Data Formulator 2: Iterative Creation of Data Visualizations, with AI Transforming Data Along the Way

Chenglong Wang
chenglong.wang@microsoft.com
Microsoft Research
Redmond, Washington, USA

Bongshin Lee
b.lee@yonsei.ac.kr
Yonsei University
Seoul, Korea

Steven Drucker
sdrucker@microsoft.com
Microsoft Research
Redmond, Washington, USA

Dan Marshall
danmar@microsoft.com
Microsoft Research
Redmond, Washington, USA

Jianfeng Gao
jfgao@microsoft.com
Microsoft Research
Redmond, Washington, USA

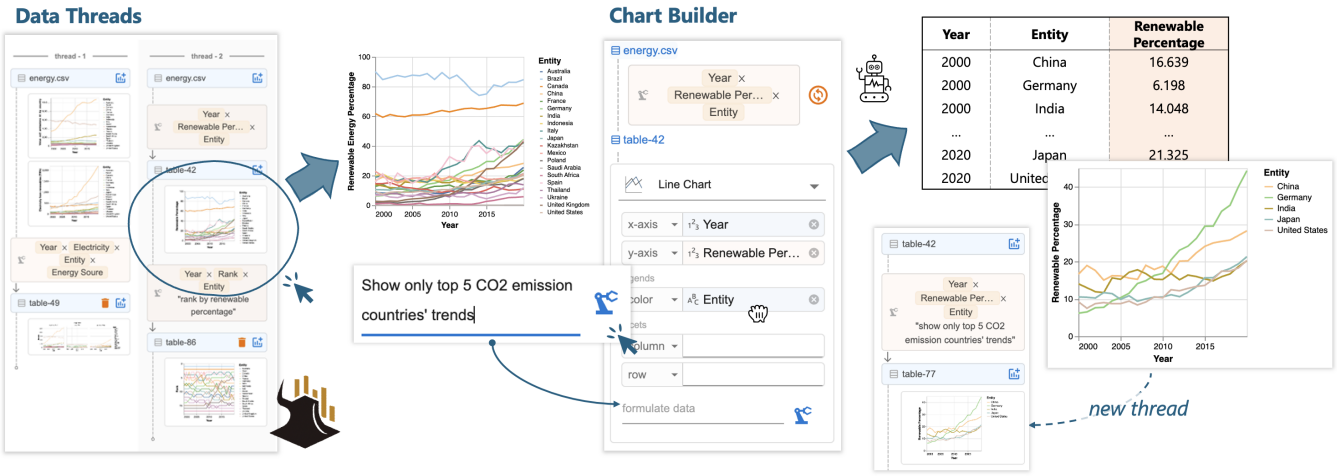

Figure 1: With Data Formulator 2, analysts can iterate on a previous design by (1) selecting a chart from data threads and (2) providing combined natural language and graphical user interface inputs in the chart builder to specify the new design. The AI model generates code to transform the data and update the chart. Data threads are updated with new charts for future use.

Abstract

Data analysts often need to iterate between data transformations and chart designs to create rich visualizations for exploratory data analysis. Although many AI-powered systems have been introduced to reduce the effort of visualization authoring, existing systems are not well suited for iterative authoring. They typically require analysts to provide, in a single turn, a text-only prompt that fully describe a complex visualization. We introduce Data Formulator 2 (DF2 for short), an AI-powered visualization system designed to overcome this limitation. DF2 blends graphical user interfaces and natural language inputs to enable users to convey their intent more effectively, while delegating data transformation to AI. Furthermore, to support efficient iteration, DF2 lets users navigate their iteration history and reuse previous designs, eliminating the need to start from scratch each time. A user study with eight participants demonstrated that DF2 allowed participants to develop their own iteration styles to complete challenging data exploration sessions.

CCS Concepts

• **Human-centered computing** → **Visualization systems and tools**; • **Computing methodologies** → **Artificial intelligence**.

ACM Reference Format:
Chenglong Wang, Bongshin Lee, Steven Drucker, Dan Marshall, and Jianfeng Gao. 2025. Data Formulator 2: Iterative Creation of Data Visualizations, with AI Transforming Data Along the Way. In *CHI Conference on Human Factors in Computing Systems (CHI '25), April 26-May 1, 2025, Yokohama, Japan.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3706598.3713296

1 Introduction

In data exploration [47], even when starting with an initial idea, analysts often need to go back and forth exploring a variety of charts before reaching their goals. Throughout this iterative process, analysts often discover insights that lead them into new directions. However, analysts need to tackle numerous execution challenges:

CHI '25, Yokohama, Japan

in addition to varying chart specifications (as many current tools facilitate), they need to perform and manage different data transformations to support the desired visualization designs. For example, when exploring renewable energy trends, an analyst may find that similar trends across countries make a simple line chart (Figure 1) too dense for detailed comparisons. This observation prompts the analyst to explore the renewable percentage trends of the top 5 CO_2 emitters and how the rankings of these countries have changed over time. To execute the plan, the analyst needs different data transformations: the first requires filtering the data based on each country's total CO_2 emissions, and the second requires partitioning the data by year to compute each country's ranking for that year.

Because data transformation can be difficult to learn and execute, many AI-powered tools have been developed [2, 10, 31, 36, 57, 58]. These tools allow users to describe their goals using natural language and leverage AI models' code generation capabilities [1, 5] to streamline data transformation and chart creation. Despite their success, current tools do not perform well in the *iterative* visualization authoring context. Most of them require analysts to provide, in a single turn, a text-only prompt that fully describes the complex visualization task to be performed, which is usually unrealistic for both users and models.

- First, even though free-form text prompts provide unbounded expressiveness for users to describe their goals, they miss UI interactions' precision and affordances, making it difficult for users to clearly describe complex chart designs. For example, to fully elaborate a faceted bar chart design, the user needs a verbose prompt to clearly specify visual encodings; without it, AI models often misinterpret the intent and create undesired charts, thus requiring further disambiguation efforts from the user. In fact, writing high-quality prompts requires skill and effort. Even with clear goals, inexperienced users sometimes find it difficult to clearly describe their intent in texts [52, 69].
- Second, existing AI-powered tools do not accommodate branching or backtracking, behaviors that commonly occur in the iterative authoring process. Using single-turn text-to-vis tools iteratively requires users to re-specify their intent from scratch for each new design, even for minor updates. This also increases the likelihood of the AI model failing, as it must solve a complex task in a single attempt. While chat-based tools [31, 39, 72] support multi-turn interactions by reusing previous outputs, they struggle with branching contexts. Users often find it difficult to clearly specify which previous messages are relevant for the next iteration. With poorly specified contexts, models may struggle at retrieving important information from the lengthy conversation history to complete the task [17, 26, 70].

To address these iterative chart authoring challenges, our first key insight is to design a **multi-modal chart builder** that blends the shelf-configuration UI [45, 58] with natural language (NL) input to enhance users' ability to structurally specify their chart designs. Resembling traditional shelf-configuration UIs, the chart builder lets user drag existing fields to corresponding visual channels to specify visual encodings. Additionally, users can type in field names that do not exist in the current data to express their intent for creating a visualization that requires data transformation. Coupled with a brief supplemental NL text that elaborates the design, the user can effectively communicate their goal to AI. Since the system can precisely extract chart configuration from the encoding shelf, the user doesn't need a verbose prompt to explicitly explain the design. The AI model then leverages the combined inputs to generate data transformation code to prepare the data required for the chart.

Our second key insight is to introduce **data threads** for users to steer iteration directions. Data threads represent user's non-linear authoring history, allowing users to navigate to an earlier result, fork a new branch, and ask AI to create charts based on that context. This reduces users' input overhead by allowing them to specify incremental updates from a previous result (e.g., "show only top 5 CO_2 emission countries' trends", Figure 1) rather than re-describing the full chart design from scratch. This design also benefits the AI models: the model can reuse previously generated code for new tasks to avoid repeating past mistakes, and it remains free from distractions caused by irrelevant messages from other threads. Data threads also provide a shortcut for users to backtrack and revise prompts to update recently created charts, allowing them to quickly clarify ambiguous inputs or fix errors made by AI.

Based on these designs, we developed Data Formulator 2 (DF2 for short), an AI-powered visualization tool for iterative visualization authoring. [1] DF2 supports diverse charts powered by the Vega-Lite grammar [49], and the AI model can flexibly transform data for different designs, supporting operators like reshaping, filtering, aggregation, window functions, and column derivation. Like other AI tools [10, 58], DF2 provides users with panels to view generated data, transformation code and code explanations to inspect AI-generated contents. To understand how our new interaction designs benefit analysts in solving challenging data visualizations tasks, we conducted a user study consisting of eight participants with varying levels of data science expertise. They were asked to reproduce two professional data scientists' analysis sessions to create a total of 16 visualizations, 12 of which require non-trivial data transformations (e.g., rank categories by a criterion and combine low-ranked ones into one category with the label, "Others"). The study shows that participants can quickly learn to use DF2 to solve these complex tasks, and the tool's flexibility and expressiveness allow participants to develop their own iteration, verification, and error correction styles to complete the tasks. Our inductive analysis of study sessions reveals interesting patterns of how users' experiences and expectations about the AI system affected their work styles. In summary, our main contributions are as follows:

- We designed new interaction approaches, specifically a multi-modal chart builder and a data threads view, to enhance users' ability to specify chart designs and control iteration directions.
- We implemented these designs in DF2, an AI-powered interactive tool that supports the iterative creation of visualizations requiring data transformations.
- We conducted a user study that discovered data analysts' different iteration styles and rich experiences using our new interaction approaches to complete iterative chart authoring tasks. We observed that analysts developed different styles iterating with the AI to perform data analysis, reflecting their personal experience and expectation with the AI model.

[1] Data Formulator 2 is open sourced at https://github.com/microsoft/data-formulator

Figure 2: An analyst explores electricity from different energy sources, renewable percentage trends, and country rankings by renewable percentages using a dataset on CO_2 and electricity for 20 countries (2000-2020, table 1). The analyst creates five data versions in three branches to support different chart designs. DF2 allows users to manage iteration directions and create rich visualizations using a blended UI and natural language inputs.

2 Illustrative Scenarios

In this section, we describe scenarios to illustrate users' experiences for creating a series of visualizations to explore global sustainability from a dataset of 20 countries' energy generation from 2000 to 2020. The initial dataset, shown in Figure 2-①, includes each country's energy produced from three sources (fossil fuel, renewables, and nuclear) each year and annual CO_2 emission value (the CO_2 emission data only ranges from 2000 to 2019). We compare different experiences and skills required for a data analyst, Megan, to complete the analysis session shown in Figure 2 with different tools, computational notebooks versus DF2.

Exploration with computational notebooks. To complete the analysis in a computation notebook, Megan can use R libraries ggplot2 and tidyverse. To use ggplot2 to create charts, Megan needs to make sure that all data fields to be visualized on visual channels (e.g., x, y-axes, color, facet) are columns in the input data, thus, Megan uses tidyverse to transform data when needed.

