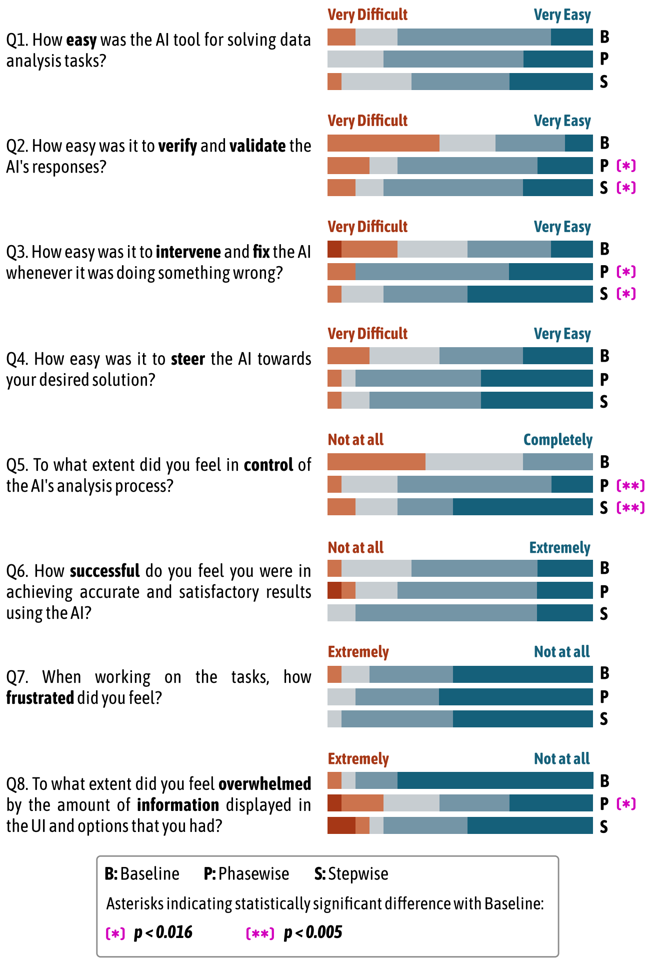


Figure 9: Summary of responses to the post-condition Likert questions for each system.

insights into factors that either facilitated or hindered their ability to steer the process.

6.2.1 Steering by Directly Editing AI's Assumptions and Actions. Participants appreciated the ability to directly edit the AI-generated assumptions, thereby aligning the system's operations with their expectations. As P16 stated: *"I could add any assumptions that I had in mind and make it the AI's assumptions."* This enabled users to *"steer the AI's decision making process to different directions."* (P3). For P11, the ability to edit assumptions fostered a more critical perspective towards them; conversely, in the baseline system where the assumptions were fixed, they tended to accept them without questioning, as P11 would *"just go with it as opposed to being critical."* Furthermore, the baseline's fixed assumptions were a source of frustration, a sense of lack of control, and a barrier to effective interaction (P2, P9, P15, P16, P17). P16 expressed a preference for editing the assumptions directly *"instead of just trying to ask a [follow-up] question,"* and P9 noted that the ability to edit assumptions eliminated the need for manually *"engineering your prompts"*.

The structured editing of assumptions, actions, and execution plan enabled direct manipulation, explicit and fine-grained control over the AI's behavior. P5 expressed that it was easier to interact with, since the assumptions were given and they just had to modify which made the interaction *"less talking and more clicking on buttons"*. Participants appreciated being able to *"edit something very specific"* (P8), such as a step in the execution plan, or *"including the pre-processing steps right beside the columns"* (P7). Similarly, P12 found it difficult to make targeted edits in the absence of structure, stating that *"making small edits [in the Baseline] requires a lot of tweaking."* Additionally, P6 reported that increased structure facilitated locating information and served as a memory aid. This contrasts with the difficulties they experienced using the STEPWISE system which lacked the amount of structure used in the PHASEWISE system. With the STEPWISE system, participants had to *"find the correct step to make an edit"* or *"find which column the assumption refers to"* (P6). P4 further indicated that the overall structure provided in the PHASEWISE system *"pushes me towards structured analysis"*, specifically *"in terms of validating assumptions"*.

While most participants were generally positive about direct and fine-grained editing of assumptions, P2 and P4 preferred the AI to update its assumptions via natural language queries. P4 additionally criticized the PHASEWISE system for the inability to see results update instantly after editing input/output assumptions.

6.2.2 Higher Perceived Control Through Step-by-Step Task Decomposition. A distinct advantage observed with the STEPWISE system was the enhanced control participants reported over the data analysis process. This perception was mainly attributed to tackling the task in smaller, manageable segments. For instance, P16 reported an increased sense of control, by being able to *"easily edit the assumptions and actions in each step."* Similarly, P11 felt *"much happier"* and *"more confident"*, attributing it to the ability to *"manipulate steps naturally."* P10 also shared a sense of more control over the AI, stating that they *"were not scared"* to make edits to what the AI was doing, as *"it was just a couple of lines,"* and *"it was more inviting to edit the assumptions."* P17 mentioned that the STEPWISE system facilitated an iterative process *"where [they] could easily go back and change something"*. While these reports indicate a perception of enhanced control, actual control should be measured in future work.

