

DANGO: A Mixed-Initiative Data Wrangling System using Large Language Model

Wei-Hao Chen
chen4129@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Amanda Case
amanda-case@uiowa.edu
University of Iowa
Iowa City, Iowa, USA

Weixi Tong
weixitong@hust.edu.cn
Huazhong University of Science and Technology
Wuhan City, Hubei, China

Tianyi Zhang
tianyi@purdue.edu
Purdue University
West Lafayette, Indiana, USA

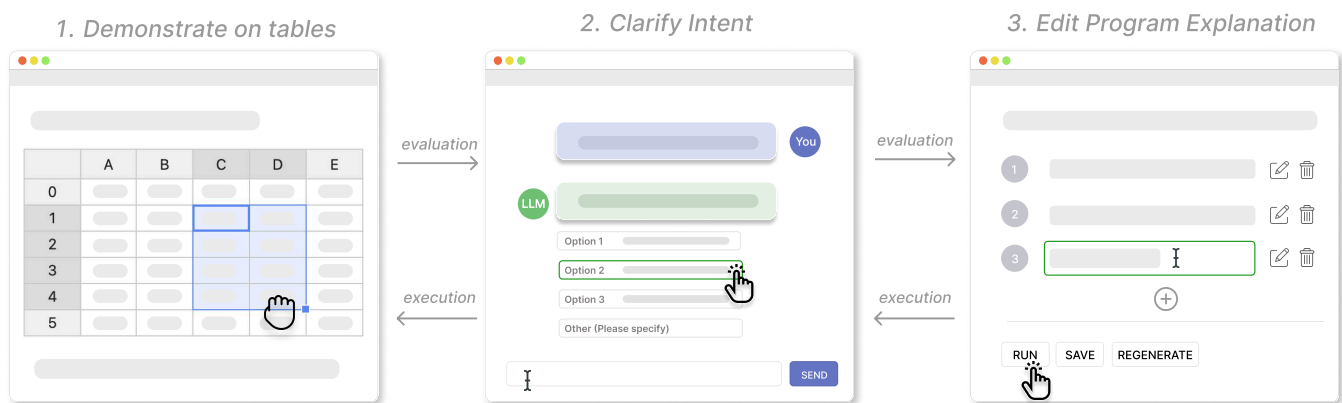


Figure 1: To create a data wrangling script, users can first demonstrate their desired actions on the tables in DANGO (*Step1*). They can edit table cells, add/delete/move columns and rows, and copy/cut content across multiple tables. For complex demonstrations, users can also describe their intent in natural language in a chatroom. When ambiguity is detected in a demonstration or a NL description, DANGO will generate a multiple-choice question about each unclear part and prompt users for clarification (*Step2*). Once the ambiguity is resolved, DANGO synthesizes a data wrangling script to automate the desired actions. To make it easier for users to understand and validate the synthesized script, DANGO explains the script in natural language step by step (*Step3*). When users notice a wrong step in the script, they can easily fix it by directly editing the NL explanation of that step. They can also add missing steps or delete redundant steps. DANGO will update the script based on user edits.

Abstract

Data wrangling is a time-consuming and challenging task in a data science pipeline. While many tools have been proposed to automate or facilitate data wrangling, they often misinterpret user intent, especially in complex tasks. We propose DANGO, a mixed-initiative multi-agent system for data wrangling. Compared to existing tools, DANGO enhances user communication of intent by: (1) allowing users to demonstrate on multiple tables and use natural language prompts in a conversation interface, (2) enabling users to clarify their intent by answering LLM-posed multiple-choice clarification questions, and (3) providing multiple forms of feedback such as step-by-step NL explanations and data provenance to help users

evaluate the data wrangling scripts. We conducted a within-subjects user study ($n=38$) and demonstrated that DANGO's features can significantly improve intent clarification, accuracy, and efficiency in data wrangling. Furthermore, we demonstrated the generalizability of DANGO by applying it to a broader set of data wrangling tasks.

CCS Concepts

• **Human-centered computing** → **Interactive systems and tools.**

Keywords

Data Wrangling, Data Science, Large Language Model

ACM Reference Format:

Wei-Hao Chen, Weixi Tong, Amanda Case, and Tianyi Zhang. 2025. DANGO: A Mixed-Initiative Data Wrangling System using Large Language Model. In *CHI Conference on Human Factors in Computing Systems (CHI '25)*, April 26-May 1, 2025, Yokohama, Japan. ACM, New York, NY, USA, 28 pages. <https://doi.org/10.1145/3706598.3714135>

1 INTRODUCTION

Data wrangling is a time-consuming and challenging task in the early stages of a data science pipeline [17, 86]. It is reported that data scientists spend up to 80% of their time changing table layouts, transforming data formats, and filling in missing or incorrect values [80]. To address this challenge, many interactive systems have been developed to help users clean data [39, 41, 54, 55]. However, they have two major limitations. First, they are limited to single-table tasks, while a recent study by Kasica et al. [56] has shown that many data wrangling tasks involve multiple tables. Second, existing systems primarily rely on user demonstration to interpret user intent, in which the user demonstrates desired data transformations on a few data points and the system generates a generalizable script for automation. However, demonstration is not convenient to specify certain transformations, such as deleting a row only if more than a quarter of the values are missing. It is difficult to precisely communicate the criterion “more than a quarter of the values are missing” via demonstration alone.

The rise of Large Language Models (LLMs) provides new opportunities for augmenting or re-designing data wrangling tools. LLM systems have shown their ability to engage in natural language (NL) conversation with humans [8, 34], lowering the entry points for novice users to interact with more complex systems. More recently, several systems have been proposed to leverage LLMs for data wrangling [24, 68, 94]. Despite their potential, these systems have several usability issues. First, they only allow NL interaction. While convenient, NL is inherently ambiguous and can be cumbersome when specifying certain operations compared with demonstrations, such as moving a specific row from one table to a target position in another table. Second, these systems do not handle the hallucination problem [52] and provide little support for users, especially end-users, in identifying and rectifying the errors in LLM-generated data transformations. Moreover, though LLMs are likely to misinterpret user intent, these systems do not provide effective ways for users to clarify their intent or provide feedback.

To address these issues, we propose DANGO, a mixed-initiative system that enhances the communication between LLM agents and users for data wrangling. DANGO supports both direct demonstration and NL interaction to enable rich intent expression. When user intent is ambiguous, DANGO generates clarification questions (CQs). Users can refine their intent by answering these CQs. The Q&A history is then incorporated into a feedback loop to continuously improve the results. To help users understand the script’s behavior, DANGO renders the synthesized scripts as step-by-step explanations in NL. Users can read these NL explanations easily and make direct edits to them. Given the complexity of multi-table tasks, DANGO automatically visualizes data provenance, helping users understand the interrelationships between tables.

We conducted a within-subjects user study with 38 participants to evaluate the usability and efficiency of DANGO. Participants using DANGO finished the assigned tasks in 3 min and 30 sec on average, reducing the task completion time by 32% and 45% compared with using the other two conditions. In the post-task survey, participants felt more confident about the generated scripts when using DANGO. To understand the generalizability of DANGO, we

conducted a quantitative experiment on 24 additional data wrangling tasks. The result shows DANGO can solve all of the tasks with an average task completion time of 62.67 and 75.90 seconds by the first two authors, respectively. This provides quantitative evidence about DANGO’s effectiveness on a variety of data wrangling tasks.

2 RELATED WORK

2.1 Data Wrangling

Data wrangling is a critical but time-consuming process in data science. A recent study by Muller et al. [80] points out that even professional data scientists may take weeks, if not months, to wrangle data to achieve high data quality. This process has been reported to be one of the most time-consuming and laborious stages in a data science pipeline [29, 55, 80, 86].

To help end-users wrangle data, researchers have developed a variety of interactive data wrangling systems to reduce manual efforts. One of the earliest systems is Nix’s Editing by Example system [83], which can automatically generate a replayable editing program by inferring users’ input/output example text. Later, Witten and Mo [78, 105] introduced the TELS system, which allows users to demonstrate text editing actions and uses these demonstration traces to synthesize text transformations. To improve the transparency of synthesized transformation scripts, Blackwell et al. developed SWYN [18] that enables users to select text examples and preview the effects of the induced scripts. Similarly, Toped⁺⁺ [91] allows users to create “topes” which are graphical representations of the data format used by the system to infer reformatting rules. Further developments such as Potluck [50] and Lapis [76] enable users to edit strings simultaneously at different locations. SMARTedit [62] applied Programming-by-Demonstration (PBD) to automate text processing tasks by allowing users to demonstrate desired actions. Karma [99] introduced a clean-by-example approach that allows users to specify the desired format of cleaned data.

While the aforementioned systems focus on wrangling unstructured data such as text, a parallel line of research focuses on wrangling structured data such as tables or spreadsheets [90]. Wrangler [42, 55], for example, employed both PBD and direct manipulation to recommend applicable transformations based on user demonstrations on tables. Gulwani developed a new PBE algorithm [38], which is later used in Microsoft Excel known as Flash-Fill [38], allowing users to synthesize programs for string transformations. Harris and Gulwani [43] later developed another PBE system that supports table structure transformation. To help data scientists wrangle data, Wrex [30] applied PBE to synthesize wrangling scripts in Python within Jupyter Notebooks. Other researchers have also employed synthesis techniques to perform data transformations [32, 45, 54, 62, 64, 107]. Despite their advancements, these systems only focus on synthesizing code for single-table tasks, while a recent study by Kasica et al. [56] found that data wrangling tasks usually involve multiple tables. In contrast, DANGO enables users to synthesize code for multi-table tasks.

The most related tools to our work are those that leverage LLM agents for wrangling tables and spreadsheets [16, 24, 68, 87, 94, 113]. However, these systems only support one-shot synthesis and lack mechanisms for users to clarify or refine their intent, resulting in errors like incomplete solutions and wrong actions [68]. This issue

can be seen as the **gulf of execution** [49]: *the challenge users face in clearly expressing their intent to instruct AI*. To bridge this gap, DANGO enables users to clarify their intent by answering AI-posed clarification questions before synthesis. Users can keep refining their intent through natural language instructions or demonstrations until the desired result is achieved.

2.2 LLM-based Interactive System

The recent advent of Large Language Models (LLMs), such as OpenAI’s GPT models [8, 34], Google’s PaLM [13], and Meta’s Llama [98], has significantly influenced interactive systems across various domains [35]. For instance, Microsoft has integrated Copilot [75] into their suite of productivity tools. Amazon has developed a custom-built LLM for Alexa [12]. In parallel, HCI researchers have also infused LLM into their system designs, revolutionizing fields in writing [26, 77, 109], education [57, 71], video editing [102], programming [74, 79], health [67], AR/VR [25, 73], visualization [101, 103], data science [36, 37, 106], among others. These LLM-based systems have introduced new interactive paradigms, shifting from traditional interactions—where users instruct computers *what to do*—to a more AI-driven approach—where the user specifies *what results they want*, a new interaction identified as *intent-based outcome specification* by Nielsen [82].

However, recent studies show that these LLM-based systems could also misinterpret user intent due to the inherent ambiguity of NL instructions [28, 37, 72, 92], leading to misalignment between AI and human intent [93]. To address these issues, many studies have used clarification questions (CQs) to resolve ambiguous queries and improve user experience [10, 28, 31, 97, 104, 116]. For instance, Danry et al. [28] found AI-framed Questioning increases human discernment accuracy of flawed statements. In the information retrieval community, CQs have been found useful to help user clarify their needs in search engines [10, 104, 116]. Similarly, in the software engineering domain, Eberhart et al. [31] introduced an approach for generating CQs to refine user queries in code search.

Despite its usefulness, clarifying ambiguous user intent using CQs has received limited attention in data wrangling. To the best of our knowledge, no data wrangling tools have utilized CQs to assist users in clarifying their intent. Traditional synthesis methods [30, 38, 43, 55], such as PBD or PBE, typically lack effective mechanisms for refining user intent. In this work, DANGO uses both PBD and a conversational interface powered by LLM to help users express their intent. When the user intent is unclear, DANGO asks CQs to help refine their intent. In addition, DANGO provides multiple feedback modalities, such as data provenance visualization and step-by-step program explanation, to help users evaluate the results, creating a rich feedback loop for effective data wrangling.

3 DESIGN GOALS AND RATIONALE

3.1 Design Goals

We reviewed prior work on data wrangling, especially those that have conducted user studies or discussed usability challenges in data wrangling tools [24, 30, 36, 39, 40, 55, 56, 61, 68, 94]. We summarized four major design goals for DANGO based on the common issues and challenges identified by previous work.

G1. Help users express and clarify their intent. Previous tools [30, 39, 55, 61] primarily use PBD to synthesize data wrangling scripts based on examples or demonstrations. However, many studies [60, 65, 81, 112] showed that users find it hard or cumbersome to provide complex examples. Recently, the rise of LLMs has brought new opportunities for facilitating data wrangling tasks with NL interfaces powered by LLM agents [16, 24, 68, 87, 94, 113]. While this enriches the interaction modes, the inherent ambiguity of natural language also brings new challenges to interpreting user intent. According to a recent study by Gu et al. [37], LLM agents could easily misinterpret user intent and generate incorrect operations. Moreover, instructing LLMs via natural language can be a brittle experience for non-AI-experts [110]. They often struggle with finding the right narratives to effectively communicate with LLMs. Therefore, it is critical to help users better express their intent and more importantly, clarify their intent in case of misinterpretations.

G2. Help users wrangle multiple tables. Previous data wrangling tools [30, 39, 55, 61] only support single-table interactions. However, a recent study by Kasica et al. [56] reveals that as data complexity and volume increase these days, data wrangling tasks often involve multi-table operations. Specifically, they identified 21 multi-table operations that are challenging to demonstrate and synthesize using existing synthesis methods. For instance, an auditor may need to split tables of annual transaction records over the past 20 years into subtables based on the product category and then only remove rows with missing values in each subtable. Currently, there is little tool support for such multi-table operations [56]. Thus, the new tool should be able to interpret user demonstrations or NL descriptions that involve multi-table operations and synthesize scripts to perform these operations.