Figure 2 shows Megan's data analysis session with three branches. She starts with two basic line charts (chart ①-A,B) showing renewable energy and CO2 emission trends. Megan observes that many countries' CO2 emissions have increased despite increased renewable energy use, prompting her to create a faceted line chart (chart ②) and visualize renewable energy percentage trends (chart ③). Discovering that renewable percentage is a better indicator for global sustainability trends, Megan explores two directions: creating a line chart of countries' renewable percentage ranks (chart ④) and highlighting the top 5 CO2 emitters' trends (chart ⑤) compared to global median values (chart ⑥). Throughout the process, Megan backtracks several times to fork new branches from a previous version of data (e.g., charts ② to ③, and ④ to ⑤) and reuses existing results to create new charts (e.g., chart ⑥ from ⑤).

Implementing these charts requires considerable data preparation efforts. While basic charts can be created by mapping existing

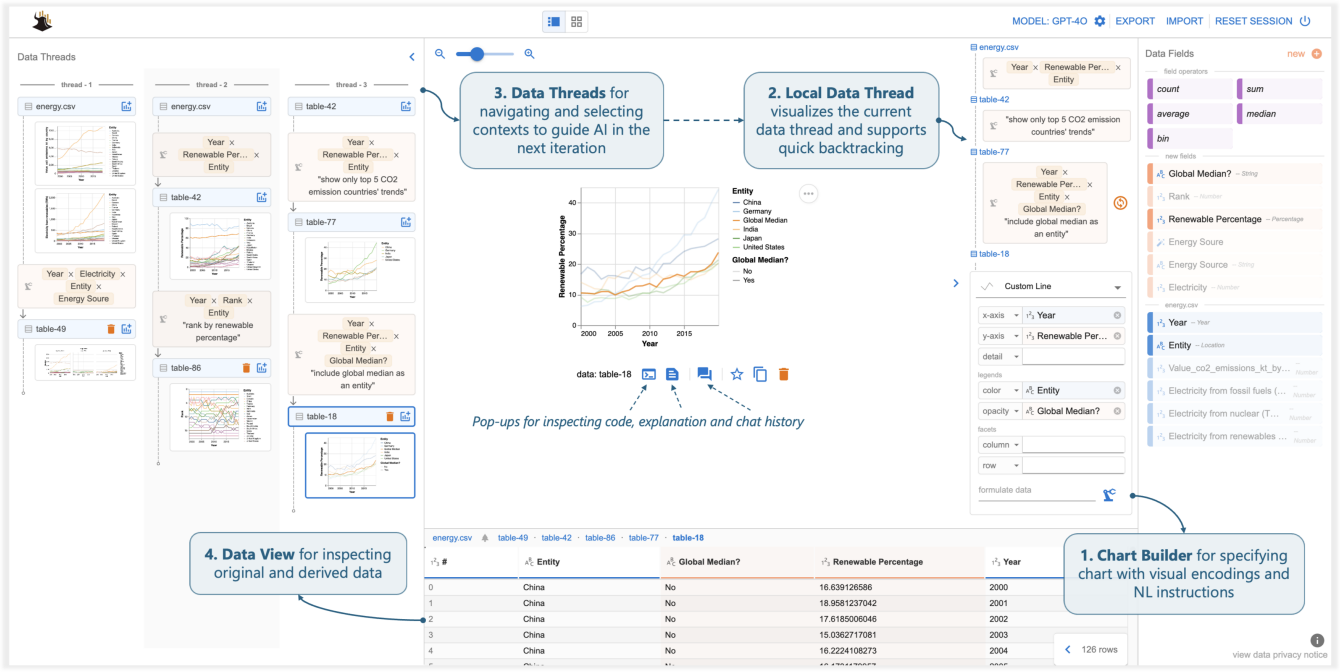

Figure 3: DF2 overview. Users create visualizations by providing fields (drag-and-drop or type) and NL instructions to the Chart Builder, delegating data transformation to AI. Data View shows derived data. Users navigate data history and select contexts for the next iteration using (the thread in use is displayed as local data threads). They refine or create new charts by providing instructions in Chart Builder. The main panel provides pop-up windows to inspect code, explanations, and chat history.

data fields to visual channels (e.g., Year→ x, Electricity from renewables (Twh)→ y, Entity→color for chart ①-A), more complex charts (③-⑥) require different data transformations. For example, Megan needs to reshape the table with pivot_longer to merge energy sources into a new field Electricity for the y-axis (chart ②); to rank countries by renewable percentage (chart ④), she partitions the data by year and uses rank; for charts ⑤ and ⑥, she computes the global median using aggregation and merges the results with the previous table to surface all necessary fields.

Exploration with DF2. Using DF2 to complete the same analysis session, Megan's experience is quite different. Instead of transforming data and creating visualizations with code, Megan's main task is to describe visualization goals with UI interactions and NL inputs and ask the AI model to realize them.

Megan starts with basic line charts to visualize trends of electricity from renewables (Figure 2-①A). Since all three required fields are available from the input data, Megan simply selects the chart type "line chart" in the encoding shelf and drags and drops fields to their corresponding visual channels (Figure 4-①). DF2 then generates the desired visualization. To visualize the CO_2 emission trends, Megan swaps the y-axis encoding with CO2 emissions (kt)→ y.

Megan now needs to create the faceted line chart to compare electricity from all energy sources, which requires new fields Electricity and Energy Source. With DF2, Megan can specify the chart using new data fields and NL instructions in the chart builder (Figure 3-2) and ask the AI to transform the data. As Figure 4-② shows, Megan first drags and drops existing fields Year and Entity to the x-axis and color, respectively. Then, she types in the names of new fields Electricity and Energy Source in the y-axis and column, respectively, to indicate to the AI agent that she expects two new fields to be derived for these properties. Finally, Megan provides an instruction, "compare electricity from all three sources," to further clarify the intent and clicks the formulate button. To create the chart, DF2 first generates a Vega-Lite spec skeleton from the encoding (to be completed based on information from the transformed data). It then summarizes the data, encodings, and NL instructions into a prompt to ask an AI model to generate data transformation code to prepare the data that fulfills all necessary fields, which is then used to instantiate the chart skeleton. After reviewing the generated chart and data, Megan is satisfied and moves to the next task. DF2 also updates data threads (Figure 3-⑤) with the newly derived data and chart. With data threads, Megan can switch the iteration contexts to instruct the AI model to create a new chart either from scratch or reusing a previous result.

Megan proceeds to visualize renewable energy percentage. Although it requires a different data transformation, Megan's experience is similar to the previous one: she drags-and-drops Year and Entity to x-axis and color (Figure 4-③), and enters the name of the new field "Renewable Energy Percentage" on the y-axis. Since Megan believes the field names are self-explanatory, she formulates the new data without an additional NL instruction. DF2 generates the desired visualization (Figure 5-①). To visualize the countries' renewable percentage ranks, building on the previous data, Megan

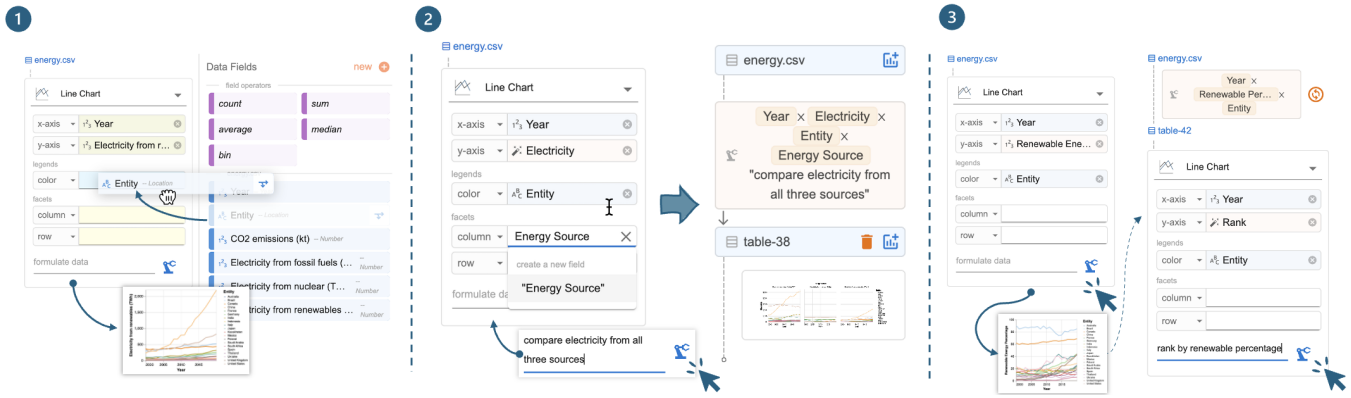

Figure 4: Experiences with DF2: (1) creating the basic renewable energy chart using drag-and-drop to encode fields; (2 and 3) creating charts requiring new fields by providing field names and optional natural language instructions to derive new data.

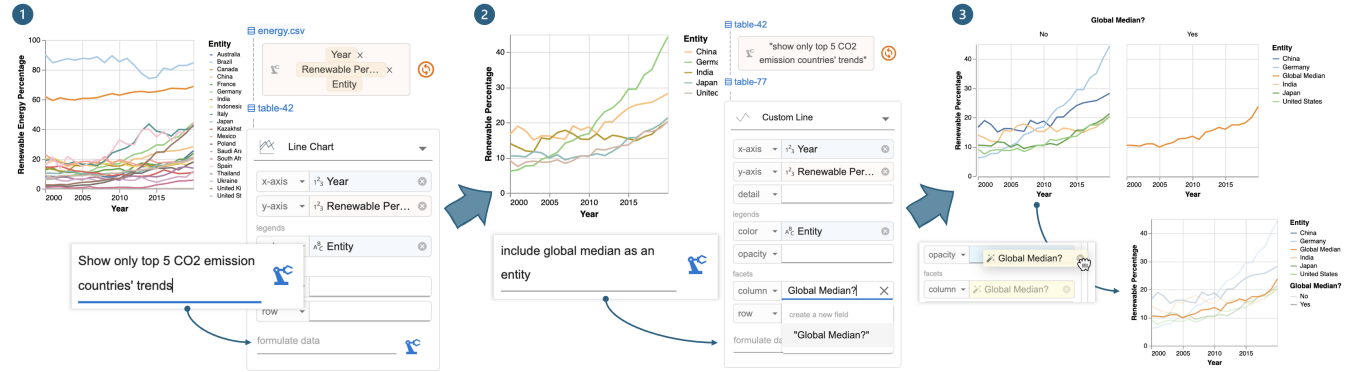

Figure 5: Iteration with DF2: (1) provide an instruction to filter the renewable energy percentage chart by top CO_2 countries, (2) update the chart with Global Median? and instruct DF2 to add the global median alongside the top 5 CO_2 countries' trends, and (3) move Global Median? from column to opacity to update the chart design without deriving new data.

adds a new field "Rank" to the *y*-axis and provides a short instruction. Because Megan builds the new chart on top of the previous data (note that in Figure 4-③, the chart builder box is positioned under the previous table-42 as opposed to energy.csv), the AI model has more contextual information to correctly derive the renewable percentage rank (Figure 2-④) despite Megan's simple inputs.