6.2.3 Steering by Manually Editing Code. Most participants appreciated the ability to manually edit AI-generated code. P18 mentioned editing the AI-generated code was like *"you're taking over from the AI."* These edits ranged from minor modifications, such as manually changing a threshold or printing values, to more involved changes such as using the *Generate code* feature to update the logic behind a line of code. When using the *Generate code* feature, participants (P1, P3, P4, P7, P8, P9 P11, P12, P17, P18) selected a line of code and prompted the AI to update it based on a provided natural language query. For instance, P9 selected `df[df['company_location'] == 'US']` in their code and prompted the AI with "`can you change this line to look for containing 'US' instead of strict equality?`" P12 experienced increased control when using the *Generate code* feature since they *"could make very granular prompts"*.

Participants preferred manually editing code for minor changes over using the AI-steering methods provided in each system. Many participants expressed frustration with the inability to directly edit code in the baseline system. P7 stated, *"if [the code] was editable, I can just correct things which I know myself instead of prompting again."* Notably, P10, unable to edit the AI-generated code directly with the baseline, resorted to copying the desired code line, editing it in the follow-up prompt, and then asking the AI to incorporate the edited code. They found crafting a prompt for the specific edit challenging, admitting, *"I don't really know how to prompt it to get it do what I want."* However, in some cases, P3 and P10 indicated reasons for *not* wanting to edit the AI-generated code and instead preferred using the AI-steering methods to make the edits. P3 wanted *"to make sure that [their edit] is consistent with the rest of the code"* and P10 stated, *"I don't like editing the code directly when I haven't written it."*

6.2.4 Preference for Conversational Steering. In some instances, participants (P4, P5, P6, P8, and P11) specifically indicated a preference for steering the AI through natural language prompts. They favored the CONVERSATIONAL method of steering the AI over editing the structured assumptions, actions, or execution plans. For example, P4 expressed difficulty in understanding how changes to the assumptions in the PHASEWISE system, affected the LLM's output. In contrast, P4 had a more accurate mental model of how to interact with the CONVERSATIONAL baseline, stating *"I know exactly how writing a prompt is going to affect it."* Others felt constrained by the need to adhere to a specific structure, expressing a preference for more free-form interactions. For instance, P8 described the CONVERSATIONAL system as easier and faster for *"directing the AI using natural language"*, compared to the STEPWISE system where they felt they were *"trying to change the syntax of the AI."* Similarly, P5 wanted to *"intentionally write vague prompts and see how much [the AI] understands."* P4 believed that the CONVERSATIONAL system required less cognitive effort, stating *"I don't like spending that effort to think about it."* P11 mentioned feeling *"less critical"* about themselves when using the CONVERSATIONAL tool, allowing the AI to *"go and figure it out"* on their behalf. P11 elaborated: *"I knew what it was [that] I wanted it to consider, but when [the tool] is expecting a structured input then I was more concerned with providing it in a nice and structured manner."*

6.2.5 Avoiding Edits that Lead to Inconsistency or Regeneration. To discover participant reasoning behind the selection of a particular steering method from the available options, participants were asked about their specific interactions after each task. We found that participants avoided certain edits when using the STEPWISE and PHASEWISE systems in two cases:

- if they had previously edited downstream components and then decided to update an upstream component, the system ignored all downstream edits since regenerations proceed from top to bottom.
- if they decided to make edits to downstream components that conflicted with assumptions or code in upstream components.

For example, P1 was worried whether the AI would *"regenerate everything else correctly"* after updating a specific assumption. P12 mentioned that there was *"no obvious way of going back without redoing all of the earlier changes"*. Similarly, P4 mentioned that they *"want to make sure that [their] changes propagate and stay."* P8 unexpectedly found that they lost downstream edits after making an edit on the upstream components, expressing: *"Oh! So when it regenerated this, it forgot about [their previous edit]."* Participants expressed the need for bidirectional updates to maintain consistency across different components after making an edit. For example, when P18 was using the STEPWISE system, they could not make edits at the beginning of the problem, which felt natural to them, because *"everything underneath it will drop."* They preferred that the system would just highlight parts that would be invalidated in the downstream instead of regenerating everything. P10 expressed concern about requiring to *"read everything every time [they] made a small change."* In contrast, P18 accepted previous components going out of sync, as at that point they have *"taken over from the AI"* and P5 appreciated the propagation of changes between components, stating that they liked *"how interconnected things were"*.

6.3 Verification

In all systems, participants relied on reading and analyzing the AI-generated code and inspecting the intermediary variables for verification. The STEPWISE system's approach of breaking down tasks into smaller steps, along with the side conversation feature available in both STEPWISE and PHASEWISE systems, improved participants' confidence in verification. The post-condition questionnaire items indicated that both STEPWISE and PHASEWISE systems significantly facilitated easier verification (Figure 9, Q2) of the generated solution compared to the baseline (PHASEWISE-vs-Baseline: $p = .016$, $d = .47$; STEPWISE-vs-Baseline: $p = .016$, $d = 1.44$).

6.3.1 Verification through Reading Code and Asking Questions. Reading the code line-by-line was a common verification method. P12 mentioned that *"you still have to read all the code and understand what it's doing"* for verification. Participants mostly relied on their own knowledge about Python and Pandas for verification, as stated by P8: *"you have to know how to code, because you have to read the code and make sure it makes sense."* When P3 was asked how they knew that they had successfully finished the task, they responded *"I inspected the code and found that it handled that edge case correctly."* However, in many instances participants had difficulty understanding the AI-generated code, if it used idioms or functions unfamiliar to the participant. Participants appreciated the *Ask Question* feature in these situations. A majority of participants (n=13) used this feature at least once to explain a portion of the generated code. For example, when P9 was working on Task 2, they stated: *"I've never seen this function* `explode()` *so I'm just gonna ask what does this do"'*. Participants found the responses to their queries useful. For example, P2 asked about the `fillna()` function and realized that the code was doing something undesirable (replacing `nan` with `"Unknown"`). Furthermore, participants felt the absence of the Ask Question feature when using the baseline tool, where for instance P18 wanted to use a search engine, and P6 mixed the main thread of the task with a comprehension question: *"I don't understand [refers to code]."* During the study, several unanticipated, yet effective use cases of the Ask Question feature emerged, such as P4 asking questions about an assumption, and P12 requesting help with debugging by selecting part of the code and asking *"why this code produces an error."*