G3. Help users interpret and validate the automation results. Compared with other automation tasks, data wrangling faces a unique challenge due to the large volume of data involved in a task. It is hard and sometimes time-prohibitive to manually validate the wrangling results [37, 84]. While PBD systems synthesize a program that prescribes the operations behind the wrangling results, it is difficult for end-users to understand the program, which is one of the main reasons that hinder the widespread adoption of PBD systems [40, 60]. Specifically, in the reflection paper on why PBD fails, Lau mentioned the system should “*encourage trust by presenting a user-friendly model*” [60]. This echoes the *understanding barrier* and the *information barrier* in end-user programming [59]. More recently, Gu et al. [37] reveal the challenges of interpreting and validating data analysis results, such as not being able to translate the semantics of the analysis code to familiar operations in their mental models. Therefore, it is critical to design effective mechanisms to help users quickly interpret and validate the results or the code that computes the automation results.

G4. Help users make corrections. For PBD systems, the de-facto way of fixing an incorrect script is to provide more demonstrations to refute the incorrect script so that the PBD system can better extrapolate and avoid overfitting. However, this is inefficient, especially for minor errors, as the users must demonstrate from the beginning every time. Several studies have shown that end-users prefer simpler and more efficient error correction methods [46, 61]. Wrangler [55] and Wrex [30] address this by enabling

users to directly manipulate the synthesized script. However, Wrangler is limited to simple scripts, and Wrex still requires program comprehension and thus is not suitable for end-users. More recent LLM-based data wrangling systems [24, 68] allow users to make corrections by refining the initial prompt or providing NL feedback in the conversation. While this eliminates the need for directly modifying the synthesized script, users still need to overcome the communication barriers to LLMs [27, 110]. In particular, a recent study [110] shows that non-AI-experts often struggle with finding the right prompt to effectively steer the output of an LLM. These challenges highlight the need to help users make corrections more efficiently, especially for LLM-based data wrangling tools.

3.2 Design Rationale

We made two specific design choices to help users to express and clarify their intent (**G1**). First, DANGO allows users to express their intent using a combination of demonstration and natural language conversations. Both interaction modalities are convenient for end-users without a steep learning curve. Moreover, they are complementary to each other. Some actions, such as editing a specific cell and moving a row from one table to a specific position in another table, are easy to demonstrate by directly performing them on the table, while some actions are hard to demonstrate but easier to describe in natural language, such as specifying conditions for an action or performing a statistical test.

Second, DANGO leverages the intrinsic reasoning capability of LLMs to proactively detect potential ambiguity in user input and ask clarification questions. This proactive interaction design is motivated by previous findings that users often struggle with identifying which parts of their input lead to the erroneous output and how to effectively clarify their intent [37, 72, 110]. Furthermore, when users prefer or find it necessary, they can still clarify their intent by directly providing additional demonstrations or NL feedback in DANGO. This allows flexibility and forms a mixed-initiative interaction experience where the human and the agent take turns to contribute at the perceived appropriate time [11].

To achieve **G2**, we extended a popular DSL for single-table data wrangling [55, 88] to support multi-table operations, as detailed in Appendix A. We chose to synthesize scripts in a DSL rather than a general-purpose programming language such as Python, since the DSL has a much simpler grammar with highly abstracted operators, which makes it easier to synthesize and explain. Furthermore, we chose to leverage an LLM to synthesize a script instead of using traditional search-based synthesis algorithms, since traditional algorithms are only designed to interpret demonstrations. They cannot handle NL descriptions that mention desired operations. Specifically, DANGO encodes the demonstrations in NL and feeds them together with any NL description provided by users to the LLM for joint interpretation and synthesis. Finally, to help users effectively manage operations on multiple tables, DANGO automatically tracks and visualizes the data provenance across multiple tables.

To achieve **G3**, an intuitive approach is to use LLMs to explain a synthesized script in natural language. However, LLMs are prone to hallucinations [66]. Besides, a recent study [37] shows that errors in LLM-generated code are often subtle and thus require users to carefully scrutinize each step of the analysis procedure. Inspired

by recent systems that support step-by-step explanations [23, 96], we employ a grammar-based translation method to translate a generated script into step-by-step explanations in NL. This prevents LLM hallucination while providing a structured way for users to understand and verify the script. Another alternative approach is to render the program into a visual language [21, 89]. Although visually appealing, this approach still requires users to understand basic programming constructs such as variables and loops. In addition to NL explanations, we also allow users to run a synthesized script and observe its runtime behavior to confirm their understanding of the script and cross-validate its correctness.

To achieve **G4**, we allow users to directly edit the NL explanation of the data wrangling script to fix any recognized errors or describe expected behavior. The updated explanation will be fed back to the LLM to refine the script. The rationale for this choice is two-fold. First, we want to continue leveraging the structured NL explanation in error correction, so users can immediately fix an error in situ as they are reading and scrutinizing each wrangling step in the explanation without context switching. Second, compared with two alternative approaches—providing more demonstrations or providing NL feedback in a conversation, editing individual steps in the explanation allows users to directly indicate where the errors are, which saves the effort for error localization and enables the LLM to perform a more focused script refinement. Nevertheless, we still support these two alternative error correction methods in DANGO for flexibility. We measured the utility of each approach and reported the results in Section 7.2.5 and Figure 8.

4 SYSTEM DESIGN

As shown in Figure 2, DANGO includes three stages—*Intent Analysis*, *DSL-based Program Synthesis*, and *Program Evaluation & Refinement*. We adopt the multi-agent LLM framework to develop DANGO. LLMs [8, 34] have demonstrated strong capabilities in natural language understanding, contextual reasoning, and multi-turn conversations. For each stage, we develop a dedicated agent by leveraging the in-context learning capability of LLMs with few-shot prompting. Appendix B provides all prompt templates used in DANGO.

4.1 Intent Analysis

In this stage, DANGO leverages an analysis agent to analyze user intent from the user’s demonstrations and conversation history. If the intent is ambiguous, DANGO generates clarification questions. After the user answers the CQs and there is no ambiguity, DANGO generates a summary of user intent for the next stage to synthesize the desired data wrangling script.

4.1.1 User Demonstration. To capture user demonstrations, we set an event listener for every table entry in the uploaded tables. When a user modifies the table, the event listener logs the change in the demonstration history. For example, if a user changes a table entry from 0 to 3370 at row 5, column E, DANGO logs this edit as {Row: 5, Column: E, Old: 0, New: 3370, Type: Edit}. Specifically, DANGO supports the following demonstrations on the uploaded tables:

1. **[INSERT]**: insert empty columns or rows.
2. **[DELETE]**: delete columns or rows.
3. **[EDIT]**: edit specific cells.
4. **[COPY AND PASTE]**: copy and paste columns and rows.

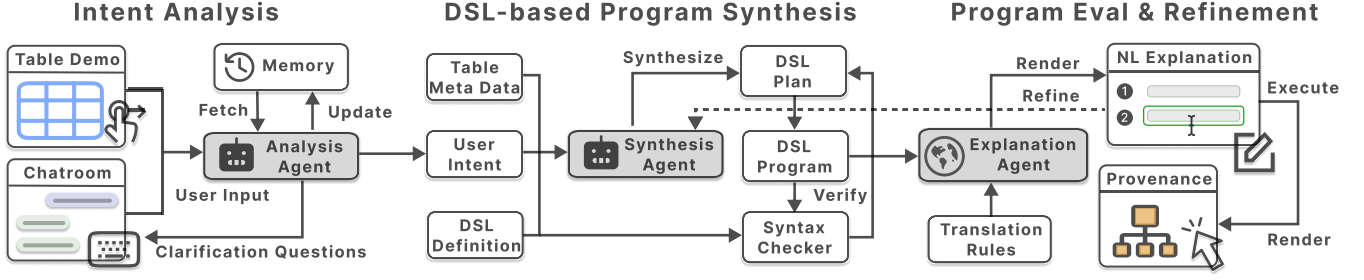


Figure 2: Three core components: Intent Analysis, DSL-based Program Synthesis, and Program Evaluation & Refinement.

5. [DRAG AND DROP]: drag and drop columns and rows.

If an edit operation is applied to every cell in a row or column, DANGO merges these individual operations to a single row/column edit operation in the demonstration history to save space.

4.1.2 Conversational Interface (Chatroom). DANGO provides a conversational interface that enables users to express their intent or feedback in NL. Moreover, when the user intent is unclear, DANGO poses multiple-choice clarification questions to help users refine their intent. Each CQ includes different options and an “Other (please specify)” option, allowing users to provide customized responses when needed, as shown in Figure 3(f).

4.1.3 Clarification Questions Generation & Intent Summarization. DANGO prompts an LLM to detect ambiguities and generate clarification questions. If no ambiguity is detected, the LLM summarizes the user intent into an NL sentence. Table 4 in Appendix B shows the prompt template. Specifically, user demonstrations are encoded as “Table Diff” in the INPUT section of the prompt, while NL inputs in the chatroom are encoded as “User Instruction”. After the user answers a clarification question, the answer is appended to “Chat History” to identify any remaining ambiguities, as shown in Table 5.

4.2 DSL-based Program Synthesis


Once users define their intent, DANGO employs a synthesis agent to generate step-by-step data wrangling scripts.

4.2.1 Domain Specific Language. DANGO’s Domain Specific Language is built on the foundational work of Potter’s Wheel [55, 88]. We use this DSL because it has been adopted and validated in several existing work [42, 54, 55]. Since the original DSL only supports single-table operations, we extended the original DSL to support multi-table operations and advanced operations such as statistical tests. As shown in Appendix A, our extended DSL supports four types of operations: (1) *Table-Level Operations*, which operate on entire tables (e.g., merge, transpose); (2) *Column/Row-Level Operations*, which modify specific rows or columns (e.g., insert, drop); (3) *Summarization Operations*, which aggregate data into new tables or statistical values (e.g., aggregate, test); and (4) *String Operations*, which manipulate text transformations (e.g., concatenate, format).

4.2.2 Plan Generation. DANGO adopts a self-planning strategy [53, 108] to first prompt the LLM to generate a synthesis plan for task decomposition. Each step in the plan contains a NL description of

the desired action and the specific DSL operation that can achieve this action. Table 6 shows the prompt template.

4.2.3 DSL Program Synthesis. After generating the step-by-step plan, DANGO curates the arguments according to the DSL grammar specifications. To prevent potential hallucinations, we designed a syntax checker to statically check the correctness of the operation’s arguments based on our DSL grammar. If verification fails, DANGO appends error messages to the input and prompts the synthesis agent to generate a refined script. The prompt templates used in this step are detailed in Tables 7, 8 and 9.

4.2.4 DSL Program Execution. When users click  button to execute the script, DANGO first imports the DSL function implementations into the execution environment. DANGO then synthesizes a Python code snippet that uses these DSL functions, following the LLM prompt in Table 10 in Appendix B. The snippet is executed using Python’s `exec()` command [3]. This approach facilitates dynamic code execution and handling of conditional operations. Note that the original table content will not be altered. Instead, new versions of the table are named with sequential version numbers (e.g., `table_v1`, `table_v2`). This idea is inspired by versioning systems [58, 111, 115].

4.3 Program Evaluation & Refinement

4.3.1 Step-by-Step Natural Language Explanation. To help users understand the generated script, DANGO translates each DSL statement into a natural language explanation. To prevent potential hallucinations, we design translation rules for each statement, as shown in Table 15 in Appendix C. For instance, given a DSL statement `delete_table(sales.csv)` in a script, DANGO translates the statement based on a text template—“Delete the table” + “X”. In this template, the name X will be replaced by the first argument `sales.csv` to compose the final string “Delete the table sales.csv.” For statements with conditions, DANGO appends the condition string to the explanation. For instance, the final string will be “Delete the table sales.csv if there are more than 30% of missing values.”

4.3.2 DSL Program Refinement. Due to potential hallucinations of the LLM agents used in our system, DANGO might synthesize erroneous scripts that are not desired. To help users refine the synthesized program, DANGO allows users to refine the synthesized scripts through the following actions (Figure 3(g)).

1. [EDIT]: edit a step in the NL explanation.
2. [DELETE]: delete a step.

StudentID	Name	Gender	Course A	Course B
00336617	John	Male	9	N/A
...				
00770000	Elisa	Female	7	6

StudentID	Name	Gender	Course C	Course D
00440033	Alice	Female	3	6
...				
00993399	Ben	Male	5	5

Name	StudentID	Course A	Course B	Course C	Course D
Alice	00440033	5	7	3	6
...					
Yen	00990033	5	4	7	5

Table 1: The two raw tables (shown at the top) and the desired cleaned table (shown at the bottom).

3. [ADD]: add a new step in NL.
4. [SAVE]: save the current script.
5. [REMOVE]: remove the current script.
6. [REGENERATE]: regenerate the script.

If the user only adds a step without changing other steps, DANGO will prompt the LLM to simply update the script with the newly added step using the prompt template in Table 11. For other types of edits, DANGO will prompt the LLM to regenerate the DSL script based on the previous script using the prompt template in Table 12. In both cases, DANGO will prompt the LLM to re-summarize the user intent for further usage, e.g., in the next round of script generation or refinement.

4.3.3 Data Provenance View. Multi-table operations usually involve complex data dependencies. This may increase the user’s cognitive overload to memorize each change and the inter-dependency between tables. Thus, DANGO visualizes *data provenance* to allow users to track different versions of tables and their dependencies. When a user executes the scripts, DANGO’s backend calculates the data dependency by analyzing the data flow between different tables. For instance, when $C = \text{MERGE}(A, B)$, DANGO will render two directional edges from $A \rightarrow C$ and from $B \rightarrow C$. Each node represents a specific version of a table, with directional edges showing dependencies between tables. The dependency is done at the table level, not at a more fine-grained level such as row or column. This is because finer granularity would make the provenance view large and overwhelming, also known as *flood of information* [9, 19]. To see the content of a certain table at a certain version, users can click nodes in the *Data Provenance* view panel. The corresponding table content will appear in the *Table View* panel. This creates a visual correspondence, or *brushing* [14], allowing users to interactively check corresponding table content.