Next, to visualize the renewable percentage trends of the top five CO_2 emitting countries, Megan decides to build on a previous chart to avoid creating a verbose prompt from scratch. Megan first uses data threads (Figure 3-⑤) to locate renewable percentage chart and opens it in the main panel. On top of that, Megan provides a new instruction below the local data thread, "show only top 5 CO2 emission countries' trends," and clicks the "derive" button (Figure 5-①). DF2 updates the previous code to include a filter clause to produce the new data and visualization (Figure 5-②). Finally, to annotate the chart with global median trends, Megan forks a branch by copying the previous chart, as the new chart requires different encodings (and she wants to keep both visualizations available). Megan updates the visual encoding by (1) typing in a new field name Global Median? for column and (2) providing the edit instruction "include global median as an entity" (Figure 5-②). Once she clicks the derive button, DF2 generates the new chart (Figure 5-③). Upon inspection, Megan prefers to change the visualization type, with global average rendered in a different opacity as opposed to a different subplot. Since these two charts require the same data fields, Megan doesn't need to interact with the AI model — she can directly update the design through the UI: first selecting a new chart type "custom line" (which exposes more chart properties than the basic line chart) and moving Global Median? to the opacity channel. With all desired charts created, Megan concludes the analysis session. Figure 3-③ shows all the data threads from Megan.

Comparison of experiences. These two tools offer different experiences and skill requirements for Megan to execute the analysis. However, both enable her to iteratively refine exploration goals and explore different branches to uncover insights.

The main difference between the two experiences is data transformation. In computation notebooks, Megan needs to prepare data for design updates, even seemingly small ones (e.g., charts-③ and ⑤). She must understand the data shape required and apply the correct transformations (e.g., unpivot for table ②, join and union

for table ⑥). Proficiency in data transformation is essential for creating rich visualizations. In DF2, Megan specifies high-level chart designs, and the AI implements the transformations. Regardless of the underlying data transformations, she conveys her intents uniformly through visual encodings (UI) and natural language inputs. Because Megan can use the shelf-configuration UI to specify chart design, the supplementary NL instruction is straightforward. Though Megan doesn't write code, DF2 provides artifacts like generated data, charts, and code with natural language explanations for her to review. By lowering the implementation skill barrier, DF2 allows users to focus more on analysis planning and reasoning.

Computation notebooks naturally support reuse. Megan can copy-edit previous code snippets or reuse variables to build new charts. In DF2, Megan directs the analysis using data threads. Megan can easily review the history and select previous results to instruct the AI model to create new charts from those contexts. This simplifies instructions to incremental updates, and the AI reuses previous outputs to avoid mistakes. If undesired results occur, she can backtrack and revise inputs using data threads (Figure 3-③). Iteration isn't as easy with a chat-based tool. Iteration isn't as easy with a chat-based tool, where verbose prompts are needed to guide the AI and avoid unrelated histories.

3 System Design

In this section, we present DF2's system design. First, to enable users to specify their intent using multiple paradigms (shelf-configuration UI and NL inputs) DF2 **decouples chart specification from data transformation**, solving them with template instantiation and AI code generation respectively. Second, to support reuse, DF2 organizes **the iteration history as data threads with data as first-class objects**. This enables users to either locate a chart from a different branch and follow up or quickly revise and rerun the most recent instructions leading to the current chart. We will next detail how we implement these designs and explain how additional features help users understand AI-generated results.

3.1 Composing charts from multi-modal inputs

Figure 6 shows how DF2 decouples chart design and data transformation to support blended input methods. Given a user specification, DF2 generates the desired chart in three steps: (1) generate a Vega-Lite specification from the selected chart type, (2) compile a prompt and delegate data transformation to the AI, and (3) instantiate the Vega-Lite specification with the generated data.

Chart specification generation. DF2 adopts a chart type-based approach to represent visualizations, supporting five categories of charts: scatter (scatter plot, ranged dot plot), line (line chart, dotted line chart), bar (bar chart, stacked bar chart, grouped bar chart), statistics (histogram, heatmap, linear regression, boxplot) and custom (custom scatter, line, bar area, rectangle where all available visual channels are exposed). Each chart type is represented as a Vega-Lite template with a set of predefined visual channels, including position (x, y), legends (color, size, shape, opacity), and facet (column, row) that are shown to the user in the chart builder. For example, a line chart is represented as a Vega-Lite template { "mark": "line", "encoding" : { "x": null, "y": null, "color": null, "column": null, "row": null}}, and when the user selects line chart, channels x, y, color, column, and row are displayed in the chart builder. Chart type-based design enable DF2 to support predefined layered charts (e.g., ranged dot plot composed from line and scatter, Figure 7). Additional chart types (e.g., bullet chart) can be supported by adding Vega-Lite templates with respective channels to the library.

As the user inputs fields into the chart builder, either by dragging and dropping it from existing data fields or by typing in new fields they wish to visualize, DF2 instantiates the Vega-Lite template with provided fields. For example, as shown in Figure 6-❶, when the user drags Year→ x, Entity→ y and types Rank in y, the line chart template mentioned above is instantiated with provided fields: if the field is available in the current data table, both field name and encoding type are instantiated (e.g., Year with the temporal type), otherwise the encoding type is left as a "<placeholder>" to be instantiated later when data transformation completes. The shelf-configuration saves users efforts from writing prompts to explain complex chart designs. For example, to create a ranged dot plot–layered chart composed of scatter and line charts–the user only needs to fill the required fields in the UI. DF2 then populates corresponding fields in the predefined chart template (Figure 7).

Data transformation with AI. From the chart builder, DF2 assembles a prompt and queries an LLM to generate python code to transform data. The data transformation prompt contains three segments: the system prompt, the data transformation context and the goal (illustrated Figure 6-❷).

The **system prompt** describes the role of the LLM and the output format. Besides generic role descriptions (i.e., LLM as a data scientist for data transformation), the system prompt guides the LLM to solve the data transformation task in two steps. First, the LLM should refine the user's goal and output as a JSON object that elaborates intermediate and final fields to be computed from the original data. Then, the LLM should generate a python snippet following a provided template. The system prompt ends with an input-output example that illustrates the process. The design rationale behind the "goal refinement" step is to allow the LLM to reason about any potential discrepancy between users' provided fields and their instruction (e.g., users may ask about color by energy type but didn't put "energy type" on the color encoding) and determine the final list of fields to be computed. DF2 then assembles **context prompts** that illustrate the data to be transformed, explaining the data fields by showing the data type and example values for each field, along with sample table rows. The data context provides valuable information related to data formats (such as data types, string formats, and whether columns contain null values) to the LLM, ensuring that the generated transformation code is executable on the given data. When a chart is specified based on previous results, the dialog history between DF2 and the LLM, including user instructions and previously generated code, is appended in context. This way, even if users' follow-up prompts is short, the grounded contexts help the model understand user intent and reuse previously generated code. Finally, DF2 assembles a **goal prompt**, combining the NL instruction provided in the text box and field names used in the encodings. When users skip an NL instruction (Figure 4-❸), the instruction part is left blank. This goal will be refined by the LLM (i.e., based on the system prompt) before attempting to generate the data transformation code. With the full

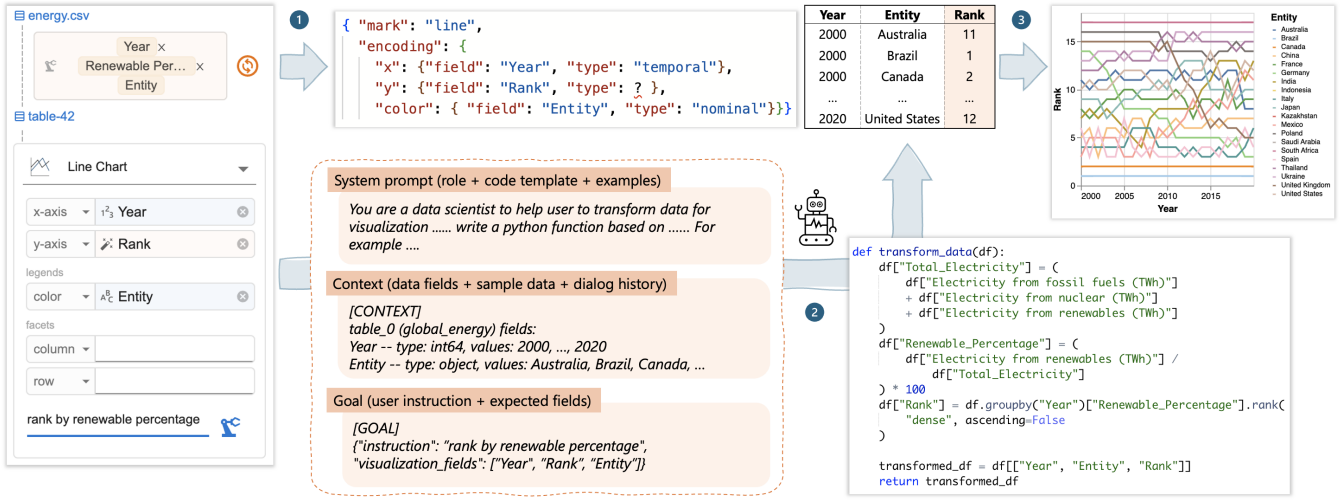

Figure 6: Df2's workflow: (1) Df2 generates a Vega-Lite spec skeleton based on user specifications and chart type. (2) If new fields (e.g., Rank) are required, Df2 prompts its AI model to generate data transformation code. (3) The Vega-Lite skeleton is then instantiated with the new data to produce the desired chart.

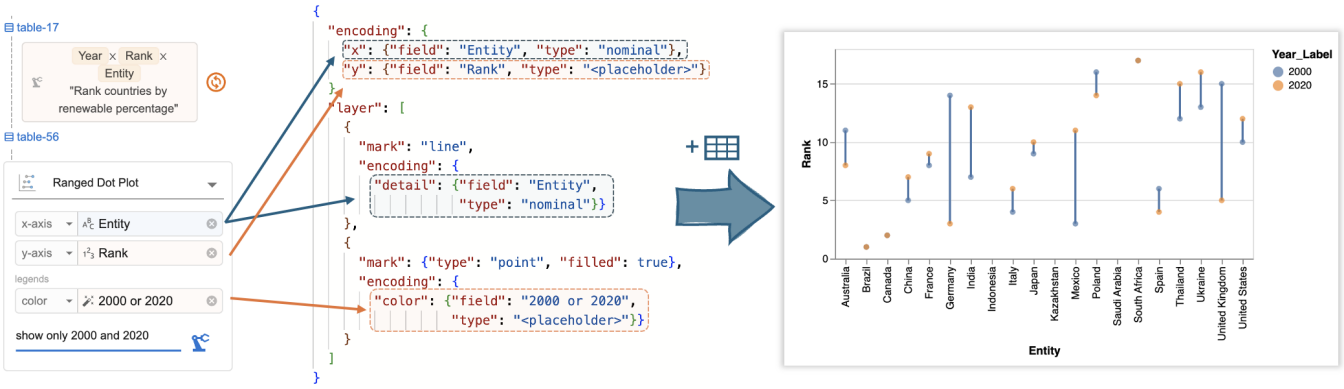

Figure 7: Df2 converts user encodings into a Vega-Lite specification, which is combined with AI-transformed data to visualize country ranks in 2000 and 2020.

input, Df2 prompts the LLM to generate a response. Below shows the LLM's refined goal for the task in Figure 6, and the generated code is shown in Figure 6-❷.

```
{"detailed_instruction": "Calculate the percentage of electricity generated
    from renewables for each country per year. Then, rank the countries by
    their renewable percentage for each year.",
 "output_fields": ["Year", "Entity", "Renewable_Percentage", "Rank"],
 "visualization_fields": ["Year", "Rank", "Entity"],
 "reason": "To rank countries by their renewable percentage, we need to
    calculate the renewable percentage for each country per year and then
    determine the rank based on this percentage." }
```

Df2 then runs the code on the input data. If the code executes without errors, the output data is used to instantiate the Vega-Lite script generated in the previous step. This is done by first inferring semantic types of newly generated columns (to determine their encoding type), and then assembling the data with the script to render the visualization (Figure 6-❸). The generated code sometimes causes runtime errors due to an attempt to use libraries that are not imported, references to invalid columns names, or incorrect handling of undefined or NaN values. When such errors occur, Df2 tries to correct the errors by querying the LLM with the error message and a follow-up instruction to repair its mistakes [9, 38]. The visualization is generated when repair completes. Df2 updates the data threads upon creating the chart.