6.3.2 Inspecting Intermediary Variables. All participants used the intermediary variable inspection feature available in all three systems for verification. Participants inspected variables to *"compare between turns"* (P1), and to *"see if [the system] has done the [operation] correctly"* (P14). P12 mentioned that the verification process was similar to *"debugging"* and P11 indicated that inspecting all the dataframes significantly increases confidence in the process.

6.3.3 Steering for Verification. However, generated code was not always easily verifiable. In some instances, the generated code overwrote variables instead of creating new dataframes, which interfered with variable inspection as only the final state of the variable after code execution was displayed. In other cases, the generated code directly computed the final result, without sufficient decomposition of steps necessary for proper verification. For example, P4 mentioned that *"the way that [the system] is generating code does not create useful intermediary dataframes ... it's showing me the end result"*. In some cases this lead to unjustified reliance on the generated code, as P3 mentioned: *"I guess I would need to trust in this case."*

Therefore, a recurring theme that emerged was participants trying to update the code, through steering, to include more informative and useful intermediary variables. For example, P16 added a new step to the execution plan to emit new outputs and other relevant columns in addition to just showing the final result. P4 added an explicit step to the execution plan `display couple of groups so I can manually verify`. Interestingly, there were also several cases that participants just did not understand the method used in the generated code, and therefore, asked the system to *"come up with a more understandable solution"* (P6).

6.3.4 Focusing on Smaller Steps Facilitated Verification. The Stepwise system provided a one-to-one mapping of code with the intermediary variables for each step. Participants found that they can easily *"focus on each small step"* (P15), improved their confidence since they were forced to *"think of edge cases along the way"* (P9). P5 mentioned that *"having it step-by-step leads to more reflecting from my side and verifying each block"*. Granular decomposition also helped with locating issues. It was *"easier to figure out what is going wrong"* (P11), and *"there was less margin of error"* (P7). For P3, the higher number of intervention points in the Stepwise system helped their *"results to be correct all the time."*

The step-by-step process was *"more natural"* (P11) and *"felt a lot more similar to how [they] would approach the analysis"* (P9), because *"don't usually solve the whole task at once."* (P10). Interestingly, P7 *"felt less need for validation since [they are] inspecting after each step"*.

Additionally, participants experienced less information overload: *"you get the blocks one-by-one so you are not overwhelmed by too much information"* (P5), and compared to the Phasewise system which felt more like *"debugging somebody else's code than writing my own code"* (P10). However, several participants (P4, P5, P13, and P18) were critical of the Stepwise system for not providing any information about the upcoming next step. P18 stated that *"I do not want it to do everything at once, but I want to know what it's gonna do next"* and P5 expressed reduced confidence for *"not knowing the steps in advance."*

6.3.5 Aggregated Information Helped with Verification. Many participants (n=7) appreciated how the Phasewise system aggregated descriptive statistics and assumptions for each column, finding that these elements scaffolded the AI's reasoning about the task. These helped P15 *"understand what are the different possibilities,"* enabled P6 to *"see exactly how each column will be treated"*, and *"forced"* P3 *"to see the data a bit better."* For example, in Task 3, P3 easily found the `"Unknown"` genre problem in the descriptive statistics, immediately updating the corresponding assumptions to handle it. Furthermore, P9 justified the usefulness of the aggregated information per columns by stating *"it is something a lot of times you would end up asking about anyways,"* and P1 appreciated that it *"gives you a preview of everything together."*

However, some participants mentioned that the aggregated statistics were a source of information overload. The post-condition questionnaires also indicated that they felt significantly overwhelmed (Figure 9, Q8) by the amount of information displayed when using the Phasewise system compared to the baseline ($p = .008$, $d = .11$). P16 felt *"frustrated by the amount of things that [they] saw on the screen"* and P8 stated that *"It could start getting quite cumbersome if the dataset was large"*. Similarly, P4 found the structure to be overwhelming and some assumptions about columns were irrelevant to what they were trying to do, and instead, they wanted the system to contextually display the right amount of information.

6.3.6 Running Side Queries. The *Run Side Query* feature of the Stepwise and Phasewise systems was the most frequently used side conversation feature with 82 usages, compared to 26 usages of *Ask Question*, and 17 usages of *Generate Code*. All participants ran side queries at least once. About 75% (n=62) of the side queries were to understand data and its limitations and 17% (n=14) were to visualize data for inspection.

The *Side Query* feature facilitated a novel and effective workflow, especially when integrated with the editable assumptions, actions, and execution plan in both systems. Participants used it to *"explore the dataset, validate assumptions, and add them to the column breakdown"* (P17), and *"to build up assumptions and edit the plan"* (P10). P9 used the Side Query to plot the distribution of a column and select a better threshold for filtering outliers. P13 mentioned gaining *"more confidence after plotting histograms"* and found that the Side Query proved more beneficial in the Stepwise system because it *"forces me to check the outputs at every step"* in contrast to the Phasewise system. In the baseline system, without the *Side Query* feature, P4 and P10 resorted to the main thread for their verification queries. This experience led them to appreciate the convenience of having the *Side Query* feature in a side panel, which prevented interference with the main thread. Reflecting on this, P10 highlighted that *"it was not taking away from the history of questions I'd established already"* and expressed a desire to avoid getting *"off track with intervening stuff"* in the baseline system.

