# DirectVis: Editing Code-Based Interactive Visualization with Direct Manipulation

Jeongin Park* Mingyu An† Hyunseo Yang† Junhyeong Hwangbo*
Min Hyeong Kim* Hyeon Jeon* Jinwook Seo‡

Seoul National University

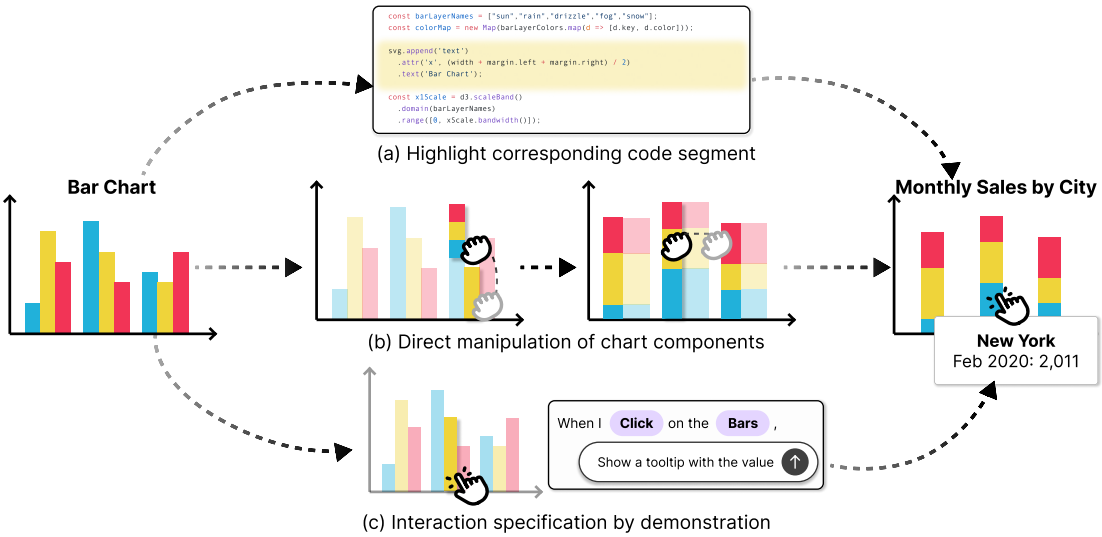


Figure 1: Overview of DirectVis. (a) Selecting chart elements highlights corresponding code segments. (b) Users directly manipulate chart components (e.g., dragging and resizing bars) to refine values and transform structures, with synchronized code updates. (c) Users specify interactive behaviors (e.g., tooltips) by demonstrating them directly on the visualization.

## Abstract

Visualization authoring novices often rely on large language model (LLM)-supported tools for code-based chart editing. However, it can be difficult for novices to precisely articulate what to change and how in natural language, which can make chart editing inefficient; e.g., requiring multiple rounds of iterative revisions. To address this problem, we propose DirectVis, a visualization editing system that integrates direct manipulation with code-based editing. Using DirectVis, users can clarify how a visualization should be edited by directly manipulating chart components. This interaction can be interpreted as users' demonstration of their intended edits to the system. A user study verifies that our system facilitates the precise delivery of user intent, thereby improving the efficiency of chart editing compared to natural-language-based systems. Based on our findings, we derive design implications for future visualization editing tools.

**Index Terms:** Visualization editing, direct manipulation

## 1 Introduction

Code-based visualization frameworks [2, 19, 21] provide the expressivity and flexibility required to develop highly interactive and customizable charts. Large language models (LLMs) have democratized this process by allowing users to specify high-level requirements in natural language, streamlining tasks such as specifying chart types and mapping data to visual channels.

However, iterative refinement remains a major bottleneck when editing existing visualizations. Users should ground their editing intent—what to change and where—into either code or natural language. Yet LLM-generated code lacks explicit correspondence to visual elements, making it difficult to identify which segments control target chart components. Meanwhile, natural language is often ambiguous for precise, fine-grained adjustments. Furthermore, LLM-based editing provides little immediate visual feedback, forcing users to mentally simulate code execution. This leads to unintended outcomes and inefficient trial-and-error cycles.

To address these challenges, we propose DirectVis, a system that integrates direct manipulation into code-based visualization editing. DirectVis renders chart components as interactive objects, enabling users to click, resize, and drag them. The system interprets these interactions as chart modifications and provides immediate visual feedback.