3.2 Data threads

Data threads visualize the analyst's interaction history with AI, allowing the analyst to control the iteration direction by selecting which data or chart the AI model should use to generate new charts. In data threads, each node represents a version of the data, and these nodes are connected by edges that represent the user's instructions provided to the AI model for data transformation. Visualizations are attached to the data from which they were created. Centering

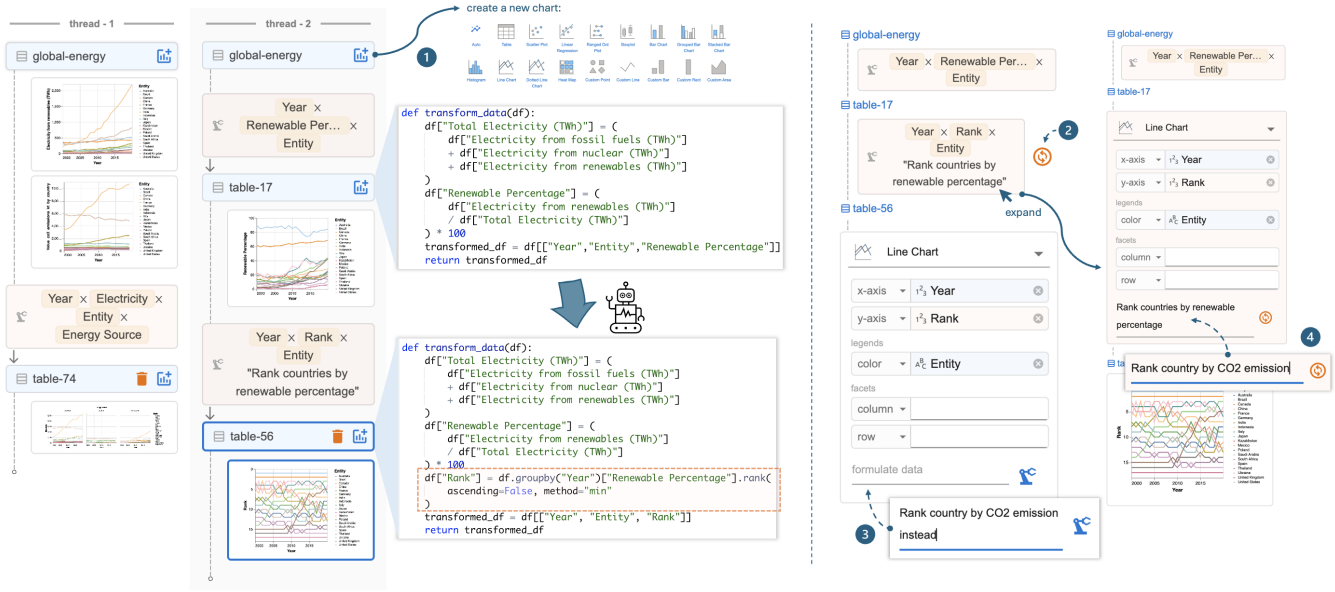

Figure 8: Data threads and local data threads (right). Users can select previous data or charts to create new branches, and the AI reuses code for new transformations based on user instructions. The local data thread offers shortcuts to (1) rerun the previous instruction, (2) issue a follow-up instruction, or (3) expand the previous card to revise and rerun the instruction.

the iteration history around data benefits user navigation because it reflects the sequence of user actions in creating these new data.

When a user issues a follow-up instruction from an existing data or chart, DF2 provides the previous conversation history to the AI and instructs it to rewrite the code towards new goals. Each time the user forks a new branch using data threads, the authoring context switches automatically and is highlighted in the main panel for the user's awareness. This way, the AI model minimizes the risk of incorrectly using information from other branches for data transformation. As shown in Figure 8, the code and the conversation history are attached to each data node. In our design, when the user issues a follow-up instruction, the AI model generates new code by updating the previous code (which may involve additions, deletions, or both) to achieve the user's goal. This ensures that the code always takes the original data as the input, with all information accessible. This way, whether the user wants to update the data (e.g., "now, calculate the average rank for each country"), revise the previous computation (e.g., "also consider nuclear as renewable energy") or create alternatives (e.g., "rank by CO2 instead"), the AI model can achieve these tasks as it has access to the full dialog history and the complete dataset. Note that an alternative design where we only pass current data to the AI model and ask it to write a new code to further transform it (i.e., reusing the data as opposed to reusing the computation leading to the data) would not be ideal. With access to only the current data, this approach cannot handle "backtracking" or "generating an alternative design" styles of instructions effectively.

During iteration, analysts need to both (1) switch to different data or a chart far from the current one to explore a different direction and (2) perform quick follow-ups or revisions of the latest instruction based on the latest data. To accommodate these different needs, DF2 presents both global data threads and local data threads. For global navigation, the key challenge is to help the user distinguish the desired content from others. To address this, data threads are located in a separate panel with previews of data, instructions, and charts to assist navigation (Figure 3). This supports users' differing navigation styles, whether they want to navigate by provenance (i.e., using instruction cards to locate desired data) or by artifacts (i.e., using visualization snapshots to recall data semantics). Once the user locates the desired data, they can click and open a previous chart, displaying it in the main panel. Additionally, they can create a new chart directly from the data Figure 8-❶. In contrast, the local data thread is designed as part of the main authoring panel (Figure 3). It features a much-simplified view (i.e., hiding other visualizations created in this thread) to display a copy of the current thread in use. The main goals of the local data thread are to provide users with awareness of the current iteration context (so they don't need to cross-reference between the chart builder and the data threads panels) and to offer shortcuts for quick revisions of recently created charts. As shown in Figure 8, the user can perform three types of revision tasks with local data threads: rerun the previous instruction (e.g., when the AI produces an incorrect result and they would like to quickly retry, ❷), provide a follow-up instruction to refine the data (❸), and quickly open the previous instruction to modify and rerun the command (❹).

3.3 Assisting user to inspect and style charts

As an AI-powered tool, DF2 allows users to verify AI-generated results and resolve AI's mistakes. It displays the transformed data and the visualization in the main panel and enables users to inspect generated code, its explanation, and the raw chat history through

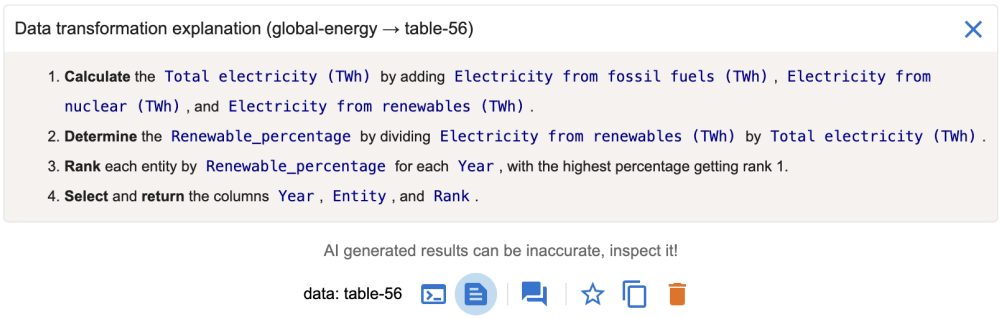

Figure 9: Df2 provides explanations of the code generated by AI to assist users understand the data transformation. This example is the explanation of the code behind table-56 in Figure 8.

pop-up windows (Figure 3). This design accommodates various user verification styles [14, 57] such as viewing high-level correctness from the chart, inspecting corner cases in the data, examining the transformation output, and understanding the transformation process through the code. Df2 utilizes a code explanation module to help users understand the code, querying the AI model to translate code into step-by-step explanations. Figure 9 shows the explanation for the code behind table-56 in Figure 8. Expert users who would like to directly view the raw chat history between Df2 and the AI model (e.g., to inspect the LLM's raw reasoning process) can access this information from the "view chat history" pop-up window. Note that despite that data transformations generated in the later iteration stages can be complex, users can verify its correctness against its predecessor because Df2 users create visualizations incrementally. This lowers users' verification efforts, as found in our study in Section 4. To fix errors, users can take advantage of the data thread's iterative mechanism to rerun, follow up, or revise instructions.

Benefiting from the decoupled chart specification and data transformation processes, when users want to update visualization styles (e.g., change color scheme, change sort order of an axis, or swap encodings) that do not require additional data transformation, they can directly perform edits in the chart builder. By updating channel properties or swapping encoded fields, these updates are directly reflected in the Vega-Lite script and rendered in the main panel. Unlike interactions with AI, which may have a slightly delayed response time, this approach allows users to achieve quick and precise edits with immediate visual feedback to refine the design.

3.4 Implementation

Df2 is a React application with a python server for data transformation. Df2 has been tested with OpenAI models including GPT-3.5-turbo, GPT-4, GPT-4o, and GPT-4o-mini. We used GPT-3.5-turbo in our user study, and all but GPT-4 can generally response within 10 seconds. Df2 can sometimes be slow due to Vega-Lite rendering overhead (e.g., large datasets with more than 20,000 rows, long data threads with more than 20 charts). We envision that on-demand re-rendering of charts can improve its performance.

4 User Study Design

To understand potential benefits and usability issues of Df2, as well as users' interaction styles, we designed a user study that asks participants to reproduce exploratory data analysis sessions involving iteratively creating visualizations.

ID	Role	Chart	Data	Coding	AI	Dataset 1	Dataset 2	Hints
P1	Developer	3	4	4	2	1047s	1666s	1
P2	Data Scientist	3	4	4	4	1636s	1886s	0
P3	Data Architect	3	4	4	4	715s	2207s	0
P4	Developer	1	2	3	2	1036s	1521s	1
P5	Developer	2	2	3	1	1251s	2937s	1
P6	Data Scientist	2	4	2	3	856s	1148s	3
P7	Data Scientist	3	4	3	3	1638s	2372s	1
P8	Developer	3	3	4	1	1043s	1987s	2

Figure 10: Participants' self-reported roles, expertise in chart creation, data transformation, programming, and AI assistants (1=novice, 4=expert), task completion time, and hints needed during study tasks.

Participants. After piloting and refining the study design with three volunteers, we recruited eight participants from a large company. Participants self-rated their skills (Figure 10) on a scale of 1 to 4 ("Novice," "Intermediate," "Proficient," and "Expert") in: (1) chart creation – experience with chart authoring tools or libraries, (2) data transformation – experience with data transformation tools and library expertise, (3) programming, and (4) AI assistants – experience with large language models (e.g., ChatGPT [1]) and prompting.