6.3.7 Deferring Steering after Seeing Initial Results. Participants frequently preferred to first see results before interacting with and steering the AI tool. P6 highlighted that they *"just want to see the* *result first and then trim it"*. P18 wanted to *"look at the result first and if the result was nonsense then go back"*. This was particularly the case in the Phasewise and Stepwise systems, as P4 mentioned that the system made them *"go through all of these steps and spend so much time before [they] could actually see the result."* P8 wished to see the generated code because they did not trust the system to generate correct code even if the execution plan was correct: *"part of me wants to just see what it does, even if the [execution] plan looks reasonable, I know there's gonna probably be errors."* Another reason why participants wanted to defer steering, particularly in the Phasewise system, was that the input/output assumptions or the execution plan did not have enough details about how the AI is going to *"handle"* or *"calculate"* something (P10, P14). Similarly, P12 was not sure about something in the plan, so they said *"I'm going to generate first and see what happens."* Lastly, P4 indicated that having a small working memory was why they preferred the baseline system, where every interaction with the AI resulted in a complete output to verify.

6.4 Summary of Results

Our study revealed that while there was no difference in task success, completion time, or number of required verification hints, participants felt significantly more in control of the data analysis process when using the Phasewise and Stepwise systems compared to the Conversational baseline. Overall, the results show that both systems were preferred over the Conversational baseline. However, because the Phasewise system led to significantly higher information overload, the Stepwise system emerged as the most balanced and effective of the three.

The study also highlighted the value of side conversations in the AI-assisted data analysis process. The ability to run side queries facilitated an iterative workflow of exploration, validation, and updating of editable assumptions, particularly in the Stepwise system. The Ask Question feature helped participants understand the AI-generated code, while the Generate Code feature allowed them to update the logic behind a line of code. In the absence of side conversations, as in the baseline system, participants mixed their queries with the main thread of the task.

In the Phasewise system, the organization of assumptions enabled direct and broad control and served as a memory aid. Participants appreciated the aggregated descriptive statistics and assumptions for each column, which helped them understand how each column would be treated. However, the amount of information displayed was also a source of overload for some participants.

The Stepwise system provided fine-grained control by breaking down tasks into smaller, manageable segments. This improved verification, as participants could focus on each small step and consider edge cases along the way. However, a limitation of the tool was the inability to see the next step in the process.

Finally, some participants preferred the Conversational system for its simpler and more familiar mental model of how asking follow-up questions would affect the AI's output. They also appreciated the flexibility of free-form interactions and the ability to see a result faster. However, the inability to directly edit the AI-generated code was a source of frustration.

7 Discussion and Implications for AI-Assisted Data Analysis Tools

Our designs for the Phasewise and Stepwise systems, and their evaluation against the Conversational tool, have provided us with a deeper understanding of the trade-offs within the design space of AI-assisted data analysis tools. This discussion will explore these trade-offs, their impact on user preferences and interactions, and suggest guidelines for design.

Our study finds that the key to designing AI-assisted data analysis tools lies in providing the user with the necessary controls to make informed decisions and maintain control over the process. This echoes the longstanding positioning of the role of user interface elements in interactive machine learning systems as providing *decision support* [53, 78, 82]. Our study finds that in the specific case of AI-assisted data analysis, decision support is subject to the following key design questions:

- **DQ1 Steering Points:** At what points should the system allow the user to intervene in the process and steer? How frequently should these steering opportunities occur?
- **DQ2 Steering Support:** How does the user verify the current state of the AI's output and determine which direction they should steer it? How should the tool facilitate users in making informed decisions at each steering point?
- **DQ3 Steering Modality:** What interface affordances are available for the user to steer the process? How structured or flexible should the modality of their interaction be?

These are similar to the design questions regarding the number and nature of "choice points" within a data analysis workflow generated by an earlier generation of tools termed Intelligent Discovery Assistants (IDAs) [83]. We find that the choices users face with IDAs (e.g., what type of regression or normalisation to apply at a particular step) still exist within generative AI-assisted data analysis, but they are embedded within the higher-level challenges of steering, and are experienced by users as a secondary concern.

7.1 DQ1: Steering Points

One design question is at which points during the generation process the user should reflect, check for correctness, and steer if needed. Our Stepwise and Conversational tools can be seen as two ends of a spectrum of intervention opportunities. The Stepwise system offers steering points after each step of the analysis, allowing for incremental adjustments. In contrast, the Conversational tool aims to complete the task with minimal user interruption, offering a chance to adjust only after attempting to solve the task.

Our results comparing user experiences with these systems reveals significant trade-offs. More frequent steering points increase users' confidence in the results and their sense of control over the AI's process. However, it demands more cognitive effort and delays the final outcome due to the frequent pauses created by the intervention points. Additionally, it may lead to premature decisions as users commit to directions without understanding their future impact. According to the Cognitive Dimensions of Notations framework [26], this is a clear case of the system imposing "premature commitment". Indeed, our findings indicate varied user preferences between the Conversational and Stepwise systems due to these factors.

To address this challenge and balance control with cognitive load, AI-assisted data analysis tools could adopt a strategy that initially attempts to solve the task without user intervention, followed by a more interactive steering and verification phase with a rich set of steering points. This is similar to the highly validated design strategy of information visualization tools to provide an "overview first", and only later "details on demand" [85].