Users can select any chart element by clicking on it to immediately highlight its corresponding code snippet, enabling users to perform targeted manual code edits. Besides interactive selection, the system supports intuitive reconfiguration. For example, dragging bars onto other bars in a grouped bar chart converts it to a stacked chart, while resizing a bar's height automatically recalibrates the y-axis scale. These direct manipulations update the underlying code while preserving visualization integrity. Users can also specify interactions by demonstration rather than textual description. DirectVis parses demonstrated actions into structured in-

*e-mail: {jipark, hwangbo, mhkim, hj}@hcil.snu.ac.kr
†e-mail: {mgahn0706, yanghs1018}@snu.ac.kr
‡e-mail: jseo@snu.ac.kr, corresponding author

teraction intents (e.g., hovering over bars) and allows users to specify the system response in natural language. By providing immediate visual feedback for every interaction, DirectVis keeps users aware of changes, eliminates the need for mental code simulation, and streamlines the refinement process.

We evaluated DirectVis through a user study comparing it to a baseline natural language prompting interface. Participants were asked to refine an initial chart with code to match specific target designs. Participants achieved higher task completion rates with reduced completion times and fewer iterations. These findings confirm that DirectVis enables more effective and efficient refinements than natural language prompting alone.

In summary, we make three contributions: (1) DirectVis, a system that integrates direct manipulation for editing code-based visualizations; (2) an evaluation demonstrating that DirectVis effectively conveys user intent and improves efficiency compared to natural-language prompting; and (3) design implications for combining generative AI with direct manipulation in visualization editing systems.

## 2 Related Work

In this section, we review related work in three areas: code-based visualization authoring, direct manipulation interfaces, and natural language-based visualization systems.

### 2.1 Code-Based visualization authoring tools

Code-based visualization authoring is largely grounded in the grammar-of-graphics paradigm, which models visualizations as compositions of data transformations and visual encodings [31]. Low-level libraries such as D3 [2] expose expressive graphical primitives that enable highly customized visualizations, however, they require users to manually manage many implementation and styling decisions [1]. Declarative grammars such as Vega raise the level of abstraction by letting users specify visual intent through structured specifications compiled into rendering pipelines [19, 20, 21, 7]. Subsequent work proposed higher-level, code-centric workflows to better support authoring and refinement in programmatic pipelines [32, 34].

Despite these advances, post-hoc refinement in code-based workflows often remains cumbersome: users must locate and edit the right variables inside nested specification structures, and debugging can be difficult when the mapping between specification and rendered output is not transparent. Our work targets this refinement gap by enabling direct manipulation of visualization properties while preserving the expressiveness and flexibility of code-based authoring.

### 2.2 Direct manipulation for visualization authoring

Direct manipulation [24] reduces the gulf of execution [14] by allowing users to interactively bind data to visual marks without writing code [18]. These tools range from shelf-based systems that use drag-and-drop metaphors for mapping data variables to visual channels [10, 12, 29] to systems supporting interaction authoring by demonstration and bespoke layout construction through interactive constraint-based design [16, 33]. Beyond authoring tools, prior work has examined direct manipulation of graphical encodings [17], spatial constraints for manipulating static visual outputs [9], and dynamically interactive visualizations derived from static charts [26]. In contrast, our work integrates direct manipulation with LLM-supported natural language editing to refine the visualization source code.

### 2.3 Natural language interfaces for visualization

Natural Language Interfaces (NLI) seek to democratize visualization authoring by translating user utterances into formal analytic specifications and visual outputs [5, 13, 22, 23, 27, 28]. Recent advancements in LLMs have further accelerated this process, enabling the rapid generation of complex visualization code from high-level descriptions [4, 11, 25, 30].

While LLMs excel at initial generation, iterative refinement remains difficult due to (1) unclear code-to-visual mappings that obscure what to change, (2) the inherent ambiguity of language for precise adjustments, and (3) a lack of immediate feedback. To bridge these gaps, our approach integrates direct manipulation, direct code editing, and LLM-based modification. This hybrid approach enables users to choose the most effective modality for refinement, providing both the precision of code and the intuitiveness of interactive editing.

## 3 DirectVis

In this section, we present DirectVis, a code-based visualization authoring system that integrates direct manipulation with LLM-powered editing to address the challenges of iterative refinement.

### 3.1 Design goal

DirectVis is designed around three goals that address the key limitations of existing LLM-based visualization tools that we identified in Section 2.3.

**DG1. Provide correspondence between code and visualization** to allow users to identify which parts of the code control specific visual elements.

**DG2. Support direct manipulation for intuitive editing** by enabling users to directly adjust visualization components and author interactions through demonstration.