Setup and procedure. Each study session, conducted remotely with screen sharing, consisted of four sections within a 2-hour slot. After introduction, participants followed step-by-step instructions in the tutorial slides (~25 minutes). Participants then completed a practice task with the option to ask questions (~15 minutes) to test their understanding. Next, participants completed two study tasks, with only clarification questions allowed – we recorded hints they requested. The two study tasks involved creating 16 visualizations, 12 requiring data transformation. Participants were encouraged to think aloud. We concluded with a debriefing to (1) compare participants' Df2 experiences with other tools, (2) understand their strategies using Df2, and (3) gather impressions and suggestions for improvements. Breaks between phases were encouraged.

Tutorial and practice tasks. We used the global energy dataset (described in Section 2) for the tutorial and practice tasks. In the

tutorial, participants followed detailed instructions to recreate the six visualizations from Figure 2 (all but chart ❹). In addition, participants also learned to inspect results and work with the AI's mistakes. In the practice tasks, participants were asked to do similar analyses but focusing on the electricity from nuclear power, they were further asked to create a bar chart to visualize the difference of energy produced from nuclear power between 2000 and 2020 for each country.

Study tasks. To focus on participants' iterative chart creation processes, rather than their ability to create a single chart or derive insights from exploration, we used an *exploration session reproduction* approach. Participants were asked to reproduce two data exploration sessions conducted by an experienced data scientist. We wanted to see if participants could iteratively create charts with Df2, without requiring them to come up with exploration objectives on the fly (otherwise we would limit our participants to highly skilled data scientists). We used two exploration sessions from David Robinson's live stream analysis of Tidy Tuesday datasets.

Figure 11-❶ shows the first data exploration session: given a dataset on college majors and income data (173 rows × 7 columns), participants were asked to create seven visualizations: two basic charts and five requiring data transformation. These visualizations progressively explored the top-earning majors and the relationship between gender ratio and major salary. The process required participants to derive new fields (e.g., gender ratio), filter data (e.g., top 20 earning majors), derive new data (e.g., derive top earning major categories), and perform conditional formatting (e.g., color by top 4 categories and "others"). We provided a task description and reference chart (like chart reproduction studies in [44, 46]) for all but the last two visualizations. Without reference charts for the final two, we asked participants to verify correctness, probing their verification strategies. We did not provide iteration directions, letting participants develop iteration techniques.

Figure 11-❷ shows the second data exploration session: given a movie dataset with budget and gross information (3281 rows × 8 columns), participants created nine visualizations. These visualizations explored movies and genres with the highest return on investment, comparing profit and profit ratios. Besides two basic box plots showing budget and worldwide gross distribution, the other seven charts required data transformation, including calculation and aggregation (average profit and profit ratio for each genre), string processing (extract year for trends), filtering (year > 2000), and partitioning and ranking (top 20 movies for each metric). We hid references for the final two charts to probe participants' verification process. In the following, we use "chart-Ck" and "chart-Mk" to refer to the k-th target charts in Figure 11 for the college and movies datasets, respectively.

5 User Study Results

Here we report user study findings including users' task completion statistics as well as their prompting, iteration and verification styles. We highlight *user quotes* and *example prompts* in this section.

Task completion. All participants successfully completed all 16 visualizations (Figure 10): participants took less than 20 mins on average to finish the seven charts in task 1, and about 33 mins for the nine charts in task 2. Since we let participants deviate from the main exploration task (e.g., in task 2, P4 asked to sort the bar chart for top profitable movies even though it was not required), the recorded completion time is an overestimate of the actual task time. During the study, six participants asked for hints to get unstuck during tasks; we categorize them as follows:

- Task clarification: P1 didn't realize that top movies were restricted to movies after 2000; P4 and P6 required hints about the difference between profit and profit ratio in task 2; P6 asked about whether the x-axis should be Year or Date for movie profit trends.
- Data clarification: P6 an P8 were prompted to notice the difference between fields Major and Major Category in task 1.
- System performance: P5 encountered a performance issue when they created large sized charts. Tn task 2, P5 created multiple bar charts with Movie mapped to the x-axis, resulting in bar charts containing 1300 categorical values. They were advised to reset the exploration session and resume tasks.
- Chart encoding: P7 and P8 required hints on "why the chart didn't render color legends" when they didn't put a field in the color encoding; they expected to specify it only in NL input but not in the concept encoding shelf. [2]

During the debriefing, participants commented that these tasks would be much more difficult to complete with tools they are familiar with. P1, a programming expert, mentioned that they were *"obviously much faster"* with Df2 as it helped with data transformations. When asked about their experience comparing against chat-based AI assistants, participants noted (1) the iteration support makes it easier to create more charts and (2) the UI + NL approach in Df2 is more effective for communicating intent structurally. For example, P2 mentioned *"with ChatGPT, I would have to put a bit more effort to specify the instructions to get what I want, iterations here is much faster with UI."* P4 mentioned that *"with ChatGPT, you need to give much more context, I need to describe in detail about what x,y-axes should be, but here I can just provide with UI,"* and further commented that UI + NL *"helped me in framing and structuring the different transformations that we need to do to get to that end result."*

Iteration styles. Participants developed their own iteration styles working with Df2–Figure 12 illustrates their organization of data threads in their workspaces upon completing the study tasks. Although our participant pool of 8 did not encompass all possible users' data exploration styles with Df2, we observed surprising behavior clusters and distinct approach differences. We characterize participants' iteration styles based on their preferences between "wider" versus "deeper" tree structures, "backtrack and revise" versus "follow up" for providing new instructions to the AI, as well as their preferences for including intermediate tables in their threads.

(Wide versus deep tree organizations): From the high-level organization of data threads, one group of participants (P1, P3, P5, P7, P8) preferred to branch out more often with shorter data threads than the other group (P2, P4, P6), who preferred to create fewer but longer data threads instead. P1 explained that their preference of more branches with shorter data threads came from their coding style of *"creating as many as transformation as I can from one single*

[2] In the study version, Df2 didn't include the feature of resolving conflicts between the users' NL and encoding shelf inputs. This feature was introduced later.

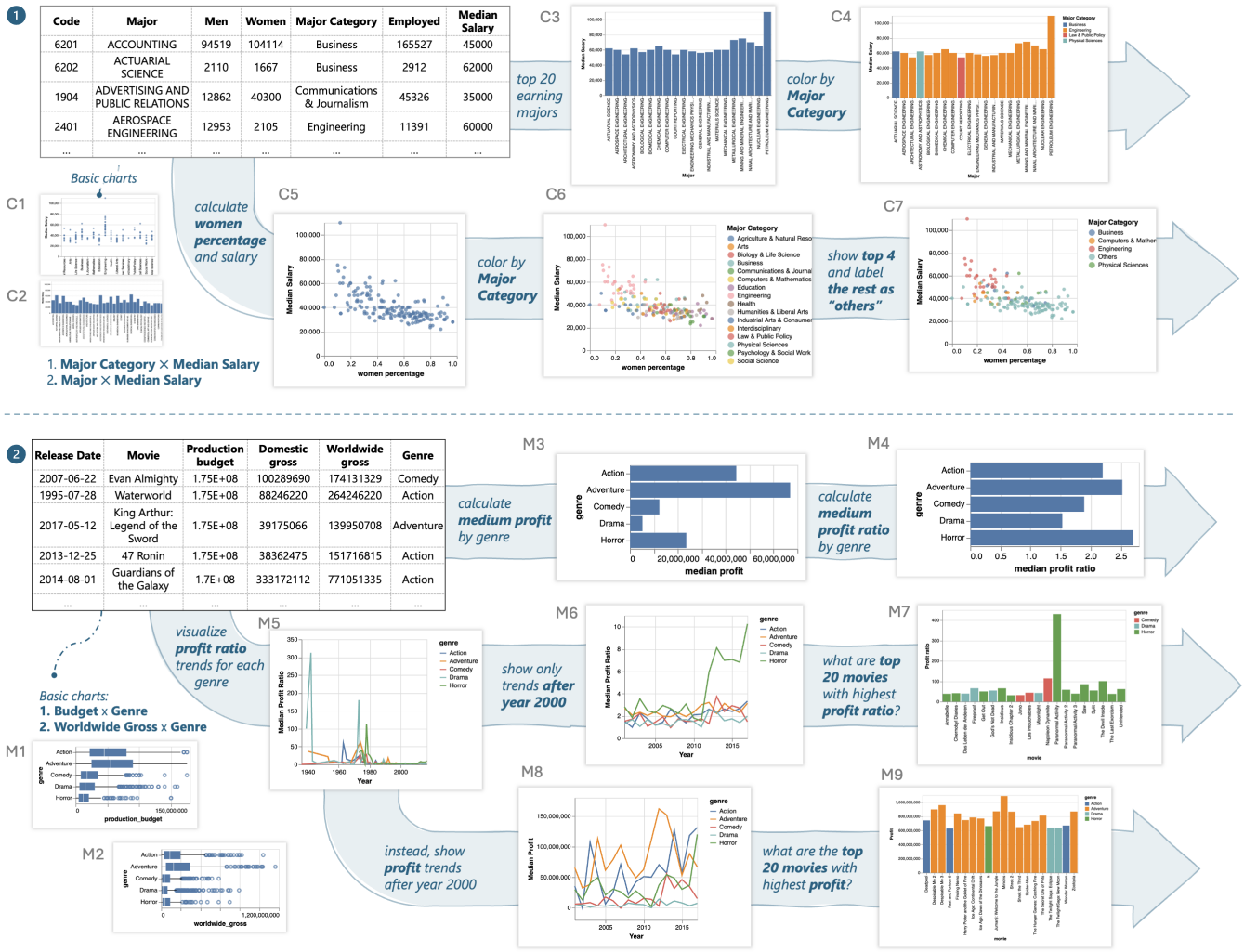

Figure 11: The dataset and tasks in our user study. (1) Dataset 1: Understanding top earning majors and the relation between salary and women percentage. (2) Dataset 2: Exploring movie genres with best return-on-investment values (profit vs. profit ratio) and top movies. The branching directions are added for illustration; participants developed their own iteration strategies. We refer to these target charts as C1-7 for the college dataset and M1-9 for the movies dataset.

table without generating derived tables" to keep the system's memory usage minimal and keep the workspace *"terse."* On the other hand, P2, who preferred longer data threads, mentioned *"I definitely like to be able to just work on top of that and like going forward by just giving a new prompt, because it remembers the context prior to the last one. It ends up generating the right data and visualization."* P2 further commented that *"going back created too much branching"* and they preferred to use longer threads to just provide updates for *"smooth train of thoughts."* To effectively work with long threads, P4 organized their exploration process thoughtfully, as they were *"using the prompts as my anchor, so, when I wanted to figure out where I wanted to go, it was the prompts that I was looking for."*

(Backtracking versus following-up): We observed interesting patterns in participants' preferences when creating new charts or correcting unexpected results: some preferred revising previous instructions (evident from workflows with more self-loop arrows), while others favored following up (characterized by more forward arrows and intermediate gray data nodes). The first group, represented by P1, P2 and P3, preferred to go back and re-issue prompts, either to enrich the previous data to support multiple target visualizations (indicated by yellow nodes with multiple target charts), or to update the data to correct unexpected results. For example, when P1 and P3 worked on coloring the top 20 earning majors with their major categories (chart-C4 in Figure 11), they revised the previous prompt (*"show only top 20 majors based on median salary"* → *"show only top 20 majors based on median salary, include major category"* by P1) to include Major_Category so that both old and the new charts can be created from the same data. To correct a mistake they made in creating chart-M7 (they forgot to instruct the AI to show only top 20