7.2 DQ2: Steering Support

At each steering point, users must verify whether the output is correct, and if not, choose a steering action. How should the tool facilitate the information seeking and exploration process that is required for the user to make informed decisions? Tools for verifying AI-generated content are termed "co-audit" [25]. Auditing and verification are part of the analyst process of sensemaking [73] and information foraging [72]. Critically, such processes are an opportunistic mix of top-down hypothesis formation and bottom-up hypothesis testing [21], with multiple activities proceeding in a parallel, non-linear fashion. This is antagonised by the sequential nature of chat interfaces.

Our exploration into the design space introduced side conversations and the *Run Side Query* function to aid this process. Furthermore, in the Phasewise system, the system displayed dataset columns relevant to the user's task, along with interactive descriptive statistics. Moreover, the execution plan component suggested optional steps for the user to consider adding before proceeding to the next step, enhancing the decision-making process.

Our findings indicate that when the system provides timely, accurate, and relevant information, it fosters a genuinely collaborative experience. Conversely, displaying irrelevant information can reduce trust in the AI and potentially leading to information overload. Users also risk becoming overly dependent on AI for guidance, potentially neglecting critical information seeking which may lead to poor decision making.

Participants noted their desire for the AI to act as an agent, aiding in the assumption-building process at steering points. Future tools could automatically retrieve assumptions by pinpointing specific, relevant evidence to support informed decision-making without overwhelming users. Additionally, these tools could help in accessing domain-specific knowledge pertinent to the data analysis task. Lastly, tools should be transparent regarding the AI's limitations in sourcing all necessary information for optimal decision-making at each decision point.

7.3 DQ3: Steering Modality

After the user has decided which direction to steer the AI, their next action is to specify their intent to the AI. The design question here is determining the right interface and modality for the user to steer the AI. In response, we introduced two distinct interfaces: a free-form text editor used in the Conversational system for maximal flexibility, and structured editors in the Stepwise and Phasewise systems. The structured editors contain the assumptions and actions generated by the LLM's chain-of-thought, allowing users to edit them for steering.

Our results highlight the trade-offs between these approaches. An unstructured and flexible modality reduces perceived cognitive load, and allows users to be less self-critical when making edits. Users can form a more straightforward mental model of how their inputs steer the AI and receive immediate feedback. However, increased flexibility shifts the responsibility of precise and effective interaction onto the user. Users lose fine-grained control and may have to engage in the challenging and time-consuming task of prompt design [60, 103].

Depending on the generative AI and data analysis expertise of the user, a range of steering methods may be appropriate. A potential approach could be adding the ability to switch between structured editing of assumptions or instructing the AI with free-form queries. Moreover, to increase transparency in the user's mental model about how their edits affect the system, tools could enable inspection of the underlying LLM prompts, and highlight how their steering edits affect the information sent to the model. Lastly, advanced users might appreciate the ability to manually adjust the underlying prompts. These design suggestions are complementary to established prompting guidelines and practices (e.g., [66]).

8 Limitations

We identified several limitations in our study and system design that should be considered when interpreting the results.

8.1 Study Limitations

In our evaluation study, the tasks were manually made more complex and less clean to always include errors when presented to the AI tools to require further verification and steering. However, this may have impacted the ecological validity of the the tasks. Another challenge to ecological validity is our decision to provide the initial NL query for each task, which does not reflect how these systems would be used in practice, and reduced the opportunity for us to study the consequences of divergent natural prompting strategies. For our study, this was an acceptable trade-off as it guaranteed that all participants would encounter the same steering and verification needs, which allowed clearer comparisons between the different systems. It also allowed us to sidestep the issue of participant queries being unevenly "primed" by the task description [60]. Future work may relax this constraint to study a wider range of participant prompting strategies.

Typicality and novelty preferences may have influenced how participants ranked their preference of features or system [34]. Participants might also be biased towards systems they believe are of personal interest to the researcher (known as the "yours is better" bias) [16]. As a mitigating measure, the researcher did not associate themselves with the prototypes in this study and elicited reflections grounded in participants' concrete experiences rather than subjective perceptions [6]. These reflections were further corroborated through screen recordings and usage logs [13, 62].

Participants only used each system for a short time (30 minutes per system, not including the tutorial), which is typical of controlled experiments in laboratory settings. These cannot capture long-term effects [80]; some phenomena only emerge over long-term use and some phenomena which appear to be salient with short-term use erode over time. Consequently, future work could aim to cross-validate our findings longitudinally using experience sampling [12] or diary studies [75].

8.2 System Limitations

During the study, we observed limitations in how the system propagated users edits, both upstream and downstream. The first limitation involved propagating changes from edited assumptions to generated code, where in some cases the language model would appear to ignore the edits. This is a fundamental failure mode of generative AI systems, however, system implementations and interfaces can exacerbate the issue. Complex prompting approaches with many instructions can make it unlikely that the model identifies small changes to assumptions. Additionally, distinguishing assumptions in the interface can set incorrect user expectations around how a model attends to assumptions.

Another limitation of the system is that edits to downstream code or assumptions are not propagated to upstream assumptions, and if a user makes an earlier edit it will overwrite any subsequent changes. The intention behind this *prima facie* design decision was to present a simple model of "cause and effect" that represented how completions were generated, namely, the *context* of any point in the system is only that which appears before. Some participants identified this limitation and it influenced their steering preferences, choosing not to edit an assumption because it would clear changes that had been made to code.