4.4 System Implementation

DANGO’s frontend is developed with React and Vite [6]. For state management, DANGO employs Zustand [7]. DANGO uses Tailwind CSS [5] to create a responsive user interface. The user interface incorporates Handsontable [2], providing an intuitive experience for spreadsheet tables, as well as React Flow [4], providing Node-Based UIs for the data provenance feature. DANGO’s backend incorporates FastAPI [1], a web framework for building backend APIs with Python. We use OpenAI’s gpt-4o-mini [48] as the LLM for DANGO. We set the temperature to 0 to make our experiments reproducible while keeping all other hyperparameters at their default settings. In particular, the default setting of `top_p` is 1, `frequency_penalty` is 0, and `presence_penalty` is 0.

5 USAGE SCENARIO

Beth is a professor in the Department of Education who wants to study the students’ ratings on different courses to improve teaching quality and student satisfaction. She has run two separate surveys for different courses and collected student ratings. The raw data have been downloaded into two separate tables, as shown in Table 1. Since manually cleaning data is time-consuming, she decides to use DANGO to generate an automated script.

She first uploads two raw tables to DANGO. Next, she clicks the **RECORD** button (Figure 3(d)) and demonstrates to DANGO how to clean the data. She deletes the Gender column and then moves the Name column ahead of StudentID. After her demonstration, DANGO generates a multiple-choice clarification question in the chatroom, asking if she wants to apply the same changes to Table 2. Beth confirms by selecting the “Yes” option (Figure 4(a)). DANGO then generates the DSL script and renders the step-by-step NL explanation of the script, as shown in Figure 4(c).

Beth reads this step-by-step NL explanation and confirms that it matches her intent. She clicks the **RUN** button and soon notices that the data provenance view is rendered (Figure 3(e)). She clicks the “Table2_v1.csv” node in the data provenance view and confirms that the transformations to Table 2 are correct.

Beth now wants to merge these two tables. She types in the chatroom: “Merge the tables into a single table by matching the StudentID”. DANGO then generates a script and its explanation (Figure 4(d)). This looks correct to Beth, so she clicks **RUN** to perform the action. Then, Beth notices some missing values in the merged table. She then types, “Please eliminate rows if containing an excessive amount of missing values. Then populate the remaining missing fields with column mean values.” DANGO then generates a clarification question to ask Beth to specify the threshold for excessive missing values (Figure 4(b)). Beth selects 30% as the threshold. DANGO generates the new script accordingly based on the clarification (Figure 4(e)).

Later, Beth changes her mind and clicks the **EDIT** button (Figure 4(f)), adjusting the threshold from 30% to 50%. She then clicks the **+ ADD A STEP** button (Figure 4(g)) and types in a new step: “Sort the table alphabetically by the values in the column Name.” Next, she clicks the **REGENERATE** button (Figure 4(h)) to obtain the new script. Lastly, Beth runs the script and gets the final table.

6 EVALUATION

We investigated five research questions in the evaluation of DANGO. To evaluate the usability and efficiency of DANGO, we conducted an IRB-approved within-subjects user study with 38 participants on 7 data wrangling tasks (RQ1-4). To understand the generalizability

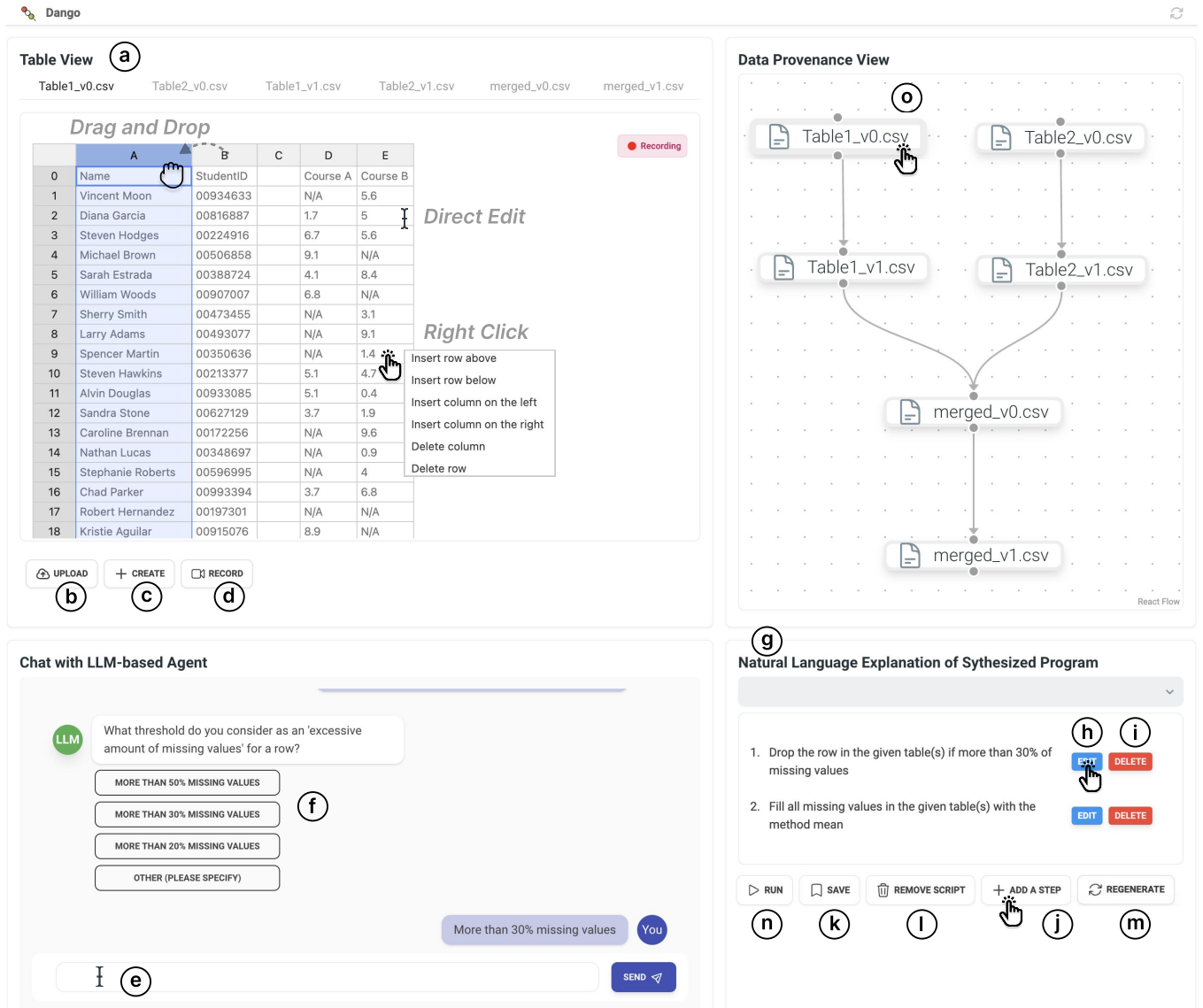


Figure 3: User interface of DANGO. In the table view (a), users can upload tables (b) or create new tables (c). Then, they can click the RECORD button (d) and start demonstrating their desired actions. Alternatively, they can express desired actions in natural language in a chatbox (e). DANGO will interpret the demonstrations and/or NL descriptions in the backend and generate multiple-choice clarification questions when needed (f). Furthermore, to help users understand and validate the synthesized script, DANGO explains it in NL step by step (g). Users can directly edit a step in natural language (h), delete a step (i), add a new step (j), save the script (k), remove the script (l), or regenerate the script (m). Users can click the RUN button (n) to execute the script on copies of the original tables and verify its behavior without messing up the original demonstrations. DANGO also renders a data provenance view to track the transformations performed on each table (o). Users can click table nodes, and the corresponding table content will appear in the table view.

of DANGO, we conducted a quantitative study with 24 additional tasks (RQ5).

- RQ1.** How different interaction paradigms of DANGO can help users in data wrangling tasks? DANGO supports multiple interaction paradigms for user intent clarification and refinement. This RQ aims to explore how these interaction paradigms contribute to data wrangling effectiveness by comparing DANGO with its variants.
- RQ2.** How do users perceive and value different features of DANGO? This RQ evaluates user perceptions of different features in DANGO by analyzing feature ratings and qualitative feedback.
- RQ3.** Why does DANGO help users perform data wrangling tasks more effectively? This RQ aims to get a deeper understanding of DANGO's effectiveness by analyzing and summarizing the common reasons why participants liked DANGO mentioned in the post-study survey.

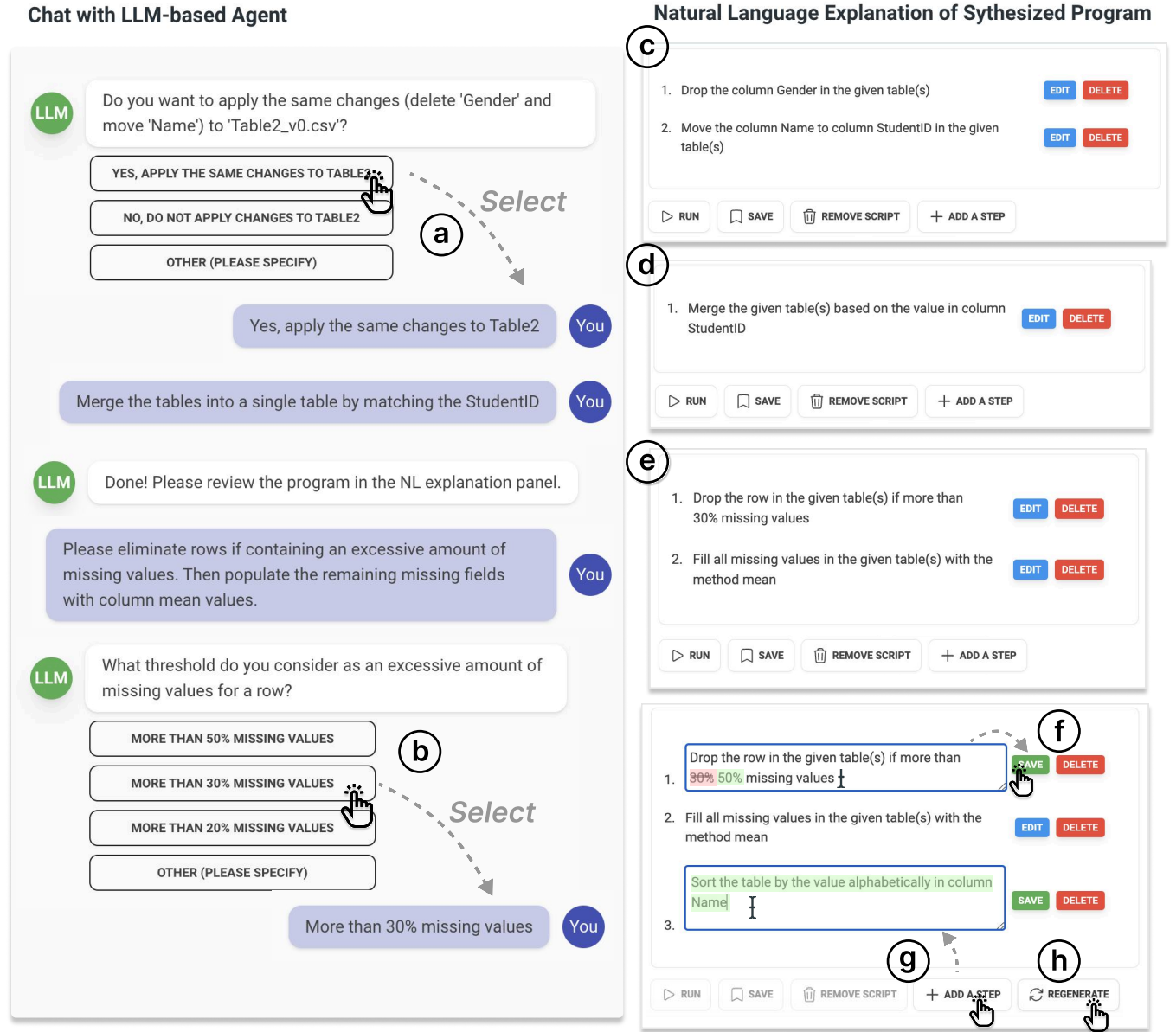


Figure 4: This figure shows a usage scenario of users using a chatroom to clarify their intent using NL prompts and answering multiple-choice clarification questions (left-hand side). Users can easily understand the program behavior by reading the step-by-step NL explanations. They can refine their program by directly editing the step-by-step NL explanations (right-hand side)

- **RQ4.** How does user expertise level influence task performance? We envision that DANGO will not only be helpful for end-users but also improve the productivity of experienced users. Thus, we recruited participants with different levels of expertise in data wrangling in our user study. This RQ examines whether user expertise affects user performance.
- **RQ5.** How well does DANGO generalize to a broader range of data wrangling tasks? To evaluate the generalizability of DANGO, we conducted a systematic evaluation across an additional set of 24 data wrangling tasks. Furthermore, we analyze the distribution

of generated DSL statements and the effectiveness of clarifying questions (CQs) in this broader task set.

6.1 User Study Conditions

We designed three user study conditions, each of which represents a specific design of DANGO. The comparison of user performance across these conditions helped us understand the effectiveness of different interaction paradigms for users to clarify their intent and provide feedback to the generated script. The data provenance view is enabled in all three conditions.

Condition A (Demonstration + Conversational Interface + NL summary): This condition represents a naive design of DANGO. It simply integrates the two interaction modalities and prompts the LLM to generate the NL summary of a synthesized script. Table 14 shows the promote template to generate the NL summary. This NL summary does not follow the step-by-step structure and there is no special treatment to mitigate LLM hallucination. To clarify user intent or refine the synthesized script, users can provide new demonstrations and provide feedback in the chatroom. However, they cannot edit the NL summary to provide feedback.

Condition B (Demonstration + Conversational Interface + Step-by-Step NL): This condition represents an enhanced design of the NL feedback mechanisms in DANGO. In this condition, DANGO generates the step-by-step NL explanation of a script using the rule-based method proposed in Section 4.3.1. This design enables a more faithful explanation with no LLM hallucinations compared to the NL summary in Condition A. Furthermore, users can provide more fine-grained feedback to refine the synthesized script by directly editing specific steps in the NL explanation.