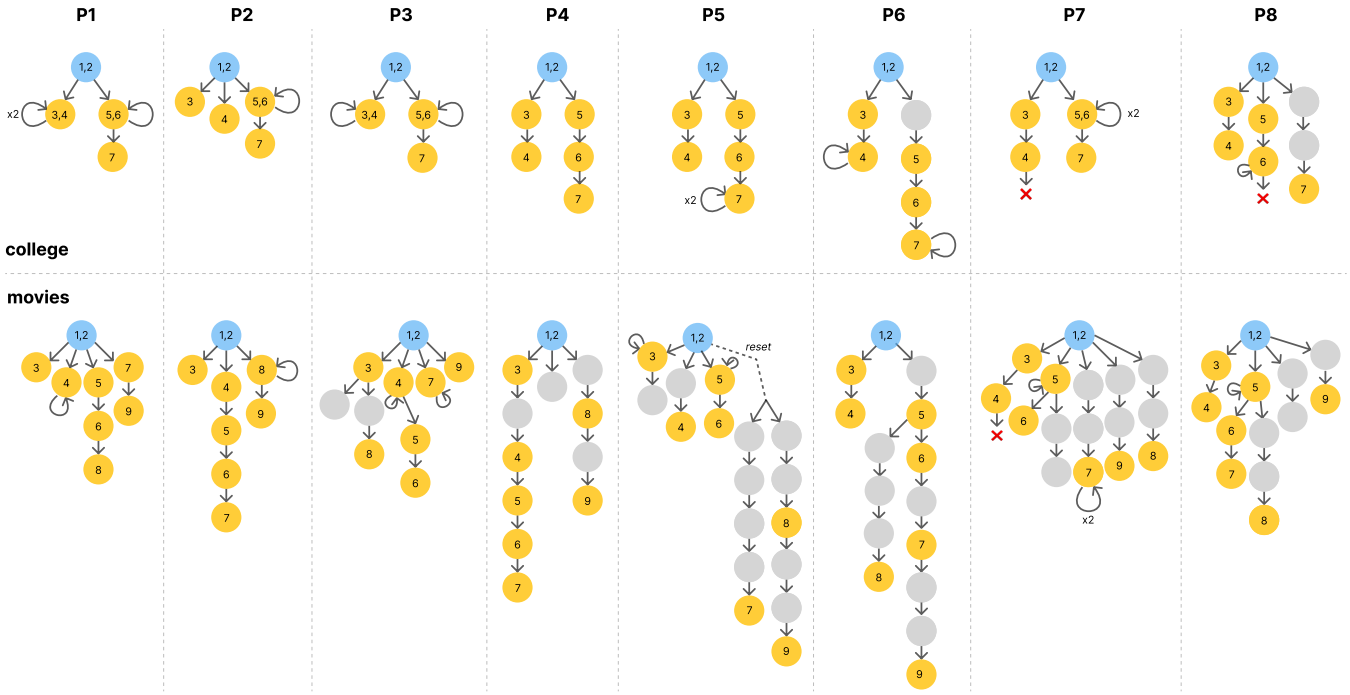

Figure 12: Participants' workflow for study tasks in Figure 11 (C1-7 for college, M1-9 for movie). Each node represents a data table version, with blue for initial datasets, yellow for data tables instantiating (one or multiple) target visualizations in Figure 11 (number *i* in the node indicate the *i*-th target visualizations for the given dataset), and gray for others. Self-loop arrows indicate prompt revisions and data table updates ('×2' indicates two revisions).

movies), P3 chose to go back and revise their previous prompt (*"calculate the profit ratio per movie (worldwide_gross/budget) after 2000"* → *"calculate the profit ratio per movie (worldwide_gross/budget) after 2000 and display the top 20 higher profit ratio movies"*). P1 commented that *"I don't like to pollute my workspace"* and *"I like to keep my workspace as clean as possible."* P3 mentioned that their preference of revision came from the concept of building a *"global expanded dataset"* so that *"[when I] need to calculate the new thing or see a new visual I can come back to the new expanded data set."*

On the other hand, another group, represented by P4, P5, P6, and P7, preferred not only to issue follow-up instructions for new charts but also to provide updates with very brief instructions at each step, creating many intermediate nodes along the way (gray nodes in Figure 12). For example, P5 created chart-M7 (top movies with highest profit ratio colored by genre) in five steps: *"filter movies after year 2000"* → *"show top 5 highest profit ratio"* → *"bring back movie"* (i.e., the Movie field) → *"show top 10"* → *"calculate profit ratio,"* creating four intermediate nodes. P5 noted that *"probably redoing would make sense, but if I can think that I can build on top of that, there is no value for me to go back and start from that, [which] kind of nullify these things [I have done],"* as they preferred to keep their work around. P6 mentioned that they adapt their iteration style based on the type of mistakes they encountered: *"if it is something intermediate where I've made the mistake, I'll go [create a new instruction] and fix the previous step"* but when it *"is a totally new kind of visualization I have in my mind"* or *"if it is something I missed altogether, I will just cancel the whole thing and start from scratch."*

(Choices of data to iterate on): Participants had different strategies deciding which previous data/charts to use to create new charts. P1 chose to derive the new chart from a previous chart that shares similar visual design. For example, P1 created chart-M9 from M7 since they are both bar charts showing top ranked movies, despite one is based on profit while another is based on profit ratio. In a different fashion, P2, P4 and P5 often branch out based on similarity of computations used. For example, P2 created chart-M7 about movies with highest profit ratio based on chart-M6 showing profit ratio trends for each genre over time, as they shared the same computation "profit ratio." P2 explained their data-centric approach was because they *"prefer to have more control over the data as opposed to the chart later on."* They also appreciated that Df2 *"sort of brings together both data-centric and chart-centric people."*

Prompt styles. Prompts created by participants are all short (less than 20 words). We observed that participants created diverse styles of prompts, both in terms of how they phrase the instruction (e.g., question, command) and the subject they asked (e.g., describing expected visual output or output data property, providing computation formula). The most common style of prompts is imperative commands, that either describe the transformation to be conducted or the property of the desired output. For example, to filter top earning movies, participants used prompts *"show only top 20"* [P6] and *"filter top 10 movies based on median profit"* [P5]. Participants

also used command-style prompts for describing computations (e.g., *"calculate ratio of worldwide_gross by production_budget"* [P5]) and for visual updates (e.g., *"color by major category"* [P8]. We also observed that some participants prompted with questions (e.g., *"can you show only the top 5 countries in terms of increases?"* [P7]), or prompted in a chat style (e.g., *"Good. We need to now find the Median profit ratio each year for each genre"* [P2].

One participant, P5, had a distinct prompting style, that directly asked the AI to add, mutate, or retrieve columns on top of the previous data. For example, P5 asked *"bring back major category"* to create chart-C4 from C3, *"divide by 100,000"* for updating profit units, *"bring back release_date"* before they used a follow-up command *"show only year greater than 2,000"* to filter movies by date. P7 preferred to use more verbose prompts to reiterate the computation they intended to achieve whenever they mentioned the concept. For example, to ensure that the AI would not interpret the computation differently, they copy/pasted the formula to the prompt whenever they mentioned profit ratio — *"median profit ratio (worldwide_gross/production_budget) by year and by genre."* P6, on the other hand, preferred to use no additional prompts and provided more descriptive field names. For example, to create chart-C5, they mapped "percentage_of_women_of_Total_Men_and Women" to x-axis, Median_Salary to y, and provided no prompt in the input box. In fact, we observed that Df2 can reliably transform data with self-explanatory field names (e.g., "renewable energy percentage", "women percentage", and "difference between 2020 and 2000") without any additional prompts. Some participants' preference for using shorter names and additional (short) prompts was *"to minimize the error space [for AI]"* [P7].

Verification. To proceed through iterative exploration, or repeat/-correct a step, participants needed to understand the chart and verify that the transformation was performed correctly. Most of the time, participants spotted unintended output easily through incorrect patterns in rendered visualizations. This happened especially when there were differences in visual encoding (e.g., when P5 incorrectly mapped release_date to the x-axis instead of year on chart-M5), cardinality (e.g., when P6 incorrectly asked the AI to color the bars by major instead of major_category for chart-C6), or high-level patterns (e.g., when P7 requested women versus median_salary for chart-C7, leading to results based on the count of women instead of the percentage). When the transformation is straightforward, participants visually inspected the chart and data to verify correctness. For example, after P3 asked *"filter the year after 2000"* to show only profit ratio trends for movies after 2000 (chart-M8), they checked the x-axis domain and compared the generated chart with the pre-filtered one. Similarly, after P2 input *"filter results to top 20 by major"* to find the highest earning majors (chart-C5), they referred to the previous chart with all of the majors' median salary sorted to check filtering correctness.

To check whether unobvious computations were done correctly (e.g., whether the LLM computed profit ratio correctly), different participants' background impacted how they validated the results: participants either referred to (1) explanations of the code, (2) the actual code (even if they are non-python programmers), or (3) values in the result table to check correctness. P3 mentioned *"as an expert, I like to see the prompt to the model, and then the code generated; but as a business user, I would imagine using more data, chart, and explanations."* while P4 commented *"[explanation] steps were really, really helpful in terms of figuring out whether it is doing the right thing as to what I'm asking it to do. That and also the data chart underneath."* P7 noted that, for trust, the definition of a new field is more crucial than the actual code: *"I just want to make sure that definition, like profit ratio, when I check in, I only look at those definitions if they are correct. I'm less worried about the real coding piece."* Thus, they use code explanations frequently to check definitions. Meanwhile, P7 stated that they felt some pressure from the study environment not to spend too much time understanding code for which they were not familiar with, but they would trust code more. We also observed participants who developed trust in a workflow (by examining code and data tables) when it was straightforward, and then, they assumed the more complicated transformations built on top of these steps worked.

Additional Feedback. Several users noted potential improvements of Df2. P1 commented on how small interface variations might give different affordances. For instance, *"if there was a large view for data threads, it would encourage me to do more transformations and do more branching."* P3 mentioned that they prefer the AI to ask the user to disambiguate when the intent is unclear rather than trying to solve the task with unclear specification. P7 used instructions that were very detailed and sometimes incorrect, which in turn, made iteration more difficult, since it was difficult to incrementally modify these instructions. We discussed the potential of having templates or AI feedback for instruction crafting to reduce errors.