9 Conclusion

In this work, we explore the design space of AI-assisted data analysis tools by presenting two novel interfaces that aim to improve steering and verification. Starting from the observation that task decomposition is an emerging characteristic of recent LLM-based systems, we developed two systems that explore different modes of interactive task decomposition, each based on unique trade-offs. The first, Stepwise, decomposes the problem step by step; the second, Phasewise, decomposes the problem into logical phases. Our evaluation demonstrates that users experienced a greater sense of control and confidence with our systems in comparison to a chat-based baseline. Still, task decomposition is not without preference or cost. Some users prefer to work through the task incrementally, whilst others prefer to see the plan upfront. Additionally, highly-structured decomposition can introduce cognitive burden. Consequently, we imagine that future AI interfaces will need to support adaptive decomposition that reacts to the user and task.

References

[1] [n. d.]. ChatGPT Plugin for Notebook. https://web.archive.org/web/20231216052624/https://noteable.io/chatgpt-plugin-for-notebook/. Accessed: 2023-12-16.

[2] Anaconda. 2024. Anaconda Assistant Launches to Bring Instant Data Analysis, Code Generation, and Insights to Users. https://www.anaconda.com/blog/anaconda-assistant-launches-to-bring-instant-data-analysis-code-generation-and-insights-to-users. Accessed: 2024-03-01.

[3] Zahra Ashktorab, Mohit Jain, Q Vera Liao, and Justin D Weisz. 2019. Resilient chatbots: Repair strategy preferences for conversational breakdowns. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–12.

[4] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[5] Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. 2019. AutoPandas: neural-backed generators for program synthesis. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–27.

[6] Michael H Bernhart, IGP Wiadnyana, Haryoko Wihardjo, and Imbalos Pohan. 1999. Patient satisfaction in developing countries. *Social science & medicine* 48, 8 (1999), 989–996.

[7] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[8] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (<conf-loc>, <city>Honolulu</city>, <state>HI</state>, <country>USA</country>, </conf-loc>) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376729

[9] Bhavya Chopra, Anna Fariha, Sumit Gulwani, Austin Z Henley, Daniel Perelman, Mohammad Raza, Sherry Shi, Danny Simmons, and Ashish Tiwari. 2023. CoWrangler: Recommender System for Data-Wrangling Scripts. In *Companion of the 2023 International Conference on Management of Data*. 147–150.

[10] 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 preprint arXiv:2310.16164* (2023).

[11] Anamaria Crisan, Brittany Fiore-Gartland, and Melanie Tory. 2020. Passing the data baton: A retrospective analysis on data science work and workers. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 1860–1870.

[12] Mihaly Csikszentmihalyi and Reed Larson. 1987. Validity and reliability of the experience-sampling method. *The Journal of nervous and mental disease* 175, 9 (1987), 526–536.

[13] Mary Czerwinski, Eric Horvitz, and Edward Cutrell. 2001. Subjective duration assessment: An implicit probe for software usability. In *Proceedings of IHM-HCI 2001 conference*, Vol. 2. 167–170.

[14] Databricks Assistant. 2024. Introducing Databricks Assistant, a context-aware AI assistant. https://www.databricks.com/blog/introducing-databricks-assistant. Accessed: 2024-03-01.

[15] DataChat. 2024. The no-code, generative AI platform for instant analytics. https://datachat.ai/. Accessed: 2024-03-01.

[16] 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*. 1321–1330.

[17] Michael A DeVito, Jeremy Birnholtz, Jeffery T Hancock, Megan French, and Sunny Liu. 2018. How people form folk theories of social media feeds and what it means for how we study self-presentation. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–12.

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

[19] Victor Dibia, Adam Fourney, Gagan Bansal, Forough Poursabzi-Sangdeh, Han Liu, and Saleema Amershi. 2022. Aligning Offline Metrics and Human Judgments of Value for Code Generation Models. *arXiv preprint arXiv:2210.16494* (2022).

[20] David Donoho. 2017. 50 years of data science. *Journal of Computational and Graphical Statistics* 26, 4 (2017), 745–766.

[21] Ian Drosos, Advait Sarkar, Xiaotong Xu, Carina Negreanu, Sean Rintel, and Lev Tankelevitch. 2024. "It's like a rubber duck that talks back": Understanding Generative AI-Assisted Data Analysis Workflows through a Participatory Prompting Study. In *Proceedings of the 3rd Annual Meeting of the Symposium on Human-Computer Interaction for Work (in press) (CHIWORK '24)*. Association for Computing Machinery, New York, NY, USA.

[22] Will Epperson, April Yi Wang, Robert DeLine, and Steven M Drucker. 2022. Strategies for reuse and sharing among data scientists in software teams. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 243–252.

[23] Alexander J Fiannaca, Chinmay Kulkarni, Carrie J Cai, and Michael Terry. 2023. Programming without a Programming Language: Challenges and Opportunities for Designing Developer Tools for Prompt Programming. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–7.

[24] Marsha E Fonteyn, Benjamin Kuipers, and Susan J Grobe. 1993. A description of think aloud method and protocol analysis. *Qualitative health research* 3, 4 (1993), 430–441.

[25] Andrew D Gordon, Carina Negreanu, José Cambronero, Rasika Chakravarthy, Ian Drosos, Hao Fang, Bhaskar Mitra, Hannah Richardson, Advait Sarkar, Stephanie Simmons, et al. 2023. Co-audit: tools to help humans double-check AI-generated content. *arXiv preprint arXiv:2310.01297* (2023).