Condition C (Demonstration + Conversation Interface+ Step-by-Step NL + CQs): This condition represents the final design of DANGO. Compared with Condition B, this condition allows DANGO to proactively ask clarification questions to resolve ambiguities, as proposed in Section 4.1.3. This transforms from user-initiative interaction in Condition B to a collaborative mixed-initiative approach, where the system and user work together to clarify and refine the intent.

6.2 User Study Participants

We recruited a total of 38 participants (11 female, 27 male) through a mailing list at Purdue University. 32% of participants had no programming experience at all or had only taken an introductory programming course for one semester. 37% had one or more years of programming experience but less than 5 years. 31% had five or more years of programming experience. 42% of participants were undergraduate students, 11% were master's students, and 47% were PhD students. These students were from diverse majors, including Computer Science, Mechanical Engineering, Industrial Engineering, Materials Engineering, Statistics, Chemical Engineering, Physics, Agriculture, Business, Hospitality, Communication, and Game Design. Table 16 in Appendix D shows the detailed demographic information of each participant.

6.3 User Study Tasks

To develop the user study tasks, we first identified 20 data wrangling tasks from previous data wrangling work [22, 55, 68] and extended our tasks with 21 multi-table tasks based on Kasica et al.'s recent empirical work [56]. Then we discussed and refined these tasks through three meetings with a domain expert from the College of Education at Purdue University, who regularly performs data wrangling tasks such as cleaning student survey responses in spreadsheets. During the discussions, we removed 13 tasks that the domain expert thought were too simple or uncommon in practice. The domain expert also suggested 3 new tasks with conditional operations and statistical tests. From a total of 31 candidate tasks, we selected 7 most representative data wrangling tasks based on

the suggestion of the domain expert, including 2 single-table tasks and 5 multi-table tasks. Among them, 2 tasks involve conditional operations, and 1 task focuses on statistical testing Table 17 in Appendix D provides a detailed description of each task.

6.4 User Study Procedure

Each participant came to our lab for a reserved one-hour session. Participants then signed the consent form. To reduce demand characteristics [85], we told participants that our goal was to understand the influence of three different conditions of an existing tool. We did not tell them that we developed this tool or which condition represented the final design. Furthermore, to mitigate the learning effect [63], both task and condition assignment orders were counterbalanced across participants. As compensation, each participant received a \$25 Amazon gift card.

Timed Data Wrangling Tasks. Participants watched a 5-minute tutorial video and spent about 5 minutes becoming familiar with the tool. Then they completed 3 randomly assigned data wrangling tasks selected from 7 user study tasks with 3 different conditions. These assignments were counterbalanced across 3 conditions and 7 tasks, resulting in 5 or 6 trials per task per condition. Since only tasks 2 and 6 are single-table tasks, we also counterbalanced the task assignments so that each user experienced at least two multi-table tasks. Participants will first read the task description and make sure they understand the tasks. We explicitly told participants not to directly copy and paste the task description to the chatroom. A task was considered failed if participants did not generate a script that could correctly clean the data within 10 minutes.

Post-task Surveys. After completing each task, participants filled out a post-task survey to give feedback, as described in Table 18 in Appendix D. The survey first asked users to report their confidence in the generated data wrangling scripts on a 7-point scale (1—very low confidence, 7—very high confidence). Then it asked users to rate the usefulness of key features in each condition on a 7-point scale. It also asked users what they liked or disliked about the tool and what they wished they had. To evaluate the cognitive load of using a tool, we included five NASA Task Load Index questions [44] as part of the post-task survey.

Post-study Survey. After completing all tasks, participants completed a post-study survey comparing DANGO across conditions. The survey included questions on which condition was most helpful, reasons for their preferences, and open-ended feedback. Table 19 in Appendix D lists the questions from the survey.

6.5 User Study Measurements & Data Analysis

We collected both quantitative and qualitative data from the user study. We measured task completion time and the number of attempts per task. We considered a participant made another attempt when they (1) submitted an incorrect script to the experimenter or (2) started over from scratch. To gain a deeper understanding of user behavior patterns in different conditions, we measured the utility of each key feature by analyzing user events in system logs, such as starting a new demonstration and editing the NL explanation.

Moreover, DANGO heavily relies on LLMs to analyze user intent and generate clarification questions and data wrangling scripts.

Since LLMs may misinterpret user intent and hallucinate, it is important to understand how often they hallucinate during a task session. Thus, the second author watched all video recordings and examined the LLM-generated clarification questions and data wrangling scripts. A hallucination is detected if a clarification question is irrelevant to user intent or if the generated script is incorrect.

To analyze the quantitative ratings collected from the post-task surveys, we used ANOVA tests [33] to examine the statistical significance of their mean differences. To analyze the open-ended feedback, the first author conducted open coding [100] to identify themes in participants' responses, followed by thematic analysis [20]. The last author reviewed the coding results and discussed with the first author to refine the initial codes and themes. The first author then incorporated feedback, made revisions and adjustments, and finalized the themes before writing the reports.

7 EVALUATION RESULTS

7.1 RQ1: Effectiveness of Different Interaction Paradigms

7.1.1 Overall Performance. All participants successfully completed the tasks in Conditions B and C. However, in Condition A, two participants were unable to generate the correct scripts within the 10-minute time limit. The average completion time using DANGO in Condition C is 3 min 30 sec, representing a 45% decrease compared to condition A and a 32% decrease compared to condition B. An ANOVA test shows that the mean differences in time used are statistically significant (p -value = $1.76e-05$). When using DANGO in Condition C, participants made an average of 1.43 attempts across all tasks, compared to 2 and 1.66 attempts in Conditions A and B, respectively. An ANOVA test shows that the mean differences in number of attempts are statistically significant (p -value = $2.89e-02$).

The average percentage of hallucination in each task session is 0.65 in Condition A, 0.59 in Condition B, and 0.18 in Condition C. An ANOVA test revealed that the mean differences of the hallucination rate across these three conditions are statistically significant (p -value = $1.32e-02$). Furthermore, Figure 8 shows the relative temporal distribution of hallucinated scripts in each task session in each condition. These results demonstrate the effectiveness of proactively asking clarification questions in reducing hallucinations.

7.1.2 Task-Specific Performance. Table 2 shows user performance in different tasks and conditions. For Tasks 2, 5, 6, and 7, Condition C significantly reduced the task completion time compared to Condition A. Pairwise ANOVA tests show that the mean differences are statistically significant (p -value= $4.89e-02$, p -value= $4.22e-02$, p -value= $2.13e-04$, p -value= $2.77e-03$, respectively). For Tasks 5, 6, and 7, Condition C significantly reduced the task completion time compared to Condition B. Pairwise ANOVA tests show that the mean differences are statistically significant (p -value= $4.82e-02$, p -value= $1.42e-02$, p -value= $3.46e-02$, respectively). There are no statistically significant differences in Tasks 1, 3, and 4 since these tasks are easier compared with other tasks. Tasks 5, 6, and 7 include complex operations or conditional operations. This suggests that condition C's advantages become more prominent in complex scenarios that require users to clarify their intent.

7.1.3 User Confidence & User Preference. As shown in Figure 5, participants expressed a higher level of confidence when using DANGO in Condition C (Mean: 5.31 vs. 6.18 vs. 6.34). The mean differences are statistically significant (One-way ANOVA: p -value = $1.64e-04$). When asked which condition they preferred for data wrangling, 82% of participants chose Condition C.

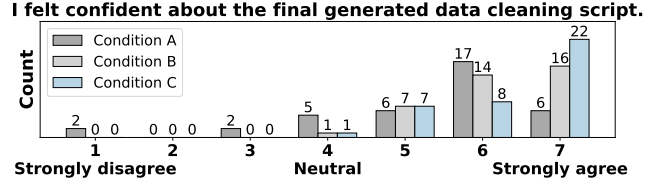


Figure 5: User confidence on the data wrangling scripts when using conditions A, B, and C.

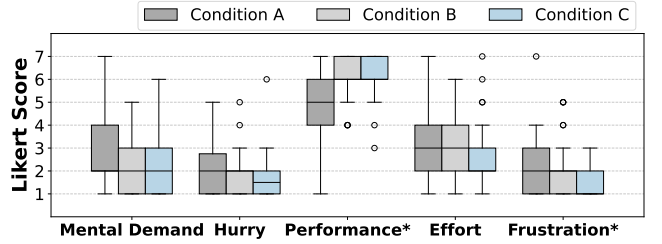


Figure 6: User responses to the NASA TLX questionnaire (*: p -value < 0.05 based on ANOVA test).

7.1.4 Cognitive Overhead. As shown in Figure 6, we detected significant differences in *Performance* and *Frustration*. The ANOVA tests reveal that the mean differences are statistically significant for both metrics (p -value = $1.42e-05$, and $2.98e-02$, respectively). For *Mental Demand*, *Hurry*, and *Effort*, we did not detect significant differences (p -value = $1.3e-01$, $3.71e-01$, and $2.77e-01$, respectively).

7.2 RQ2: User Perceptions

In the post-task survey, participants rated the usefulness of key features of DANGO in 7-point Likert scale questions (1—strongly disagree, 7—strongly agree), as shown in Figure 7.

7.2.1 Participants' perception on different intent expression features. DANGO supports three features that help users express their intent. First, users can perform demonstrations on uploaded tables. Second, users can use the chatroom to prompt their intent in natural language. Third, if their initial input is ambiguous, they can clarify their intent by answering multiple-choice clarification questions posed by the DANGO.

Demonstrations. Participants expressed mixed opinions about demonstrating on the uploaded tables, as shown in Figure 7. Only 21% of participants in Condition A, 21% in Condition B, and 29% in Condition C agreed or strongly agreed that demonstration was a convenient way to express intent. Participants mentioned that demonstrating on tables was fast and straightforward for simpler tasks (P5), but error-prone for complex tasks if demonstrations are not clear enough (P1). This suggests demonstrations might be more effective for simpler tasks but are error-prone for complex tasks.

Measure	Task 1			Task 2			Task 3			Task 4			Task 5			Task 6			Task 7		
	Condition			Condition			Condition			Condition			Condition			Condition			Condition		
	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C	A	B	C
Time	8:56	4:29	5:18	3:57	3:20	1:34	5:42	5:46	3:40	4:40	3:06	3:31	7:46	6:49	4:29	5:10	5:30	2:40	7:42	6:35	3:33
Attempts	2.67	1.60	2.00	1.00	1.50	1.00	1.75	1.75	1.67	1.83	1.50	1.00	2.00	2.00	1.75	2.00	1.80	1.33	2.40	1.40	1.20
# of CQs	—	—	0.40	—	—	0.60	—	—	1.33	—	—	0.50	—	—	3.00	—	—	1.00	—	—	1.40

Table 2: The average task completion time, average number of attempts, and average number of CQ generated.

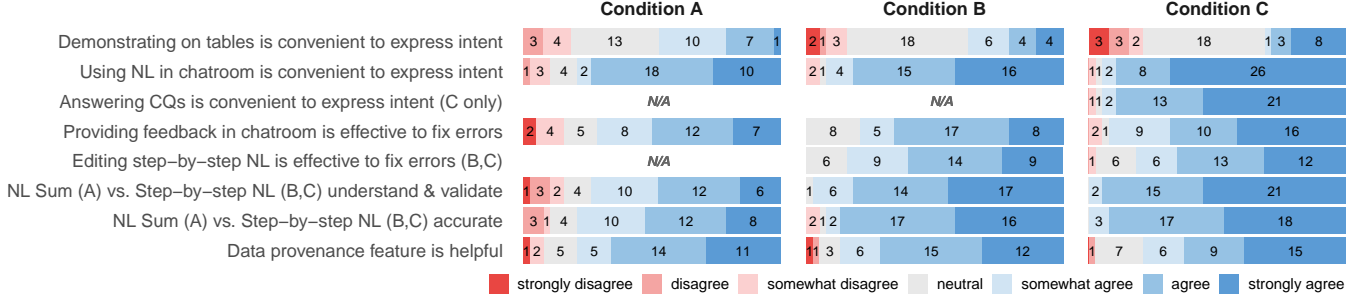


Figure 7: 7-point Likert scale evaluations (1—strongly disagree, 7—strongly agree) of user-perceived convenience, and effectiveness of DANGO's key features across three conditions.

Using NL Prompts in the Chatroom. Compared to demonstrations, significantly more users preferred to express their intent in NL. 74% of participants in Condition A, 82% in Condition B, and 89% in Condition C agreed or strongly agreed that using NL in the chatroom was convenient to express their intent. Participants appreciated the chatroom feature for allowing easy expression of requests in natural language (P16), facilitating straightforward communication (P3), and enabling clear articulation of data cleaning actions (P27). The result suggests that the conversational UI is effective in communicating user intent.

Answering CQs. As shown in Figure 7, 89% of the participants in Condition C agreed or strongly agreed that answering CQs is convenient to express their intent. Participants mentioned that it is helpful to have DANGO first ask clarification questions because it shows its understanding of their intent (P11) and reduces the burden to type more detailed clarification instructions when the output is not desired (P12, P22). This result suggests that proactively asking clarification questions can effectively help users clarify intent and complement NL prompting.

7.2.2 Participants' Perceptions of Different Error-Fixing Features. DANGO supports two features that help users fix their programming errors. First, in all conditions, users can use NL prompts to provide feedback in the chatroom and ask the system to regenerate the script accordingly. In Conditions B and C, users can also directly edit the step-by-step NL explanation.

Providing Feedback in the Chatroom. As shown in Figure 7, 50% of participants in Condition A, 66% in Condition B, and 68% in Condition C agreed or strongly agreed that providing feedback in the chatroom is effective for error correction. However, some participants complained that chatroom is not an efficient way to fix errors, since it requires more clarifications to specify which step (P9), and type out every detail to fix even small errors (P15).