6 Discussion and Future Work

Supporting recommendations in exploratory analysis. Df2 focuses on visualization authoring, where an AI completes tasks needed to achieve a user's intended action. We envision that Df2 can be enhanced with recommendation capabilities like Voyager [65], Draco [35], and Lux [25] for suggesting visualization goals to help users "cold start" their analysis. Df2's designs can benefit user experiences with visualization recommendation tools in two ways. First, because Df2 supports visualization beyond initial data formats, it overcomes the limitation of most existing tools, which only consider fields in the input table for recommendation. Second, Df2's data threads provide a natural way for users to follow up the system's initial recommendations, either to dive deeper into an exploration direction, revising suggested charts, or to ask for different recommendations. To achieve this, we can add a recommendation component that can suggest a list of fields to be explored and let Df2 prepare data to surface the fields and create visualizations. The initial recommendation of the fields of interests can be generated either automatically from the analysis of input data characteristics or in a mixed-initiative approach, i.e., leveraging AI to generate them using a high-level natural language instruction provided by the user — these fields do not have to be fields in input data, as Df2 can transform the data to derive them from existing ones. While Df2's data transformation ability can extend the visualization exploration space, thus bringing in more potential insights to be discovered, it also increases the chances of suggesting field combinations that are either trivial, distracting, or even biased. Therefore, as part of future work, it would be valuable to explore ways to support visual

recommendation in a larger exploration space, especially managing and communicating exploration paths to the user to prevent unintentional bias towards an undesired direction.

Coordinating data transformation and chart editing. DF2 derives new data based on users' inputs to instantiate the chart design, but it does not modify the chart itself (represented as a Vega-Lite specification). When the user wants to refine the chart design (e.g., updating color scheme or x-axis ordering), they edit it through GUI widgets in the encoding panel after the chart is created. This design leverages the natural and precise nature of UI updates, providing immediate visual feedback [55]. It also utilizes current models' strengths in data transformation for more reliable outputs [12, 24]. In contrast, current models perform less effectively in editing charts or generating charts from data based on NL instructions, even when the data is prepared [6]. Despite this, some participants in the user study showed interest in asking DF2 to perform chart edits within the chart builder alongside data transformation. A potential solution is an agent-based system [66, 71] that plans whether to transform data, edit the chart script, or both based on user inputs, and dispatches agents to handle these tasks. The key challenge is managing response time and maintaining reliability, as AI agents often require multiple interactions to reach consensus.

Asking users to clarify ambiguous inputs. DF2 adopts a generation verification approach: AI attempts to complete the user's request, and the user inspects the result to provide follow-up instructions. This interaction loop is enhanced by DF2's local data thread design. There is an opportunity to make AI more proactive, such as actively seeking clarification from users when their inputs are ambiguous, before attempting to solve the task. This could reduce users' verification and revision efforts. For example, when the user issues an unclear request (e.g., *"show top 5"*), the system can first analyze the goal, and then either present a refined goal for confirmation (e.g., *"do you mean top 5 by renewable percentage?"*) or ask the users to clarify their intent (e.g., *"what criteria should be used for ranking?"*). This proactive approach could also promote users' trust in the AI system. It is an interesting research direction to explore ways to prompt or train AI models to ask only necessary clarification questions, preventing users from being overwhelmed with low-level questions that might interrupt their workflow.

Study limitations. In our user study, we used the reproduction of professional data analysts' exploration sessions as the study tasks, rather than asking participants to perform free explorations. This choice was made to minimize the impact of participants' data analysis skills on their experience with DF2, as our goal was not to assess their exploration skills. A follow-up study, where participants perform open exploration with their own data, can further investigate how DF2 can assist analysts with planning during exploration. In addition, as a limitation of our lab study, we could not capture users' longer-term learning effects. A future longitudinal study could further investigate how users' expectations with DF2 change over time and how this affects their specification styles and iteration strategies.

7 Related Work

Compared to its predecessor, Data Formulator [58], DF2 has transitioned from a single-turn chart authoring tool into an iterative visualization tool designed for data exploration. Concretely, Data Formulator [58] is a single-turn authoring tool that leverages different authoring paradigms for various types of data transformations. It uses programming-by-example for table reshaping and employs LLMs to generate code for single column derivation. However, users may struggle with choosing the appropriate paradigm for the required transformations. DF2 unifies the interaction paradigms with a blended UI and natural language input design, supporting iterative authoring. This allows users to build new charts from previous ones with minimal additional specification. DF2's new interaction approaches not only broaden the expressiveness of supported data transformations but also reduce the users' specification overhead. We next illustrate related work on chart authoring, data transformation, and data exploration tools that inspired the design of DF2.

LLM-powered visualization tools. Large language models' code generation ability [1, 5, 29, 54] motivates the designs of new AI-powered visualizations tools [11, 31, 53, 58] that allows users to create visualization using high-level natural language descriptions. For example, LIDA [10] can summarize data and use LLM to generate python code to generate visualizations. Because LLMs can struggle in understanding complex chart logic, ChartGPT [53] decomposes visualization tasks into fine-grained reasoning pipelines (e.g., column selection, filtering, chart type selection, visual encoding), using chain-of-thoughts prompting [59]. As single-turn interactive tools, they are not suitable for iterative analysis. For multi-turn interactions, users can directly chat with LLMs in Code Interpreter [1] or Chat2Vis [31]. Code Interpreter equips the LLM with a Python interpreter so that the model can generate and execute code to transform data and create charts; Chat2Vis includes visualization-specific prompts to help the model generate visualizations more reliably. Since these tools organize the dialog linearly, users need to put in extra effort to clarify the context when there are branches, to reduce the chances of models applying incorrect contexts and making mistakes in the new task [17, 26, 70].

DF2 is also an LLM-powered tool that shares similar prompt designs to LIDA and Chat2Vis (e.g., the use of data summaries) and supports NL interaction. The key difference is that DF2 blends UI and natural language inputs for chart specification, balancing precision and flexibility, rather than requiring users to describe everything in text. DF2's data threads generalize linear contexts used in existing dialog systems, allowing users to control the iteration direction by providing authoring contexts to the AI model.

Other AI and synthesis-powered tools. Besides LLM-powered tools above, neural semantic parsing [7, 34, 36], and program synthesis-based tools [57] have also been developed to address the visualization challenge. For example, NL4DV [36] and NcNet [30] leverage recurrent neural networks trained to translate NL queries into charts. NL2Vis [67] and Graphy [7] use a semantic parser to extract entities from the user's NL query and apply program synthesis algorithms to compose charts. Unlike LLMs, these tools are more restrictive in the supported data transformations and chart types, requiring very specific chart descriptions from the user. While programming-by-examples (PBE) techniques are developed to tackle data reshaping challenges in chart authoring (e.g., Falx [57] and Data Formulator [58]'s reshaping module), users need to prepare low-level examples to demonstrate the transformation

intent, which deviates users from the high-level visualization workflow. For disambiguation, DataTone [13] introduces disambiguation widgets for users to experiment with different entity extraction outputs for the generated query, and users can inspect paraphrased queries (in NL) to resolve ambiguity; Falx [57] previews charts from multiple versions of data consistent with user examples. Benefiting from the use of LLMs, DF2 is more expressive. Inspired by how prior work displays candidate results and explains code to help users understand system outputs [13, 14, 58], DF2 displays generated code, data, chart and code explanation to assist user inspection.

Visualization grammars and tools. The grammar of graphics [64] inspired many modern visualization grammars (e.g., ggplot2 [61], Vega-Lite [49], Altair [56]), where visualizations are mainly described by mappings from data columns to visual channels. Comparing to more expressive languages like D3 [3] and Atlas [27], high-level grammars hide the computation process of linking data items to visual objects to reduce visualization effort. Powered by these high-level grammars, interactive tools like Lyra [48], Data Illustrator [28], Charticulator [45], Tableau [51]) have been introduced, where users leverage the shelf-configuration interface to specify visual encodings. To reduce authoring efforts, tools like Voyager [65], Lux [25], and Draco [35] leverage rule and logic-based recommendation techniques to suggest visualizations from users' partial chart specifications. For example, Voyager lets users put a wildcard field into an encoding slot, and then automatically instantiates the wildcard field with different existing fields from the table, to produce interesting charts for users to explore. Note that these tools all require tidy input data [62], where all fields to be visualized should be data columns. Thus, users need to learn to use data transformation tools to prepare data [8, 18–20, 22, 40–42, 63].

DF2 benefits from Vega-Lite's expressiveness to support rich visualization designs. DF2 inherits the shelf-configuration design from existing interactive tools and enhanced it with NL inputs for users to create charts that require data transformation. While DF2's custom fields resemble wildcard fields in Voyager [65], they are semantically different: a custom field is for a field that users desire to visualize but not yet exist in the current table, requiring data transformation to surface, while a wildcard field refers to a field in the current table that the user does not specify explicitly. There is potential to unify these two as "wildcard custom fields" so that the system can recommend unspecified fields beyond the available fields in the current data (leveraging data transformation), which would broaden the exploration space.

Exploration history. Graphical history [16] and data provenance [4] are essential in visualization authoring, especially in exploration tasks where branching and iterations are common. In computation notebooks, the exploration history is organized based on code blocks [33, 37]. Data transformation tools like somnus [68] and Tableau Prep visualize data provenance based on transformation operators. Directed-graph models [23, 50] based on visual similarity are also used for visualization organization. To assist data scientists manage (messy) programming histories in computation notebooks, Verdant [21] introduces a design that visualizes users' edit histories of notebook and artifacts, allowing them to revisit different versions of the notebook; code gathering tools [15] leverage data dependency to extract a clean and minimal code snippet from a notebook that can reproduce a variable of interest. To support the management of different versions of code snippets created during the ideation process, Variolite [21] allows users to explicitly create branches when experimenting different implementations of a function and to switch among them later on.

DF2's data threads draw inspiration from these systems. The key difference is that data threads are designed for users to steer iteration directions by providing authoring contexts with AI. This approach organizes history around high-level user interactions with AI and hides operator-level details. We characterized users' interaction strategies based on their exploration tree [60]. Provenance management techniques for notebooks can be applied to manage long data threads users created across different sessions (e.g., compressing long data threads into shorter ones with summaries). In the future, DF2 could render data threads as hierarchical trees [23] to support navigation of large data threads in multiple granularities. Additionally, it could incorporate version toggles, similar to Variolite, allowing users to explore different versions of generated code more compactly, rather than presenting all exploration branches as separate data threads.

Multi-modal interaction. Despite natural language providing flexible and expressive interactions between human and AI, NL-only interaction is not always optimal for the users to clearly convey their intent, especially for conveying designs pictured only in the user's mind. To address this limitation, multi-modal models like ChatGPT [1] and Gemini [43] have been introduced, allowing users to provide audios and images in their conversation with AI. New interactive tools are also developed to support multi-modal interaction. For example, DirectGPT [32] allows users to directly point and click on a canvas to specify contexts or objects that NL instruction is based on to reduce prompting effort, Mage [22] provides interactive widgets for users to control content in notebook, and DynaVis [55] generates UI widgets dynamically based on user's NL inputs for chart editing with LLMs so that users can explore and repeat edits and see instant visual feedback from edits. DF2's chart builder bridges the precision and affordance of GUI interaction with flexibility of NL inputs and thus exploits a multi-modal UI design for visualization authoring.

8 Conclusion

Visualization authors often create visualizations iteratively, alternating between data transformation and visualization steps. This process requires proficiency with tools and considerable effort to manage various versions of data and charts. Although AI-powered tools aim to reduce user effort, they fall short for iterative analysis, expecting users to specify their intent at once with NL inputs. We present DF2, an interactive system for iterative visualization authoring. DF2 features a multi-modal UI that allows users to specify visualizations using a blend of UI and NL inputs, enabling users to convey complex designs more precisely without verbose prompts. To help user manage iteration directions, DF2 introduces data threads for users to navigate, branch, and reuse previous designs. In a user study with eight participants reproducing two challenging data exploration sessions consisting of 16 visualizations, we observed that DF2 enabled participants to develop their own iteration and verification strategies confidently with minimal hints.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

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

[3] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE Trans. Vis. Comput. Graph.* 17, 12 (2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[4] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*. Springer, 316–330.