[26] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

[27] Garrett Grolemund and Hadley Wickham. 2014. A cognitive interpretation of data analysis. *International Statistical Review* 82, 2 (2014), 184–204.

[28] Ken Gu, Madeleine Grunde-McLaughlin, Andrew M McNutt, Jeffrey Heer, and Tim Althoff. 2023. How do data analysts respond to ai assistance? a wizard-of-oz study. *arXiv preprint arXiv:2309.10108* (2023).

[29] Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M Drucker. 2024. How Do Analysts Understand and Verify AI-Assisted Data Analyses? (2024).

[30] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.

[31] Sumit Gulwani and Mark Marron. 2014. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 803–814.

[32] Philip J Guo, Sean Kandel, Joseph M Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 65–74.

[33] Andrew Head, Fred Hohman, Titus Barik, Steven M Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[34] Paul Hekkert, Dirk Snelders, and Piet CW Van Wieringen. 2003. 'Most advanced, yet acceptable': Typicality and novelty as joint predictors of aesthetic preference in industrial design. *British journal of Psychology* 94, 1 (2003), 111–124.

[35] Gan Keng Hoon, Loo Ji Yong, and Goh Kau Yang. 2020. Interfacing chatbot with data retrieval and analytics queries for decision making. In *RITA 2018: Proceedings of the 6th International Conference on Robot Intelligence Technology and Applications*. Springer, 385–394.

[36] Edwin L Hutchins, James D Hollan, and Donald A Norman. 1985. Direct manipulation interfaces. *Human–computer interaction* 1, 4 (1985), 311–338.

[37] Ji-Youn Jung, Sihang Qiu, Alessandro Bozzon, and Ujwal Gadiraju. 2022. Great chain of agents: The role of metaphorical representation of agents in conversational crowdsourcing. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–22.

[38] Jupyter AI. 2024. Jupyter AI, brings generative AI to Jupyter notebooks. https://jupyter-ai.readthedocs.io/en/latest/. Accessed: 2024-03-01.

[39] Kaggle. 2023. Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/. Accessed: 2024-03-01.

[40] Eirini Kalliamvakou. 2022. Research: quantifying GitHub Copilot's impact on developer productivity and happiness. *The GitHub Blog* (2022).

[41] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*. 3363–3372.

[42] Sean Kandel, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. *IEEE transactions on visualization and computer graphics* 18, 12 (2012), 2917–2926.

[43] Jan-Frederik Kassel and Michael Rohs. 2018. Valletto: A multimodal interface for ubiquitous visual analytics. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–6.

[44] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Z Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. (2024).

[45] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists.. In *CHI*, Vol. 10. 3025453–3025626.

[46] Mary Beth Kery, Bonnie E John, Patrick O'Flaherty, Amber Horvath, and Brad A Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.

[47] Mary Beth Kery and Brad A Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 25–29.

[48] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E John, and Brad A Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–11.

[49] Pranav Khadpe, Ranjay Krishna, Li Fei-Fei, Jeffrey T Hancock, and Michael S Bernstein. 2020. Conceptual metaphors impact perceptions of human-AI collaboration. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW2 (2020), 1–26.

[50] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*. 96–107.

[51] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2017. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering* 44, 11 (2017), 1024–1038.

[52] Sunnie SY Kim, Elizabeth Anne Watkins, Olga Russakovsky, Ruth Fong, and Andrés Monroy-Hernández. 2023. "Help Me Help the AI": Understanding How Explainability Can Support Human-AI Interaction. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.

[53] Rafal Kocielnik, Saleema Amershi, and Paul N Bennett. 2019. Will you accept an imperfect ai? exploring designs for adjusting end-user expectations of ai systems. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–14.

[54] Laura Koesten, Kathleen Gregory, Paul Groth, and Elena Simperl. 2021. Talking datasets–understanding data sensemaking behaviours. *International journal of human-computer studies* 146 (2021), 102562.

[55] Himabindu Lakkaraju, Dylan Slack, Yuxin Chen, Chenhao Tan, and Sameer Singh. 2022. Rethinking explainability as a dialogue: A practitioner's perspective. *arXiv preprint arXiv:2202.01875* (2022).

[56] Jenny T Liang, Chenyang Yang, and Brad A Myers. 2023. Understanding the usability of AI programming assistants. *arXiv preprint arXiv:2303.17125* (2023).

[57] Q Vera Liao, Daniel Gruen, and Sarah Miller. 2020. Questioning the AI: informing design practices for explainable AI user experiences. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–15.

[58] Q Vera Liao and Kush R Varshney. 2021. Human-centered explainable ai (xai): From algorithms to user experiences. *arXiv preprint arXiv:2110.10790* (2021).

[59] Mark Liffiton, Brad Sheese, Jaromir Savelka, and Paul Denny. 2023. CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes. arXiv:2308.06921 [cs.CY]

[60] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D 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 2023 CHI Conference on Human Factors in Computing Systems*. 1–31.

[61] Yang Liu, Tim Althoff, and Jeffrey Heer. 2020. Paths explored, paths omitted, paths obscured: Decision points & selective reporting in end-to-end data analysis. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–14.

[62] Wendy E Mackay. 1998. Triangulation within and across HCI disciplines. *Human-Computer Interaction* 13, 3 (1998), 310–315.

[63] Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. 2024. Directgpt: A direct manipulation interface to interact with large language models. (2024).

[64] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. *Proceedings of the ACM on human-computer interaction* 3, CSCW (2019), 1–23.

[65] 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 2023 CHI Conference on Human Factors in Computing Systems*. 1–16.