**DG3. Provide immediate feedback on expected visualization outcomes** to help users understand the anticipated effects of an edit before execution, thereby reducing reliance on mental prediction.

### 3.2 DirectVis

DirectVis consists of four components: a chart canvas, a code editor, a natural language prompt input, and a demonstration-based interaction specification module. We describe how each component is designed to support our design goals.

#### 3.2.1 Chart canvas

A chart canvas (Fig 2 (a)) serves as a primary chart builder for direct manipulation. Visualizations are rendered as SVG elements, allowing interactions to individual components such as marks, axes, and labels.

DirectVis supports hover, click, drag, zoom, pan, and brush interactions implemented at the SVG element level and translated into structured event-handler definitions in the underlying code. Users can click on an SVG element to select it. Each rendered element is assigned a unique identifier during D3-based rendering, and this identifier is maintained within the chart state representation. Upon selection, the system resolves its identifier through the chart state and regenerates the corresponding code segment for highlighting in the editor. Conversely, selecting a code block references the same state structure to identify and highlight its associated SVG elements. This state-mediated synchronization enables bidirectional correspondence between code and visualization without requiring static code analysis (DG1).

Selected visual components can be directly manipulated through dragging and resizing (DG2). Supported operations include drag-and-drop of marks (with re-grouping and stacking), resizing of visual elements, and scale adjustments via zoom and pan gestures. During these interactions, the system provides a preview of the expected outcome, enabling users to see how their actions would affect their visualization (DG3).

#### 3.2.2 Code editor

Since DirectVis targets users who edit visualizations through code, the system provides a code editor panel (Fig. 2b) that allows users to edit code directly as in a conventional editor. Each logical code block is derived from the current chart state rather than parsed from user-written source code. Because both the visualization and code are generated from the same structured state representation, selecting a code block or visual element resolves to the same underlying state node, guaranteeing consistency between visual edits, LLM-driven modifications, and regenerated code (DG1). When users click a code block, the corresponding SVG element is automatically highlighted, making the bidirectional connection between code and visualization intuitive (DG1).

#### 3.2.3 Prompt input

A prompt input (Fig 2 (c)) enables high-level specification with natural language to complement direct manipulation and code editing. While direct manipulation excels at perceptual adjustments and code editing supports precise logical control, natural-language prompting allows users to express high-level intents, constraints, or transformations that are difficult to achieve through direct manipulation or code-editing. To address the ambiguity of natural language in prior LLM-based systems, the prompt input section in DirectVis allows component-wise prompting (DG2).

#### 3.2.4 Interaction specification module

Specifying interactive behaviors in visualization code requires users to precisely identify event targets and handlers, which is challenging without clear code–visual correspondence. DirectVis addresses this by enabling interaction specification through direct demonstration on the chart canvas. User actions such as hovering or clicking are captured and translated into explicit action–target representations that are immediately displayed. This grounded, feedback-driven approach reduces ambiguity in interaction specification and lowers the barrier to authoring interactive visualizations (Fig. 2d).

### 3.3 Implementation

DirectVis is built on React and D3.js: React provides component-based UI management, while D3.js enables expressive visualization authoring through low-level SVG manipulation. DirectVis is implemented and evaluated on bar and line charts, chosen as a controlled domain due to their prevalence and clear data-to-encoding mappings for studying semantic grounding between direct manipulation and code transformations. The core mechanism is not chart-specific: DirectVis operates at the SVG level and maintains explicit mark-to-code correspondence via D3 rendering, enabling the bidirectional transformation pipeline to generalize to other visualization types as long as marks are data-bound and mapped to identifiable code blocks. Moreover, the interaction-to-code translation approach is library-agnostic and extendable to frameworks such as Matplotlib [7] and Vega-Lite [19]. For the prompt input, we employed GPT-4o as the backend LLM, as it demonstrates strong performance and high accuracy in editing D3-based visualization code.

## 4 User Study

We conducted a user study to evaluate whether DirectVis effectively supports visualization editing. We employed a within-subjects design comparing DirectVis with a baseline condition that supported only natural-language prompting for code-based visualization editing. Specifically, we investigated the following research questions:
**RQ1.** Does DirectVis effectively convey user intent during visualization editing?
**RQ2.** Does DirectVis improve the efficiency of visualization editing tasks?
**RQ3.** How do DirectVis's features support users during visualization editing?