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[6] Nan Chen, Yuge Zhang, Jiahang Xu, Kan Ren, and Yuqing Yang. 2024. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics* (2024).

[7] Qiaochu Chen, Shankara Pailoor, Celeste Barnaby, Abby Criswell, Chenglong Wang, Greg Durrett, and Işil Dillig. 2022. Type-directed synthesis of visualizations from natural language queries. *Proceedings of the ACM on Programming Languages* 6, OOPSLA2 (2022), 532–559.

[8] Qiaochu Chen, Xinyu Wang, Xi Ye, Greg Durrett, and Isil Dillig. 2020. Multi-modal synthesis of regular expressions. In *Proceedings of the 41st ACM SIGPLAN conference on programming language design and implementation*. 487–502.

[9] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128* (2023).

[10] Victor Dibia. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics using Large Language Models. *arXiv preprint arXiv:2303.02927* (2023).

[11] Victor Dibia and Çağatay Demiralp. 2019. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE computer graphics and applications* 39, 5 (2019), 33–46.

[12] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[13] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, UIST 2015, Charlotte, NC, USA, November 8-11, 2015*, Celine Latulipe, Bjoern Hartmann, and Tovi Grossman (Eds.). ACM, 489–500. https://doi.org/10.1145/2807442.2807478

[14] Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M Drucker. 2023. How Do Analysts Understand and Verify AI-Assisted Data Analyses? *arXiv preprint arXiv:2309.10947* (2023).

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

[16] Jeffrey Heer, Jock Mackinlay, Chris Stolte, and Maneesh Agrawala. 2008. Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE transactions on visualization and computer graphics* 14, 6 (2008), 1189–1196.

[17] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint arXiv:2404.06654* (2024).

[18] Yanwei Huang, Yunfan Zhou, Ran Chen, Changhao Pan, Xinhuan Shu, Di Weng, and Yingcai Wu. 2023. Interactive table synthesis with natural language. *IEEE Transactions on Visualization and Computer Graphics* (2023).

[19] Zhongjun Jin, Michael R. Anderson, Michael J. Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 683–698. https://doi.org/10.1145/3035918.3064034

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

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

[22] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. Mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '20)*. Association for Computing Machinery, New York, NY, USA, 140–151. https://doi.org/10.1145/3379337.3415842

[23] Younghoon Kim, Kanit Wongsuphasawat, Jessica Hullman, and Jeffrey Heer. 2017. Graphscape: A model for automated reasoning about visualization similarity and sequencing. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2628–2638.

[24] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*. PMLR, 18319–18345.

[25] Doris Jung Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, and Aditya G. Parameswaran. 2021. Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows. *Proc. VLDB Endow.* 15, 3 (2021), 727–738. https://doi.org/10.14778/3494124.3494151

[26] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173.

[27] Zhicheng Liu, Chen Chen, Francisco Morales, and Yishan Zhao. 2021. Atlas: Grammar-based Procedural Generation of Data Visualizations. In *2021 IEEE Visualization Conference, 2021 - Short Papers, New Orleans, LA, USA, October 24-29, 2021*. IEEE, 171–175. https://doi.org/10.1109/VIS49827.2021.9623315

[28] Zhicheng Liu, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. 123:1–13. https://doi.org/10.1145/3173574.3173697

[29] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173* (2024).

[30] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. *IEEE Trans. Vis. Comput. Graph.* 28, 1 (2022), 217–226. https://doi.org/10.1109/TVCG.2021.3114848

[31] Paula Maddigan and Teo Susnjak. 2023. Chat2vis: Generating data visualisations via natural language using chatgpt, codex and gpt-3 large language models. *Ieee Access* (2023).

[32] Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. 2024. Directgpt: A direct manipulation interface to interact with large language models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–16.

[33] 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.

[34] Rishab Mitra, Arpit Narechania, Alex Endert, and John Stasko. 2022. Facilitating conversational interaction in natural language interfaces for visualization. In *2022 IEEE Visualization and Visual Analytics (VIS)*. IEEE, 6–10.

[35] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. 2019. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Trans. Vis. Comput. Graph.* 25, 1 (2019), 438–448. https://doi.org/10.1109/TVCG.2018.2865240

[36] Arpit Narechania, Arjun Srinivasan, and John T. Stasko. 2021. NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries. *IEEE Trans. Vis. Comput. Graph.* 27, 2 (2021), 369–379. https://doi.org/10.1109/TVCG.2020.3030378

[37] Observable. [n. d.]. *https://observablehq.com/*.

[38] Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2023. Is Self-Repair a Silver Bullet for Code Generation?. In *The Twelfth International Conference on Learning Representations*.

[39] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, and Sam Altman et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[40] The pandas development team. 2023. *pandas-dev/pandas: Pandas*. https://doi.org/10.5281/zenodo.7741580

[41] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: a framework for inductive program synthesis. In *Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2015, Pittsburgh, PA, USA, October 25-30, 2015*, Jonathan Aldrich and Patrick Eugster (Eds.). ACM, 107–126. https://doi.org/10.1145/2814270.2814310

[42] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.). Morgan Kaufmann, 381–390. http://www.vldb.org/conf/2001/P381.pdf

[43] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat,

Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).

[44] Donghao Ren, Matthew Brehmer, Bongshin Lee, Tobias Höllerer, and Eun Kyoung Choe. 2017. Chartaccent: Annotation for data-driven storytelling. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*. Ieee, 230–239.

[45] Donghao Ren, Bongshin Lee, and Matthew Brehmer. 2019. Charticulator: Interactive Construction of Bespoke Chart Layouts. *IEEE Trans. Vis. Comput. Graph. (Proceedings of InfoVis)* 25, 1 (2019). https://doi.org/10.1109/TVCG.2018.2865158

[46] Donghao Ren, Bongshin Lee, Matthew Brehmer, and Nathalie Henry Riche. 2018. Reflecting on the Evaluation of Visualization Authoring Systems : Position Paper. In *2018 IEEE Evaluation and Beyond - Methodological Approaches for Visualization, BELIV 2018, Berlin, Germany, October 21, 2018*, Michael Sedlmair, Petra Isenberg, Miriah Meyer, and Tobias Isenberg (Eds.). IEEE Computer Society, 86–92. https://doi.org/10.1109/BELIV.2018.8634297

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

[48] A. Satyanarayan and J. Heer. 2014. Lyra: An interactive visualization design environment. *Computer Graphics Forum (Proceedings of EuroVis)* 33, 3 (2014). https://doi.org/10.1111/cgf.12391

[49] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)* 23, 1 (2017), 341–350. https://doi.org/10.1109/TVCG.2016.2599030

[50] Danqing Shi, Yang Shi, Xinyue Xu, Nan Chen, Siwei Fu, Hongjin Wu, and Nan Cao. 2019. Task-oriented optimal sequencing of visualization charts. In *2019 IEEE Visualization in Data Science (VDS)*. IEEE, 58–66.

[51] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Query, analysis, and visualization of hierarchically structured data using Polaris. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. ACM, 112–122. https://doi.org/10.1145/775047.775064

[52] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The Metacognitive Demands and Opportunities of Generative AI. In *Proceedings of the CHI Conference on Human Factors in Computing Systems, CHI 2024, Honolulu, HI, USA, May 11-16, 2024*, Florian 'Floyd' Mueller, Penny Kyburz, Julie R. Williamson, Corina Sas, Max L. Wilson, Phoebe O. Toups Dugas, and Irina Shklovski (Eds.). ACM, 680:1–680:24. https://doi.org/10.1145/3613904.3642902

[53] Yuan Tian, Weiwei Cui, Dazhen Deng, Xinjing Yi, Yurun Yang, Haidong Zhang, and Yingcai Wu. 2024. Chartgpt: Leveraging llms to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics* (2024).

[54] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[55] 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.

[56] Jacob VanderPlas, Brian E. Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: Interactive Statistical Visualizations for Python. *J. Open Source Softw.* 3, 32, 1057. https://doi.org/10.21105/joss.01057

[57] Chenglong Wang, Yu Feng, Rastislav Bodík, Isil Dillig, Alvin Cheung, and Amy J. Ko. 2021. Falx: Synthesis-Powered Visualization Authoring. In *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, Yoshifumi Kitamura, Aaron Quigley, Katherine Isbister, Takeo Igarashi, Pernille Bjørn, and Steven Mark Drucker (Eds.). ACM, 106:1–106:15. https://doi.org/10.1145/3411764.3445249

[58] Chenglong Wang, John Thompson, and Bongshin Lee. 2023. Data Formulator: Ai-powered concept-driven visualization authoring. *IEEE Transactions on Visualization and Computer Graphics* (2023).

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

[60] Ryen W White and Steven M Drucker. 2007. Investigating behavioral variability in web search. In *Proceedings of the 16th international conference on World Wide Web*. 21–30.

[61] Hadley Wickham. 2009. *ggplot2 - Elegant Graphics for Data Analysis*. Springer. https://doi.org/10.1007/978-0-387-98141-3

[62] Hadley Wickham. 2014. Tidy data. *The Journal of Statistical Software* 59 (2014). Issue 10. http://www.jstatsoft.org/v59/i10/

[63] Hadley Wickham, Mara Averick, Jennifer Bryan, Winston Chang, Lucy McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, Max Kuhn, Thomas Pedersen, Evan Miller, Stephan Bache, Kirill Müller, Jeroen Ooms, David Robinson, Dana Seidel, Vitalie Spinu, Kohske Takahashi, Davis Vaughan, Claus Wilke, Kara Woo, and Hiroaki Yutani. 2019. Welcome to the tidyverse. *J. Open Source Softw.* 4, 43 (Nov. 2019), 1686. https://doi.org/10.21105/joss.01686

[64] Leland Wilkinson. 2005. *The Grammar of Graphics, Second Edition*. Springer.

[65] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 2648–2659. https://doi.org/10.1145/3025453.3025768

[66] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155* (2023).

[67] Zhengkai Wu, Vu Le, Ashish Tiwari, Sumit Gulwani, Arjun Radhakrishna, Ivan Radiček, Gustavo Soares, Xinyu Wang, Zhenwen Li, and Tao Xie. 2022. NL2Viz: natural language to visualization via constrained syntax-guided synthesis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 972–983.

[68] Kai Xiong, Siwei Fu, Guoming Ding, Zhongsu Luo, Rong Yu, Wei Chen, Hujun Bao, and Yingcai Wu. 2022. Visualizing the scripts of data wrangling with SOMNUS. *IEEE Transactions on Visualization and Computer Graphics* (2022).

[69] 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.

[70] Qingru Zhang, Chandan Singh, Liyuan Liu, Xiaodong Liu, Bin Yu, Jianfeng Gao, and Tuo Zhao. 2023. Tell your model where to attend: Post-hoc attention steering for llms. *arXiv preprint arXiv:2311.02262* (2023).

[71] Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. 2024. Training Language Model Agents without Modifying Language Models. *ICML'24* (2024).

[72] Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. OpenCodeInterpreter: Integrating Code Generation with Execution and Refinement. *arXiv preprint arXiv:2402.14658* (2024).