[66] Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2021. Reframing Instructional Prompts to GPTk's Language. *arXiv preprint arXiv:2109.07830* (2021).

[67] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading between the lines: Modeling user behavior and costs in AI-assisted programming. (2024).

[68] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q. Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3290605.3300356

[69] Jakob Nielsen. 2006. Progressive disclosure. *nngroup. com* (2006).

[70] OpenAI. 2023. ChatGPT plugins. https://openai.com/blog/chatgpt-plugins#code-interpreter. Accessed: 2024-03-01.

[71] Rock Yuren Pang, Ruotong Wang, Joely Nelson, and Leilani Battle. 2022. How do data science workers communicate intermediate results?. In *2022 IEEE Visualization in Data Science (VDS)*. IEEE, 46–54.

[72] Peter Pirolli and Stuart Card. 1999. Information foraging. *Psychological review* 106, 4 (1999), 643.

[73] Peter Pirolli and Stuart Card. 2005. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, Vol. 5. McLean, VA, USA, 2–4.

[74] Xiaoying Pu, Sean Kross, Jake M Hofman, and Daniel G Goldstein. 2021. Datamations: Animated explanations of data analysis pipelines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[75] John Rieman. 1993. The diary study: a workplace-oriented research tool to guide laboratory efforts. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. 321–326.

[76] Adam Rule, Aurélien Tabard, and James D Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[77] Daniel M Russell, Mark J Stefik, Peter Pirolli, and Stuart K Card. 1993. The cost structure of sensemaking. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. 269–276.

[78] Advait Sarkar. 2016. *Interactive analytical modelling*. Technical Report UCAM-CL-TR-920. University of Cambridge, Computer Laboratory. https://doi.org/10.48456/tr-920

[79] Advait Sarkar. 2022. Is explainable AI a race against model complexity?. In *Workshop on Transparency and Explanations in Smart Systems (TeXSS), in conjunction with ACM Intelligent User Interfaces (IUI 2022) (CEUR Workshop Proceedings, 3124)*. 192–199. http://ceur-ws.org/Vol-3124/paper22.pdf

[80] Advait Sarkar. 2023. Should Computers Be Easy To Use? Questioning the Doctrine of Simplicity in User Interface Design. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI EA '23)*. Association for Computing Machinery, New York, NY, USA, Article 419, 10 pages. https://doi.org/10.1145/3544549.3582741

[81] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *arXiv preprint arXiv:2208.06213* (2022).

[82] Advait Sarkar, Mateja Jamnik, Alan F Blackwell, and Martin Spott. 2015. Interactive visual machine learning in spreadsheets. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 159–163.

[83] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. 2013. A survey of intelligent assistants for data analysis. *ACM Computing Surveys (CSUR)* 45, 3 (2013), 1–35.

[84] Vidya Setlur and Melanie Tory. 2022. How do you converse with an analytical chatbot? revisiting gricean maxims for designing analytical conversational behavior. In *Proceedings of the 2022 CHI conference on human factors in computing systems*. 1–17.

[85] Ben Shneiderman. 2003. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*. Elsevier, 364–371.

[86] Sangho Suh, Meng Chen, Bryan Min, Toby Jia-Jun Li, and Haijun Xia. 2024. Structured Generation and Exploration of Design Space with Large Language Models for Human-AI Co-Creation. (2024).

[87] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling multilevel exploration and sensemaking with large language models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–18.

[88] Jiao Sun, Q Vera Liao, Michael Muller, Mayank Agarwal, Stephanie Houde, Kartik Talamadupula, and Justin D Weisz. 2022. Investigating explainability of generative AI for code through scenario-based design. In *27th International Conference on Intelligent User Interfaces*. 212–228.

[89] 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. (2024).

[90] Yuan Tian, Zheng Zhang, Zheng Ning, Toby Jia-Jun Li, Jonathan K Kummerfeld, and Tianyi Zhang. 2023. Interactive text-to-SQL generation via editable step-by-step explanations. *arXiv preprint arXiv:2305.07372* (2023).

[91] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[92] Helena Vasconcelos, Gagan Bansal, Adam Fourney, Q Vera Liao, and Jennifer Wortman Vaughan. 2023. Generation probabilities are not enough: Exploring the effectiveness of uncertainty highlighting in AI-powered code completions. *arXiv preprint arXiv:2302.07248* (2023).

[93] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–30.

[94] Dakuo Wang, Josh Andres, Justin D Weisz, Erick Oduor, and Casey Dugan. 2021. Autods: Towards human-centered automation of data science. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–12.

[95] Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. 2020. Assessing and restoring reproducibility of Jupyter notebooks. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 138–149.

[96] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[97] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging code and interactive visualization in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 152–165.

[98] 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 Transactions on Visualization and Computer Graphics* (2022).

[99] Cong Yan and Yeye He. 2020. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1539–1554.

[100] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.

[101] Ryan Yen, Jiawen Zhu, Sangho Suh, Haijun Xia, and Jian Zhao. 2023. CoLadder: Supporting Programmers with Hierarchical Code Generation in Multi-Level Abstraction. *arXiv preprint arXiv:2310.08699* (2023).

[102] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, et al. 2022. Natural language to code generation in interactive data science notebooks. *arXiv preprint arXiv:2212.09248* (2022).

[103] JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–21.

[104] Amy X Zhang, Michael Muller, and Dakuo Wang. 2020. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–23.

[105] Qiyu Zhi and Ronald Metoyer. 2020. Gamebot: A visualization-augmented chatbot for sports game. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–7.