Editing Step-by-step NL Explanations. In Conditions B and C, DANGO allowed users to fix errors by directly editing the step-by-step NL description of the scripts. As shown in Figure 7, 61% in Condition B and 66% in Condition C agreed or strongly agreed that directly editing the step-by-step NL description is effective in fixing errors. Participants found that step-by-step NL explanations were valuable in multiple ways: including clearly indicating where modifications were needed (P9), offering flexibility in program modification while illustrating the problem-solving sequence (P20), and helping verify whether their instructions were correctly interpreted (P16). However, compared to directly providing feedback in the chatroom, we found no statistically significant differences in the mean ratings between these two features.

7.2.3 Participants' Perceptions of Different Program Explanation Features. We experimented with two methods to explain a synthesized script. As described in Section 6.1, in Condition A, DANGO simply generates an NL summary by prompting an LLM. In Conditions B and C, DANGO uses a rule-based method to translate the synthesized script into a step-by-step NL explanation.

NL Summary. As shown in Figure 7, in Condition A, only 47% of participants agreed or strongly agreed that NL summary helps them understand the script and 53% agreed or strongly agreed that NL summary accurately represented program behavior. Some participants found the NL summaries challenging to work with, noting that they were overly verbose (P10), and required effort to parse and verify against original prompts (P18).

Step-by-Step NL Explanation. As shown in Figure 7, significantly more users appreciated the step-by-step NL explanations compared with NL summary. 82% in Condition B and 95% in Condition C agreed or strongly agreed that these explanations helped them understand the scripts. Furthermore, 87% in Condition B and 92% in Condition C agreed or strongly agreed that the step-by-step

NL explanations accurately represented program behavior. Participants found that the step-by-step approach enhanced program comprehension by improving readability for error checking (P4) and making scripts easier to understand and execute (P21). The results suggest that structured step-by-step natural language explanations can effectively help users better validate and understand the scripts' behavior.

7.2.4 Participants' Perceptions of Data Provenance Feature.

The majority of the participants appreciated the data provenance feature that allows them to check the contents of different versions. 66% of participants in Condition A, 71% in Condition B, and 68% in Condition C agreed or strongly agreed that seeing data provenance helps check the contents of different versions of tables. Participants found that the data provenance view helped build confidence by showing correspondences between selected tables (P15), and became more valuable as tasks increased in complexity (P16).

7.2.5 Utility Rates of Different Features. To understand user behavior when using DANGO, we analyzed our system logs and recordings and examined how participants interacted with different features to complete their tasks effectively. Figure 8 renders the temporal distribution of different user events for all task sessions in different conditions. Given the variances in the task completion time of each session, we normalized the occurrence of each user event with respect to the task session length. Notably, participants in Condition C relied less on user-initiated features like demonstrations and step-by-step explanations compared to Conditions A and B. The differences were statistically significant (p -values = $1.2e-02$, and $5.64e-04$, respectively). This indicates that participants in Condition C did not need to spend time on re-prompting and demonstrations. Instead, participants engaged more with multiple-choice clarification questions, which helped them effectively clarify their intent. This improvement aligns with the reduced completion times and improved success rate reported in Section 7.1.

7.3 RQ3: Reasons of DANGO's Effectiveness

We analyzed participants' responses in the post-study survey and identified several key findings that explain why DANGO helped participants better perform data wrangling tasks.

Finding 1: Step-by-step NL explanation helps users understand the program behavior. We found that the step-by-step NL explanation of the program accelerated the process of program comprehension. 82% of participants in Condition B, and 95% of participants in Condition C agreed or strongly agreed that step-by-step NL descriptions helped them understand the synthesis script, as shown in Figure 7. P19 said, "It divided the task into NL steps, easy to understand the steps performed." In contrast, participants in Condition A spent more time comprehending the NL summary of the program. P8 said, "I disliked that the natural language summaries didn't clearly distinguish between partially inaccurate and accurate responses. Their similarity made it time-consuming to read the entire summary and compare different versions."

Finding 2: Direct edits on step-by-step NL statement make correction easier. We observed that the LLM misinterpreted user intent in many cases and generated an incorrect script initially. In Condition A, participants could only provide feedback to fix

incorrect scripts by providing more demonstrations or giving NL feedback in the chatroom. These two kinds of feedback turned out to be not very effective in guiding the LLM to refine the script. Specifically, in Condition A, 57% of participants encountered an incorrect script in their first attempt and spent an average of 4 minutes and 6 seconds to fix the incorrect script. By contrast, in Conditions B and C, participants could directly edit the NL description of an erroneous step of the script. This provided a more precise and fine-grained feedback to the LLM. While 49% of participants in Condition B and 34% of participants in Condition C encountered an incorrect script in their first attempts, they only spent 2 minutes and 41 seconds and 2 minutes and 25 seconds to fix the incorrect script. Participants appreciated that they could easily edit and delete steps in the script (P7) and make corrections while preserving the conversational context (P1).

Finding 3: Clarification questions help users clarify their intent effectively. Participants frequently provided incomplete or ambiguous prompts. In Condition C, participants could directly answer relevant multiple-choice clarification questions (CQs) without needing to articulate their intent in a new prompt. A total of 40 CQs were generated in Condition C. Notably, 62% of participants who answered CQs succeeded on their first attempt. By contrast, in Conditions A and B, only 43% and 51% of participants succeeded in their first attempt. In the post-task survey, 89% of participants agreed or strongly agreed that answering CQs was an effective way to express their intent, as shown in Figure 7. P14 commented, "I appreciate how DANGO asks clarifying questions before generating the program. This approach is better than directly outputting information, as ChatGPT might do, without truly understanding the user's intent." Similarly, P11 stated, "I appreciated the questions posed by the LLM, as they eliminated the need for me to specify all the details in my initial prompt."

7.4 RQ4: Impact of User Expertise

7.4.1 Overall Performance Comparison of Users with Different Expertise. We categorized participants into three groups based on their programming experiences, including novices (<1 year, N=12), intermediate programmers (1-5 years, N=13), and experts (> 5 years, N=12). We compared task completion time across these three groups. Experts completed tasks in an average of 4 minutes and 17 seconds, intermediate programmers in 5 minutes and 14 seconds, and novices in 5 minutes and 32 seconds. These mean differences were not statistically significant (ANOVA test: p -value= $1.33e-01$). The average number of attempts was similar across groups—experts made 1.67 attempts, intermediate programmers 1.74 attempts, and novices 1.67 attempts. These mean differences were also not statistically significant differences (ANOVA test: p -value= $0.92e-01$). This suggests that user expertise had a limited impact on performance. We interpret this as a positive impact of DANGO, which narrows the performance gap between users of different levels of programming expertise.

7.4.2 Task-specific Comparison of Users with Different Expertise. In a detailed analysis of individual tasks, we found that task completion time was generally consistent across user groups for Tasks 1-6, aligning with our overall finding of no significant differences. However, Task 7 emerged as a notable exception. Experts

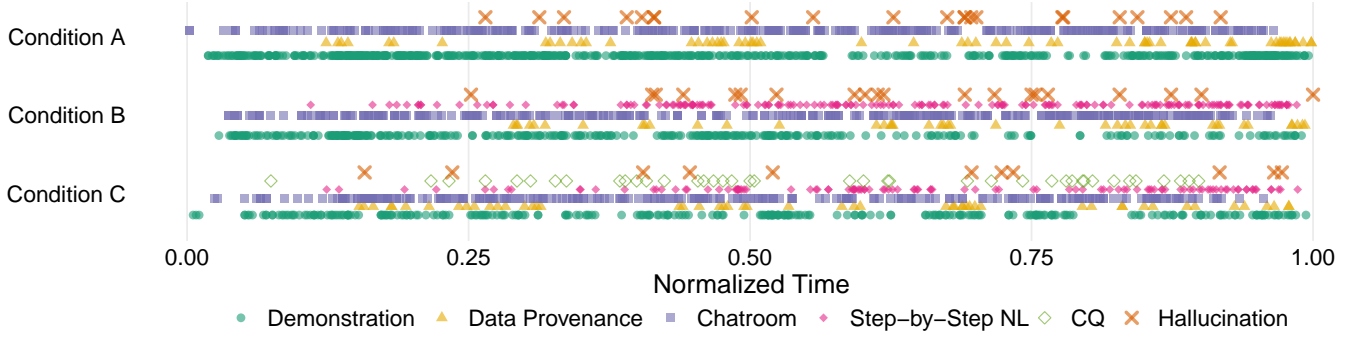


Figure 8: A scatter plot displays events occurring throughout normalized time across all conditions.

completed this task in an average of 2 minutes and 10 seconds, while intermediate programmers and novices took considerably longer time—6 minutes and 54 seconds and 6 minutes and 53 seconds, respectively. This difference was statistically significant (ANOVA test p -value=1.15e-03). One plausible reason is that Task 7 required complex operations such as conditional data transformations and statistical testing. In particular, participants were required to perform a statistical test on their data and then apply an conditional filtering of data records based on the test results. Compared with novices and intermediate programmers, experts had more advantage in knowledge and experience to handle such a complex task. This suggests that while DANGO successfully leveled the playing field for common data wrangling operations, more complex tasks may still benefit from user expertise.

7.5 RQ5: Generalizability of DANGO

To evaluate the generalizability of DANGO, we conducted a case study leveraging 10 data wrangling tasks collected from prior work [22, 55, 56, 68]. Since the original tasks primarily focused on single-table tasks, we extended the benchmark by including another 14 multi-table data wrangling tasks introduced by Kasic et al. [56]. The final benchmark includes a total of 24 tasks.¹ It includes some common data wrangling tasks, such as transposing tables (benchmark ID 24), splitting one column into two columns based on a delimiter (benchmark ID 16), and summarizing tables by calculating average numbers (benchmark ID 21). It also includes more difficult tasks that involve conditional operations. For instance, deleting rows as long as there is a missing value in the row (benchmark ID 23). Since this quantitative study aims to verify that DANGO can solve various data wrangling tasks instead of evaluating its learnability or usability, the first two authors independently solved these 24 tasks using DANGO. They represent expert users who are familiar with the tool and have rich experience in data wrangling. Thus, the results of this study should be interpreted as the performance of DANGO in ideal situations. A task is considered failed if an author cannot solve the task after 10 minutes. Overall, the two authors solved all 24 tasks with an average of 62.67 seconds and 75.90 seconds, respectively. On average, they required 1.54 and 1.33 attempts per task. This result shows that DANGO can be used to solve a wide range of data wrangling tasks.

¹We provide the task descriptions and synthesized data wrangling scripts in the supplemental material.

7.5.1 DSL Statement Distribution in Synthesized Programs.

To understand the generalizability of individual DSL statements that are supported by DANGO, we investigated the number of DSL statements that appeared in the synthesized programs. Specifically, Figure 9 shows how often each DSL statement was synthesized in these 24 tasks. The translation rules for `DROP` and `MERGE` were the most utilized, both were triggered 13 times. The frequent use of `DROP` implies that many data wrangling tasks require removing undesired data, e.g., rows with missing values, columns with sensitive information, etc. Furthermore, the frequent use of `MERGE` implies a common need of combining cleaned data from multiple tables into one single final table.

7.5.2 Effectiveness of Clarification Questions. When solving these 24 tasks, the first author encountered 46 CQs, while the second author encountered 32 CQs. The authors examined the helpfulness of each CQ by checking if (1) it asked relevant and essential information critical to task success, and (2) it helped clarify their intent. The first author found 87% of the questions helpful, while the second author rated 94% as helpful. Regarding user responses to CQs, the first author selected the provided options 74% of the time, while the second author selected the provided options 75% of the time. In the remaining case, they chose “Other (please specify)” to clarify their intent. This implies that the auto-generated options in the CQs are likely to cover user intent in the majority of cases.

8 DISCUSSION

8.1 Design Implications

8.1.1 Aligning the LLM with human intent via interactive clarification questions. Compared to existing data wrangling tools, our tool allows the LLM to proactively ask clarification questions to engage with users and understand their intent. The success of DANGO also echoes Grice’s Maxims of Manner in cooperative principles [15], which emphasize being clear and avoiding ambiguity in communication with users. Besides, we implement multiple-choice questions with potential options for users to select from, rather than relying on open-ended questions. This allows for precise intent clarification while reducing the manual effort of typing down details to articulate user intent. By integrating the Q&A history, our tool improves the accuracy of data wrangling tasks and provides a more cooperative and user-friendly experience.

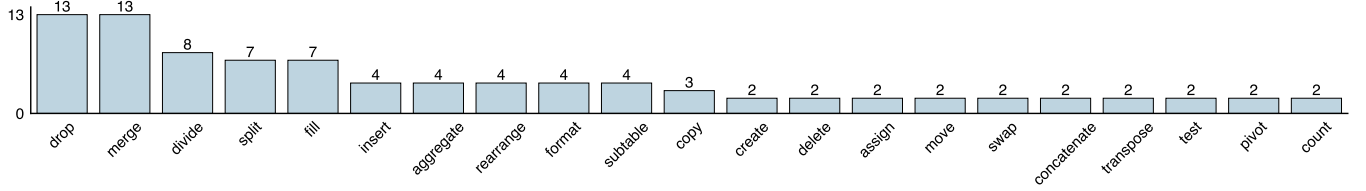


Figure 9: The number of DSL statements that cover different kinds of data-wrangling tasks in the case study.

8.1.2 Scaffolding LLM understanding using step-by-step NL explanations. The synthesized script can be interpreted as a manifesto of the LLM’s understanding of user intent. If the LLM misinterprets user intent, the synthesized script is likely to be wrong. The NL explanation in DANGO serves as an effective bi-directional communication vehicle between the user and the LLM. By explaining the script in NL, users can easily understand how the LLM interprets their intent without the need for programming knowledge. Additionally, the step-by-step structure of the NL explanation provides an effective scaffold for users to indicate the generation errors and provide more targeted and granular feedback than supplementing more demonstrations or entering feedback in the chatbot. The targeted and granular feedback can further help the LLM rectify its misunderstanding and refine the synthesized script.

8.1.3 Help user track data lineage of multiple tables using data provenance. DANGO provides the data provenance feature that enables users to inspect different versions of their data. This is particularly useful for complex data wrangling tasks involving multiple tables. This feature allows users to trace the lineage of tables, revealing how specific data points have been modified throughout the wrangling process. Users can interact with the data provenance view by clicking on a table node, triggering the corresponding table content to appear in the table view panel. This visual correspondence, also known as *brushing* [14], allows users to visualize table interdependencies and build confidence in the synthesized script.

8.2 Limitations

First, our predefined DSL functions may not cover all possible data wrangling scenarios. The DSL does not explicitly include traditional control structures like `for` and `while` loops. Second, to mitigate hallucinations, we only implement a syntax checker to validate the synthesized script, omitting semantic correctness. Third, some participants mentioned the lack of advanced table features, such as the ability to use formulas directly in table cells, which are commonly supported by Microsoft Excel and Google Sheets. However, DANGO currently does not support synthesizing formulas. Finally, our user study is conducted in a controlled lab setting. It does not reflect the complexity of real-world data wrangling scenarios, which may involve more complex operations, richer datasets, and even collaboration between multiple users in a longer period of time.

8.3 Future Work

8.3.1 Challenges and considerations of prompt design. LLMs introduce several design challenges. First, LLMs are prone to hallucination [66]. We acknowledge that some errors or failures in our study stem from hallucinations. We observe that LLMs misuse our

DSL even when our prompt clearly states the DSL grammar and provides detailed usage instructions. We have attempted to mitigate this by building a syntax checker to give feedback to the LLM. However, this increases synthesis time, and hallucination still exists. Second, we invested considerable effort and time to balance the level of detail and the number of few-shot examples in the prompt. We found too few examples might increase the error rate and elicit overgeneralized answers. Conversely, too many specific examples might reduce an LLM’s inference ability. For instance, we observed that LLMs tend to rely more on pattern matching of the provided examples rather than truly understanding the underlying tasks. Future work should consider these prompt engineering challenges that have recently been coined as the **gulf of envision** [95].

8.3.2 Handling ambiguous user intent. We found many errors stem from users’ ambiguous NL inputs and demonstrations. These errors cannot be solely attributed to the model’s hallucination problems or prompt designs. When given ambiguous input, the LLM could interpret it differently. For example, if a user requests to “sort the elements in the table,” several questions arise: Should the sorting be ascending or descending? Should LLM sort it based on the name alphabetically or the ID values? Our study shows that simple clarification questions could greatly mitigate the effect of ambiguous intent. However, current LLM systems [8, 13, 34, 98] rarely check with users or help them clarify intent, instead synthesizing output in one shot. Future work could develop more interactive approaches to disambiguate users’ intent, rather than expecting the LLM to magically “read minds” when given ambiguous input.

8.3.3 Mitigating the abstract matching problem in NL interfaces. We analyzed the recordings to understand why some users failed or spent more time to complete tasks. We found that while these users typically had well-formed intent, they often struggled to articulate them with sufficient precision and detail for the LLM to accurately map their intent to desired solutions. Such a challenge is common in NL interfaces, and it can be generally seen as the **gulf of execution** [49], or more precisely the **abstraction matching problem** [72]: *selecting words that align with the appropriate level of abstraction required by the system to choose correct actions*. However, our clarification question is not currently designed to bridge the abstraction gap. A potential solution is to build a bidirectional channel that maps the NL interface to the code. This could create visual correspondence, allowing users to link specific NL to code. Such a feature could also serve as a valuable learning tool for end-users.

8.3.4 Visualizing intermediate states for fine-grained examination. Some users have expressed concerns about the visibility of intermediate states in a data wrangling script. They noted that some

intermediate states are not visible when executing the script. For example, when synthesizing a script to delete a column based on the p-value computed by a statistical test, users may want to check the p-value to make sure the t-test is performed correctly. There have been some existing efforts to enable users to inspect the intermediate states of a synthesized script [23, 96, 114]. A design challenge in the domain of data wrangling is that some intermediate states may involve a large volume of data, which may be overwhelming to visualize to users. Furthermore, given the large volume of data, helping users recognize the intermediate state change before and after a data transformation operation could be challenging. These challenges are worthwhile to investigate in the future.

8.3.5 Supporting multimodal interaction paradigms. DANGO currently supports demonstrations and NL interaction. We believe recent advancements in multimodal LLMs [47, 69, 70] will further expand the potential design space for data wrangling tasks. For instance, future systems could enable users to sketch and circle data to clarify references such as “that column” or “this row.” They could also support hand gestures to zoom, filter, and navigate multi-table tasks that are tedious with NL prompts alone. However, implementing such systems poses several challenges. First, it is difficult to “translate” different intent from various modalities into a unified language. Second, a multimodal system should also be able to handle ambiguous inputs and generate conclusions with varying levels of certainty [51]. Future work should consider these challenges, also known as *fusion* [51] issues, in the era of multimodal LLMs.

9 CONCLUSION

This paper introduces DANGO, a mixed-initiative LLM system for data wrangling. Compared with previous work, DANGO allows the LLM to proactively ask clarification questions to resolve ambiguities in user intent. It also provides multiple feedback mechanisms to help the user understand and refine the synthesized script. A within-subjects study ($n = 38$) demonstrates that DANGO’s features significantly improved data wrangling efficiency. Furthermore, a case study with 24 additional tasks demonstrates the generalizability of DANGO’s effectiveness on different kinds of tasks.

References

- [1] 2024. FastAPI. <https://fastapi.tiangolo.com/>. Accessed: 2024.
- [2] 2024. Handsontable. <https://handsontable.com/>. Accessed: 2024.
- [3] 2024. Python Built-in Functions. <https://docs.python.org/3/>. Accessed: 2024.
- [4] 2024. React Flow. <https://reactflow.dev/>. Accessed: 2024.
- [5] 2024. Tailwind CSS. <https://tailwindcss.com/>. Accessed: 2024.
- [6] 2024. Vite. <https://vitejs.dev/>. Accessed: 2024.
- [7] 2024. Zustand. <https://zustand-demo.pmnd.rs/>. Accessed: 2024.
- [8] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
- [9] Christopher Ahlberg and Ben Shneiderman. 1994. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 313–317.
- [10] Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W Bruce Croft. 2019. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd international acm sigir conference on research and development in information retrieval*. 475–484.
- [11] James E Allen, Curry I Guinn, and Eric Horvitz. 1999. Mixed-initiative interaction. *IEEE Intelligent Systems and their Applications* 14, 5 (1999), 14–23.
- [12] Amazon. 2023. Introducing the Next Generation of Alexa—Larger Language Models for Enhanced Conversational AI. <https://developer.amazon.com/en-US/blogs/alexa/alexa-skills-kit/2023/09/alexa-llm-fall-devices-services-sep-2023>.
- [13] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403* (2023).
- [14] Richard A Becker and William S Cleveland. 1987. Brushing scatterplots. *Technometrics* 29, 2 (1987), 127–142.
- [15] Niels Ole Bernsen, Hans Dybkjær, and Laila Dybkjær. 1996. Cooperativity in human-machine and human-human spoken dialogue. *Discourse processes* 21, 2 (1996), 213–236.
- [16] Fabian Biester, Mohamed Abdelaal, and Daniel Del Gaudio. 2024. LLMClean: Context-Aware Tabular Data Cleaning via LLM-Generated OFDs. *arXiv preprint arXiv:2404.18681* (2024).
- [17] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2022. The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large. In *Proceedings of the 44th International Conference on Software Engineering*. 2091–2103.
- [18] Alan F Blackwell. 2001. SWYN: A visual representation for regular expressions. In *Your wish is my command*. Elsevier, 245–XIII.
- [19] Christine L Borgman. 1986. Why are online catalogs hard to use? Lessons learned from information-retrieval studies. *Journal of the American society for information science* 37, 6 (1986), 387–400.
- [20] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
- [21] Sarah E Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scraping distributed hierarchical web data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 963–975.
- [22] Ran Chen, Di Weng, Yanwei Huang, Xinhuan Shu, Jiayi Zhou, Guodao Sun, and Yingcai Wu. 2022. Rigel: Transforming tabular data by declarative mapping. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2022), 128–138.
- [23] Weihao Chen, Xiaoyu Liu, Jiacheng Zhang, Ian Iong Lam, Zhicheng Huang, Rui Dong, Xinyu Wang, and Tianyi Zhang. 2023. MIWA: Mixed-Initiative Web Automation for Better User Control and Confidence. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–15.
- [24] Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, and Jianye Hao. 2024. SheetAgent: A Generalist Agent for Spreadsheet Reasoning and Manipulation via Large Language Models. *arXiv preprint arXiv:2403.03636* (2024).
- [25] Alan Y Cheng, Meng Guo, Melissa Ran, Arpit Ranasingha, Arjun Sharma, Anthony Xie, Khuyen N Le, Bala Vinaithirathan, Shihe Luan, David Thomas Henry Wright, et al. 2024. Scientific and Fantastical: Creating Immersive, Culturally Relevant Learning Experiences with Augmented Reality and Large Language Models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–23.
- [26] John Joon Young Chung, Wooseok Kim, Kang Min Yoo, Hwaran Lee, Eytan Adar, and Minsuk Chang. 2022. TaleBrush: Sketching stories with generative pretrained language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–19.
- [27] Hai Dang, Lukas Mecke, Florian Lehmann, Sven Goller, and Daniel Buschek. 2022. How to prompt? Opportunities and challenges of zero- and few-shot learning for human-AI interaction in creative applications of generative models. *arXiv preprint arXiv:2209.01390* (2022).
- [28] Valdemar Danry, Pat Pataranutaporn, Yaoli Mao, and Pattie Maes. 2023. Don’t just tell me, ask me: Ai systems that intelligently frame explanations as questions improve human logical discernment accuracy over causal ai explanations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [29] Tamraparni Dasu and Theodore Johnson. 2003. *Exploratory data mining and data cleaning*. John Wiley & Sons.
- [30] Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–12.
- [31] Zachary Eberhart and Collin McMillan. 2022. Generating clarifying questions for query refinement in source code search. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 140–151.
- [32] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. 2017. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices* 52, 6 (2017), 422–436.
- [33] Ronald Aylmer Fisher, Ronald Aylmer Fisher, Statistiker Genetiker, Ronald Aylmer Fisher, Statistician Genetician, Great Britain, Ronald Aylmer Fisher, and Statisticien Généticien. 1966. *The design of experiments*. Vol. 21. Springer.
- [34] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines* 30 (2020), 681–694.
- [35] Jie Gao, Simret Araya Gebreegziabher, Kenny Tsu Wei Choo, Toby Jia-Jun Li, Simon Tangi Perrault, and Thomas W Malone. 2024. A Taxonomy for Human-LLM Interaction Modes: An Initial Exploration. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–11.

- [36] Ken Gu, Madeleine Grunde-McLaughlin, Andrew McNutt, Jeffrey Heer, and Tim Althoff. 2024. How do data analysts respond to ai assistance? a wizard-of-oz study. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–22.
- [37] Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M Drucker. 2024. How Do Analysts Understand and Verify AI-Assisted Data Analyses?. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–22.
- [38] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices* 46, 1 (2011), 317–330.
- [39] Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.
- [40] Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H Muggleton, Ute Schmid, and Benjamin Zorn. 2015. Inductive programming meets the real world. *Commun. ACM* 58, 11 (2015), 90–99.
- [41] Sumit Gulwani, Vijay Anand Korthikanti, and Ashish Tiwari. 2011. Synthesizing geometry constructions. *ACM SIGPLAN Notices* 46, 6 (2011), 50–61.
- [42] 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.
- [43] William R Harris and Sumit Gulwani. 2011. Spreadsheet table transformations from examples. *ACM SIGPLAN Notices* 46, 6 (2011), 317–328.
- [44] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Advances in psychology*. Vol. 52. Elsevier, 139–183.
- [45] Yeye He, Kris Ganjam, Kulkjn Lee, Yue Wang, Vivek Narasayya, Surajit Chaudhuri, Xu Chu, and Yudian Zheng. 2018. Transform-data-by-example (tde) extensible data transformation in excel. In *Proceedings of the 2018 International Conference on Management of Data*. 1785–1788.
- [46] Chris Hess and Sarah E Chasins. 2022. Informing housing policy through web automation: Lessons for designing programming tools for domain experts. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–9.
- [47] Wenbo Hu, Yifan Xu, Yi Li, Weiye Li, Zeyuan Chen, and Zhuowen Tu. 2024. Bliva: A simple multimodal llm for better handling of text-rich visual questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 2256–2264.
- [48] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).
- [49] Edwin L Hutchins, James D Hollan, and Donald A Norman. 1985. Direct manipulation interfaces. *Human-computer interaction* 1, 4 (1985), 311–338.
- [50] David F Huynh, Robert C Miller, and David R Karger. 2007. Potluck: Data mash-up tool for casual users. In *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007+ ASWC 2007, Busan, Korea, November 11–15, 2007. Proceedings*. Springer, 239–252.
- [51] Alejandro Jaimes and Nicu Sebe. 2007. Multimodal human-computer interaction: A survey. *Computer vision and image understanding* 108, 1-2 (2007), 116–134.
- [52] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.
- [53] Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology* 33, 7 (2024), 1–30.
- [54] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 683–698.
- [55] 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.
- [56] Stephen Kascia, Charles Berret, and Tamara Munzner. 2020. Table scraps: an actionable framework for multi-table data wrangling from an artifact study of computational journalism. *IEEE Transactions on visualization and computer graphics* 27, 2 (2020), 957–966.
- [57] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary 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. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–20.
- [58] 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.
- [59] Amy J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 199–206.
- [60] Tessa Lau. 2009. Why Programming by Demonstration Systems Fail: Lessons Learned for Usable AI. *AI Magazine* 30, 4 (2009), 65–67.
- [61] Tessa Lau et al. 2008. Why PBD systems fail: Lessons learned for usable AI. In *CHI 2008 Workshop on Usable AI*. 65–67.
- [62] Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. 2001. Learning repetitive text-editing procedures with SMARTedit. In *Your wish is my command*. Elsevier, 209–21.
- [63] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. *Research methods in human-computer interaction*. Morgan Kaufmann.
- [64] Vu Le and Sumit Gulwani. 2014. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 542–553.
- [65] Tak Yeon Lee, Casey Dugan, and Benjamin B Bederson. 2017. Towards understanding human mistakes of programming by example: an online user study. In *Proceedings of the 22Nd International Conference on Intelligent User Interfaces*. 257–261.
- [66] Florian Leiter, Sven Eckhardt, Valentin Leuthe, Merlin Knaeble, Alexander Maedche, Gerhard Schwabe, and Ali Sunyaev. 2024. Hill: A hallucination identifier for large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–13.
- [67] Brenna Li, Amy Wang, Patricia Strachan, Julie Anne Séguin, Sami Lachgar, Karyn C Schroeder, Mathias S Fleck, Renee Wong, Alan Karthikesalingam, Vivek Natarajan, et al. 2024. Conversational AI in health: Design considerations from a Wizard-of-Oz dermatology case study with users, clinicians and a medical LLM. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–10.
- [68] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and ZHAO-XIANG ZHANG. 2024. SheetCopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems* 36 (2024).
- [69] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*. PMLR, 12888–12900.
- [70] Yaowei Li, Ruijie Quan, Linchao Zhu, and Yi Yang. 2023. Efficient multimodal fusion via interactive prompting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2604–2613.
- [71] Anna Lieb and Toshali Goel. 2024. Student Interaction with NewtBot: An LLM-as-tutor Chatbot for Secondary Physics Education. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–8.
- [72] 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.
- [73] Ziyi Liu, Zhengzhe Zhu, Lijun Zhu, Enze Jiang, Xiyun Hu, Kylie A Peppler, and Karthik Ramani. 2024. ClassMeta: Designing Interactive Virtual Classmate to Promote VR Classroom Participation. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–17.
- [74] 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.
- [75] Microsoft. 2023. Introducing Microsoft 365 Copilot—Your Copilot for Work. <https://blogs.microsoft.com/blog/2023/03/16/introducing-microsoft-365-copilot-your-copilot-for-work/>.
- [76] Robert C Miller and Brad A Myers. 2001. Outlier finding: Focusing user attention on possible errors. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*. 81–90.
- [77] Piotr Mirowski, Kory W Mathewson, Jaylen Pittman, and Richard Evans. 2023. Co-writing screenplays and theatre scripts with language models: Evaluation by industry professionals. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–34.
- [78] Dan H Mo and Ian H Witten. 1992. Learning text editing tasks from examples: a procedural approach. *Behaviour & Information Technology* 11, 1 (1992), 32–45.
- [79] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading between the lines: Modeling user behavior and costs in AI-assisted programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–16.
- [80] 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*. 1–15.
- [81] Brad A Myers and Richard McDaniel. 2001. Demonstrational interfaces: sometimes you need a little intelligence, sometimes you need a lot. In *Your Wish is My Command*. Elsevier, 45–III.
- [82] Jakob Nielsen. 2023. *AI: First New UI Paradigm in 60 Years*. <https://www.nngroup.com/articles/ai-paradigm/> Nielsen Norman Group.
- [83] Robert P Nix. 1985. Editing by example. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7, 4 (1985), 600–621.

- [84] Jack E Olson. 2003. *Data quality: the accuracy dimension*. Elsevier.
- [85] Martin T Orne. 2017. On the social psychology of the psychological experiment: With particular reference to demand characteristics and their implications. In *Sociological methods*. Routledge, 279–299.
- [86] Gil Press. 2016. Cleaning big data: Most time-consuming, least enjoyable data science task, survey says. *Forbes* (Mar 2016). <https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/>.
- [87] Danrui Qi and Jiannan Wang. 2024. CleanAgent: Automating Data Standardization with LLM-based Agents. *arXiv preprint arXiv:2403.08291* (2024).
- [88] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. 381–390.
- [89] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.
- [90] Boaz Ronen, Michael A Palley, and Henry C Lucas Jr. 1989. Spreadsheet analysis and design. *Commun. ACM* 32, 1 (1989), 84–93.
- [91] Christopher Scaffidi, Brad Myers, and Mary Shaw. 2009. Intelligently creating and recommending reusable reformatting rules. In *Proceedings of the 14th international conference on Intelligent user interfaces*. 297–306.
- [92] 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.
- [93] Hua Shen, Tiffany Kneare, Reshmi Ghosh, Kenan Alkiek, Kundan Krishna, Yachuan Liu, Ziqiao Ma, Savvas Petridis, Yi-Hao Peng, Li Qiwei, et al. 2024. Towards Bidirectional Human-AI Alignment: A Systematic Review for Clarifications, Framework, and Future Directions. *arXiv preprint arXiv:2406.09264* (2024).
- [94] Sruti Srinivasa Ragavan, Zhitao Hou, Yun Wang, Andrew D Gordon, Haidong Zhang, and Dongmei Zhang. 2022. Gridbook: Natural language formulas for the spreadsheet grid. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*. 345–368.
- [95] Hari Subramonyam, Roy Pea, Christopher Pondoc, Maneesh Agrawala, and Colleen Seifert. 2024. Bridging the Gulf of Envisioning: Cognitive Challenges in Prompt Based Interactions with LLMs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–19.
- [96] Yuan Tian, Jonathan K Kummerfeld, Toby Jia-Jun Li, and Tianyi Zhang. 2024. SQLucid: Grounding Natural Language Database Queries with Interactive Explanations. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–20.
- [97] Bernadette Tix and Kim Binsted. 2024. Better Results Through Ambiguity Resolution: Large Language Models that Ask Clarifying Questions. In *International Conference on Human-Computer Interaction*. Springer, 72–87.
- [98] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).
- [99] Rattapoom Tuchinda, Pedro Szekely, and Craig A Knoblock. 2008. Building mashups by example. In *Proceedings of the 13th international conference on Intelligent user interfaces*. 139–148.
- [100] Mojtaba Vaismoradi, Hannele Turunen, and Terese Bondas. 2013. Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. *Nursing & health sciences* 15, 3 (2013), 398–405.
- [101] Priyan Vaithilingam, Elena L Glassman, Jeevana Priya Inala, and Chenglong Wang. 2024. DynaVis: Dynamically Synthesized UI Widgets for Visualization Editing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–17.
- [102] Bryan Wang, Yuliang Li, Zhaoyang Lv, Haijun Xia, Yan Xu, and Raj Sodhi. 2024. LAVE: LLM-Powered Agent Assistance and Language Augmentation for Video Editing. In *Proceedings of the 29th International Conference on Intelligent User Interfaces*. 699–714.
- [103] Chenglong Wang, John Thompson, and Bongshin Lee. 2023. Data Formulator: AI-powered Concept-driven Visualization Authoring. *IEEE Transactions on Visualization and Computer Graphics* (2023).
- [104] Jian Wang and Wenjie Li. 2021. Template-guided clarifying question generation for web search clarification. In *Proceedings of the 30th ACM international conference on information & knowledge management*. 3468–3472.
- [105] Ian H Witten and Dan Mo. 1993. TELS: Learning text editing tasks from examples. In *Watch what I do: programming by demonstration*. 183–203.
- [106] Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. WaitGPT: Monitoring and Steering Conversational LLM Agent in Data Analysis with On-the-Fly Code Visualization. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–14.
- [107] Navid Yaghmazadeh, Xinyu Wang, and Isil Dillig. 2018. Automated migration of hierarchical data to relational tables using programming-by-example. *Proceedings of the VLDB Endowment* 11, 5 (2018), 580–593.
- [108] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).
- [109] Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. 2022. Wordcraft: story writing with large language models. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*. 841–852.
- [110] 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.
- [111] Lei Zhang, Ashutosh Agrawal, Steve Oney, and Anhong Guo. 2023. Vrgit: A version control system for collaborative content creation in virtual reality. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.
- [112] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. 2020. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 627–648.
- [113] Xuanhe Zhou, Xinyang Zhao, and Guoliang Li. 2024. LLM-Enhanced Data Management. *arXiv preprint arXiv:2402.02643* (2024).
- [114] Zhanhui Zhou, Man To Tang, Qiping Pan, Shangyin Tan, Xinyu Wang, and Tianyi Zhang. 2022. Intent: Interactive tensor transformation synthesis. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–16.
- [115] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. 2018. Version control system: A review. *Procedia Computer Science* 135 (2018), 408–415.
- [116] Jie Zou, Mohammad Aliannejadi, Evangelos Kanoulas, Maria Soledad Pera, and Yiqun Liu. 2023. Users meet clarifying questions: Toward a better understanding of user interactions for search clarification. *ACM Transactions on Information Systems* 41, 1 (2023), 1–25.

A Domain Specific Language

Table-level Functions
1. create_table(row_number, column_number): Returns an empty table with the specified number of rows and columns.
2. delete_table(table_name): Deletes a table from the database.
3. pivot_table(table, index, columns, values, aggfunc): Reshapes the table so that each unique value in columns becomes a separate column, with index values as row headers, and the corresponding values filled in their respective cells.
4. merge(table_a, table_b, how='outer', on=None): Merges two tables based on a common column.
5. subtable(table, labels, axis): Extracts a subtable from a DataFrame based on specified rows or columns.
6. transpose(table): Transposes the given table.
Column/Row-level Functions
1. insert(table, index, index_name, axis): Inserts an empty row or column at the specified index in the table. Other rows or columns will be moved down or to the right.
2. drop(table, label, axis): Drops one or more rows or columns from the table.
3. assign(table, start_row_index, end_row_index, start_column_index, end_column_index, values): Assigns fixed constant values to specific cells in the table.
4. move(origin_table, origin_index, target_table, target_index, axis): Moves a row or column from the origin table to the target table.
5. copy(origin_table, origin_label, target_table, target_label, axis): Copies a row or column from the origin table to the target table at the specified label.
6. swap(table_a, label_a, table_b, label_b, axis): Swaps rows or columns between two tables.
7. rearrange(table, by_values=None, by_array=None, axis): Rearranges the rows or columns of the table based on the specified order.
8. divide(table, by, axis): Divides the table by the specified row or column, returning a list of tables.
9. fill(table, method, labels, axis): Fills missing values in the table using the specified method.
Summarization Functions
1. aggregate(table, functions, axis): Aggregates the table using a specified function.
2. test(table_a, label_a, table_b, label_b, strategy, axis): Compares two labels using the specified statistical test and returns a tuple (statistic, p_value).
3. count(table, label, value, axis): Counts the occurrences of a specified value within a given column or row in a DataFrame, then stores the result in a new DataFrame.
String Operation Functions
1. concatenate(table, label_a, label_b, glue, new_label, axis): Concatenates two rows or columns using a string as glue and appends the merged row or column to the table.
2. split(table, label, delimiter, new_label_list, axis): Separates rows or columns based on a string delimiter within the values.
3. format(table, label, pattern, replace_with, axis): Formats the values in a row or column based on the specified pattern and "replace_with" using re.sub().

Table 3: DSL of DANGO.

B Prompt Designs

CONTEXT
You are a professional data scientist. A user has made some changes in the CSV files. Your task is to understand the user intent regarding how they want to clean the data.
OBJECTIVE
Ask clarification questions to understand the user intent.
GUIDELINES
1. Infer their intent through the table diff and user instruction. Do not infer beyond the information provided in the input. 2. Avoid directly asking, "What is your intent?" Instead, ask questions related to the changes made in the table and the instructions given. 3. If the user intent is clear, you can output the intent summary.
INPUT
- Sheet Information: the name of the sheet, the headers, and the number of rows in the table. - Table Diff: the changes made to the table. - User Instruction: the user's instruction that indicates the changes they want to make.
OUTPUT
Your output must be in JSON list form. If you need more information, output a question to ask the user: { "type": "question", "summary": "<summary>", "question": "<question>", "choices": ["<choice_1>", "<choice_2>", ..., "<choice_n>", "other"] } If the intent is clear enough, output a summary of the user intent: { "type": "finish", "summary": "<summary>" }
EXAMPLES
<SOME EXAMPLES>

Table 4: The prompt of intent summarization & CQ generation.

CONTEXT
You are a professional data scientist. You have already asked some clarification questions and the user has replied. Now, you might want to ask additional questions to gain a deeper understanding of their intent.
OBJECTIVE
<SAME AS TABLE 4>
GUIDELINES
<SAME AS TABLE 4>
INPUT
<ul style="list-style-type: none">- Sheet Information: the name of the sheet, the headers, and the number of rows in the table.- Table Diff: the changes made to the table, including the cells that have been modified.- User Instruction: the user’s instruction that indicates the changes they want to make.- Chat History: The history of chat between the assistant and the user.
OUTPUT
<SAME AS TABLE 4>
EXAMPLES
<SOME EXAMPLES>

Table 5: The follow-up prompt of intent summarization & CQ generation.

CONTEXT
You are a professional data scientist. Your task is to generate a step-by-step plan to clean the data based on the user intent and the sheet information.
OBJECTIVE
Generate a step-by-step plan to clean the data based on the user intent and the sheet information.
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
<ol style="list-style-type: none">1. You should specify the DSL function after the description. Do not add or invent new functions.2. You should point out the arguments for each function based on the given description, please refer to the column and row information in the sheet information.3. Please attention that the row index starts from 0, which is the header row. The column index starts from 1.
INPUT
<ul style="list-style-type: none">- Sheet Information: the name of the sheet, the headers, and the number of rows in the table.- User Intent: the user intent.
OUTPUT
<ul style="list-style-type: none">- Step-by-step plan: a JSON list.<ul style="list-style-type: none">- Each step should only include the function name and its description.- If there are multiple steps, list them in the order they should be executed.
EXAMPLES
<SOME EXAMPLES>

Table 6: The prompt for step-by-step planning.

CONTEXT
You are a professional data scientist. Your task is to generate a step-by-step plan to clean the data based on the user intent and an error message from the last step-by-step plan.
OBJECTIVE
<SAME AS TABLE 6>
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
<SAME AS TABLE 6>
INPUT
- Sheet Information: the name of the sheet, the headers, and the number of rows in the table. - User Intent: the user intent. - Last step-by-step plan: A JSON list. Each step should include the function name and its description. - Error Message: the error message from the last generation.
OUTPUT
<SAME AS TABLE 6>
EXAMPLES
<SOME EXAMPLES>

Table 7: The prompt for step-by-step planning with error messages.

CONTEXT
You are a professional DSL (Domain Specific Language) generator. You will be given a step-by-step description of a data cleaning plan. You need to follow the description and create a DSL script to help user clean and manipulate the data.
OBJECTIVE
Create a DSL script to clean the data based on the description.
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
1. You should only use the DSL functions provided in the DSL Grammar. Do not add or invent new functions. 2. For every step, you need to find the best function from the DSL Grammar to perform the described action. 3. Table names should end with ".csv" to indicate that they are CSV files. 4. If you use an integer for a row index, it should be 0-based. If you use a string for a row index, it should start from "1". 5. For None values, you can use "null" in the output. 6. This DSL script is not the final program. Please use the table names instead of real pandas DataFrames in the arguments.
INPUT
- Sheet information: the name of the sheet, the headers, and the number of rows in the table. - Step-by-step Plan: a detailed description of the process.
OUTPUT
Your output should be in a JSON form. The JSON should contain two parts: - "required_tables": A list of table names that are required to perform the cleaning process. - "program": A list of objects that represent the functions to be applied to the tables. Each object of the list should contain the function name and its arguments. If a function needs to be applied to special cells, you can add a "condition" parameter to the object. Do not add any other characters to the output.
EXAMPLES
<SOME EXAMPLES>

Table 8: The prompt for generating a DSL script.

CONTEXT
You are a professional DSL (Domain Specific Language) generator. You will be given a step-by-step description of a data cleaning plan, and an error message from the last generation. You need to follow the description and create a DSL script to help the user clean and manipulate the data.
OBJECTIVE
<SAME AS TABLE 8>
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
<SAME AS TABLE 8>
INPUT
- Sheet information: the name of the sheet, the headers, and the number of rows in the table. - Step-by-step Plan: a detailed description of the process. - Error Message: The error message from the last generation.
OUTPUT
<SAME AS TABLE 8>
EXAMPLES
<SOME EXAMPLES>

Table 9: The prompt for generating a DSL script with error messages.

CONTEXT
You are a professional DSL (Domain Specific Language) expert. You will be given a required tables list, a DSL functions list in a JSON list format, and user intent. You need to create a Python code snippet that executes the given DSL function.
OBJECTIVE
Create a Python code snippet that executes the given DSL functions list.
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
1. All output should include the save_table function. 2. When creating a new table using the save_table function, ensure the table is formatted as "{name}_v{version}.csv". Please start the version from 0. 3. When using merge_table function, here is the naming convention for the tables: - merge_table function: "merged_v0.csv" 4. The row indexes are strings that start from "1" and should be enclosed in double-quotes. 5. Notice the number of output arguments for each function and assign them accordingly.
INPUT
- Required Tables: A list of table names that are required to perform the DSL functions. - DSL Program: A JSON list containing the DSL functions to be executed. - User Intent: A natural language description of the user intent.
OUTPUT
Your output should be between ``` tags and contain the Python code snippet that executes the given DSL function.
EXAMPLES
<SOME EXAMPLES>

Table 10: The prompt for executing a DSL script.

CONTEXT
You are a professional DSL (Domain Specific Language) generator. You will be given an instruction to create a DSL and information including a previous version DSL list and a DSL grammar.
OBJECTIVE
Create the DSL script as the instructions specified.
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
1. You should only use the DSL functions provided in the DSL Grammar. Do not add or invent new functions. 2. Table names should end with ".csv" to indicate that they are CSV files. 3. This DSL script is not the final program. Please use the table names instead of real pandas DataFrames in the arguments.
INPUT
- Previous generated DSL - New Instruction
OUTPUT
Your output should be in a JSON object. Each object should contain the function name and its arguments. If a function needs to be applied to specific cells, you can add a "condition" parameter to the object.
EXAMPLES
<SOME EXAMPLES>

Table 11: The prompt for regenerating a DSL script based on user feedback.

CONTEXT
You are a professional DSL (Domain Specific Language) generator. You will be given instructions on how to change a DSL from the previous version to the new DSL.
OBJECTIVE
Change the DSL script as the instructions specified.
DSL GRAMMAR
<DSL_GRAMMAR>
GUIDELINES
1. You should only use the DSL functions provided in the DSL Grammar. Do not add or invent new functions. 2. Table names should end with ".csv" to indicate that they are CSV files. 3. This DSL script is not the final program. Please use the table names instead of real pandas DataFrames in the arguments.
INPUT
- Previous generated DSL - New Instruction
OUTPUT
Your output should be in a JSON object. Each object should contain the function name and its arguments. If a function needs to be applied to specific cells, you can add a "condition" parameter to the object.
EXAMPLES
<SOME EXAMPLES>

Table 12: The prompt for editing a DSL script based on user feedback.

CONTEXT
You are a professional data scientist. You will be given a DSL script that is used to clean and manipulate the data and a previous user intent.
OBJECTIVE
Update the user intent based on the new DSL script.
INPUT
- New DSL Script: a list of objects that represent the functions to be applied to the tables.
OUTPUT
New user intent based on the new DSL script.
EXAMPLES
<SOME EXAMPLES>

Table 13: The prompt for updating the user intent for program refinement.

CONTEXT
You are a professional Data Scientist. You will be given a DSL script that is used to clean and manipulate the data.
OBJECTIVE
Provide a natural language description for the DSL script.
DSL GRAMMAR
<DSL_GRAMMAR>
INPUT
- DSL Script: a list of objects that represent the functions to be applied to the tables.
OUTPUT
- NL Explanation: a natural language description of the DSL script.
EXAMPLES
<SOME EXAMPLES>

Table 14: The prompt to generate a NL summary of a synthesized DSL in Condition A of the user study.

C Translation Rules

Statements	Translation rules
create_table(X, Y)	"Create a blank table with" + X + "rows and" + Y + "columns"
delete_table(X)	"Delete the table" + X
insert(table, X, Y, axis)	"Insert a column at position" + X + "in the given table(s)"
drop(table, X, axis)	"Drop the column" + X + "in the given table(s)"
assign(table, row, col, value, values)	"Assign the values" + json.dumps(values) + "in the given table(s)"
move(table, X, Y, axis)	"Move the column" + X + "to column" + Y + "in the given table(s)"
copy(table, X, Y, axis)	"Copy the column" + X + "to column" + Y + "in the given table(s)"
swap(table, X, Y, axis)	"Swap the column" + X + "and the column" + Y + "in the given table(s)"
merge(table_a, X, Y)	"Merge the given table(s) with the table" + X + "based on the values in the column" + Y
concatenate(table, X, Y, Z, axis)	"Concatenate the columns" + X + "and" + Y + "in the given table(s) with the glue" + Z
split(table, X, Y, axis)	"Split the values in the column" + X + "in the given table(s) with the delimiter" + Y
transpose(table)	"Transpose the given table(s)"
aggregate(table, X)	"Aggregate the given table(s) with the functions" + X
test(A, B, C, D, E, axis)	"Test the columns" + B + "in table" + A + "and" + D + "in table" + C + "using the" + E
rearrange(table, X, axis)	"Rearrange the columns in the given table(s) based on the values in the column" + X
format(table, W, X, Y, axis)	"Format the values in the column" + W + "in the given table(s) with the pattern" + X + "and replace them with" + Y
divide(table, X, axis)	"Divide the given table(s) by the values in the column" + X
fill(table, X, Y)	"Fill the missing values in the column" + X + "in the given table(s) with the method" + Y
pivot_table(table, W, X, Y, Z)	"Create a pivot table in the given table(s) with the index" + W + ", columns" + X + ", values" + Y + ", and the aggregation function" + Z
subtable(table, X, axis)	"Extract a subtable from the given table(s) based on the columns" + X
count(table, X, Y, axis)	"Count the occurrences of the value" + Y + "in the column" + X + "in the given table(s)"

Table 15: Translation rules for DSL to Natural Language.

D User Study Material

D.1 Participant Demographics

ID	Gender	Programming Experience	Education	Department	Expertise
P1	Male	1-5 years	Undergraduate	Computer Science	Novice
P2	Male	5+ years	PhD	Computer Science	Expert
P3	Female	1-5 years	PhD	Statistics	Novice
P4	Male	5+ years	Master	Computer Science	Expert
P5	Male	5+ years	Master	Computer Science	Expert
P6	Female	1-5 years	Undergraduate	Computer Science	Novice
P7	Male	1-5 years	Undergraduate	Computer Science	Novice
P8	Male	1-5 years	Undergraduate	Computer Science	Novice
P9	Female	1-5 years	PhD	Industrial Engineering	Novice
P10	Female	< 1 year	PhD	Hospitality	End-user
P11	Male	5+ years	PhD	Computer Science	Expert
P12	Male	5+ years	PhD	Computer Science	Expert
P13	Female	1-5 years	PhD	Computer Science	Novice
P14	Male	1-5 years	Undergraduate	Computer Science	Novice
P15	Male	1-5 years	Undergraduate	Computer Science	Novice
P16	Male	1-5 years	Undergraduate	Computer Science	Novice
P17	Male	5+ years	PhD	Computer Science	Expert
P18	Male	5+ years	PhD	Computer Science	Expert
P19	Female	5+ years	Master	Computer Science	Expert
P20	Male	1-5 years	PhD	Physics	Novice
P21	Male	1-5 years	Undergraduate	Computer Science	Novice
P22	Male	5+ years	Undergraduate	Computer Science	Expert
P23	Male	5+ years	Undergraduate	Computer Science	Expert
P24	Female	5+ years	Undergraduate	Computer Science	Expert
P25	Female	5+ years	Undergraduate	Computer Science	Expert
P26	Male	< 1 year	PhD	Materials Engineering	End-user
P27	Male	< 1 year	Undergraduate	Game Design	End-user
P28	Male	< 1 year	Master	Business Analysis	End-user
P29	Female	1-5 years	Undergraduate	Business Analysis	Novice
P30	Male	< 1 year	PhD	Industrial Engineering	End-user
P31	Male	< 1 year	Postdoc	Agricultural Engineering	End-user
P32	Male	< 1 year	PhD	Game Design	End-user
P33	Female	< 1 year	Undergraduate	Agricultural Economics	End-user
P34	Male	1-5 years	PhD	Mechanical Engineering	Novice
P35	Male	< 1 year	PhD	Chemical Engineering	End-user
P36	Male	< 1 year	PhD	Mechanical Engineering	End-user
P37	Male	< 1 year	Undergraduate	Mechanical Engineering	End-user
P38	Female	< 1 year	PhD	Communication	End-user

Table 16: Participant Demographics

D.2 User Study Tasks

Task	Category	Description
Task 1	Table Segmentation	Create four tables: “Q1” for sales from January to March, “Q2” for April to June, “Q3” for July to September, and “Q4” for October to December.
Task 2*	Data Imputation	Fill all missing values in the rows across the table with the mean of each respective column.
Task 3	Categorical Analysis	Split the table into separate tables based on the “Educational Level” column. For each resulting table, summarize the “income” values by calculating their mean.
Task 4	Table Integration	Merge two tables based on matching student IDs, keeping only rows where the ID appears in both tables. Then count the number of “Male” entries in the “sex” column of the resulting table.
Task 5	Fuzzy Matching	Merge two tables using fuzzy matching on student names. Separate the merged names into distinct “last name” and “first name” columns. Finally, sort the resulting table alphabetically by last name.
Task 6*	Data Cleaning	Remove rows with excessive missing values (user decides the threshold). For any remaining rows with missing values, fill these entries with the average values of their respective columns.
Task 7	Statistical Analysis	Conduct a statistical analysis comparing the “Years of Experience” column in Table 1 with the “Age” column in Table 2. If the result is statistically significant, remove one of these columns, retaining only the other.

*Single-table tasks

Table 17: User Study Tasks Overview

D.3 Post-task questionnaires

ID	Questions	Scale
User Confidence		
Q1	I felt confident about the final generated data cleaning script.	1 (Strongly Disagree) – 7 (Strongly Agree)
NASA TLX Questions		
Q2	How mentally demanding was using this tool?	1 (Very Low) – 7 (Very High)
Q3	How hurried or rushed were you during the task?	1 (Very Low) – 7 (Very High)
Q4	How successful would you rate yourself in accomplishing the task?	1 (Failure) – 7 (Perfect)
Q5	How hard did you have to work to accomplish your level of performance?	1 (Very Low) – 7 (Very High)
Q6	How insecure, discouraged, irritated, stressed, and annoyed were you?	1 (Very Low) – 7 (Very High)
Ratings of Key features		
Q7	Demonstrating on the uploaded tables is a convenient way to express my intent.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q8	Using natural language in the chatroom is a convenient way to express my intent.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q9 [§]	Answering clarification questions is an effective way to clarify my intent.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q10	Providing feedback through the chatroom is an effective way to fix errors.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q11	Directly editing descriptions of erroneous steps is an effective way to fix errors.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q12 [‡]	Step-by-step NL descriptions helped me understand and validate script behavior.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q13 [‡]	Step-by-step NL descriptions accurately represented the program’s behavior.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q14 [†]	NL summary of the script helped me understand and validate script behavior.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q15 [†]	NL summary of the script accurately represented the program’s behavior.	1 (Strongly Disagree) – 7 (Strongly Agree)
Q16	Seeing table’s data provenance and version contents was helpful.	1 (Strongly Disagree) – 7 (Strongly Agree)
Open-ended Feedback		
Q17	What do you like about DANGO in this setting?	Open-ended
Q18	What do you dislike about DANGO in this setting?	Open-ended
Q19	What suggestions do you have for improving DANGO in this setting?	Open-ended

†: Condition A only; ‡: Condition B, C only; §: Condition C only

Table 18: Post-task survey questions including NASA TLX measures for subjective workload assessment

D.4 Post-study questionnaires

ID	Questions	Scale
Q1	Which condition was more helpful in cleaning data?	(Condition A, B, C)
Q2	Why did you find it more helpful?	Open-ended
Q3	Why did you find other conditions <i>less helpful</i> ?	Open-ended
Q4	Any other thoughts, comments, or feedback?	Open-ended

Table 19: After finishing all the tasks, participants rated which condition was more helpful for data cleaning and provided feedback on why they found certain conditions more or less helpful, using both Likert scale and open-ended responses. (Note: In the survey, UI names were coded to prevent bias)