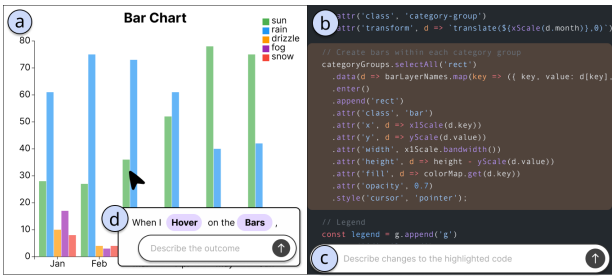


Figure 2: DirectVis interface. (a) Chart canvas for direct manipulation of SVG elements. (b) Code editor with clickable code blocks. (c) Prompt input for natural language specifications. (d) Interaction specification by demonstration.

### 4.1 Participants

We recruited 12 participants (age 23–25; 10 male, 2 female). All had prior coding experience: 6 with 5+ years, 4 with 3–4 years, and 2 with 1–2 years. Self-reported visualization experience (5-point Likert scale) averaged 2.6 (SD = 1.1, range 1–4). Four participants had prior D3.js experience; eight had none. We targeted individuals with coding expertise but limited visualization experience, as this population represents the primary user base for LLM-assisted visualization tools—developers who can understand and modify code but lack specialized knowledge of visualization libraries and design principles. Sessions lasted approximately 60 minutes, and participants received 10 USD compensation.

### 4.2 Tasks

Tasks were designed to reflect common visualization editing scenarios in which users adjust existing visualizations to more effectively convey analytical intent, including view faceting and interaction authoring. Each task consisted of two subtasks: Subtask 1 with chart type transformation and Subtask 2 with interaction authoring (Fig. 3). Given initial code and charts, participants reproduced target visualizations and interactions demonstrated through images and videos instead of textual descriptions, to prevent participants from directly translating provided text into prompts.
**Task 1.** Participants transformed a grouped bar chart into a stacked bar chart (Subtask 1), then implemented a click interaction to dim non-selected bars (Subtask 2).
**Task 2.** Participants transformed a multi-line chart into small multiples (Subtask 1), then implemented a hover interaction to display tooltips (Subtask 2).

### 4.3 Procedure

The study was conducted remotely via Zoom or in person. After obtaining consent, we provided a tutorial on both systems, introducing available interaction techniques without revealing specific task solutions. Participants then freely explored the systems until they felt comfortable with the interface.

Each participant used both systems and completed one task per system, with each task comprising two subtasks. System order and task assignment were randomized and counterbalanced across participants. Participants had up to 20 minutes to complete each task. After completing tasks with each system, participants completed a post-task survey. Following both conditions, we conducted semi-structured interviews about interaction preferences and overall experience. We logged all user interactions and recorded screen and audio throughout each session.

### 4.4 Measurements and analysis

We measured user experience via post-task surveys (7-point Likert scales, Table 1), which were analyzed with Wilcoxon signed-rank tests. Task completion was recorded at the subtask level; subtasks


Figure 3: Study tasks. (a) Task 1: Bar chart transformation and interaction, (b) Task 2: Line chart transformation and interaction. Each task shows starter chart and target outcomes for each subtasks.

Table 1: Post-task survey questions and summary labels.

| Label | Survey Question |
|---|---|
| Speed | The system allowed me to complete tasks quickly. |
| Intent Reflection | The system accurately reflected my intended modifications in the results. |
| Ease of Manipulation | It was easy to select or manipulate the desired visual elements. |
| Cognitive Load | Using the system required low mental effort. |
| Ease of Use | Overall, the system was easy to use. |
| Future Usage | I would be willing to use this system for real-world tasks in the future. |

were marked as failed if incomplete after a time limit of 20 minutes or if the user gave up. Completion times for successful attempts are reported as means and standard deviations. We do not report statistical comparisons for completion times due to missing data from failures. From interaction logs, we extracted and counted editing attempts, iterations, and modality usage (natural language commands, code editing, and direct manipulation), comparing conditions using Wilcoxon signed-rank tests. Qualitative analysis involved open coding of interviews by the first author, refined iteratively with one co-author.

## 5 Results

We present our user study results comparing DirectVis and the baseline.

### 5.1 Survey results

Figure 4 summarizes the post-task survey results. Participants rated DirectVis significantly higher than the baseline for Speed ($p < .05$), Ease of Manipulation ($p < .01$), and Ease of Use ($p < .05$). Ratings for Intent Reflection, Cognitive Load, and Future Use showed higher mean scores for DirectVis but did not reach statistical significance.

### 5.2 Task completion

Task completion was evaluated using task success rate and task completion time.

**Task success rate.** For Subtask 1, 10 participants successfully completed the task with the baseline system, whereas all 12 participants successfully completed the task using DirectVis. For Subtask 2, 10 participants successfully completed the task using the baseline

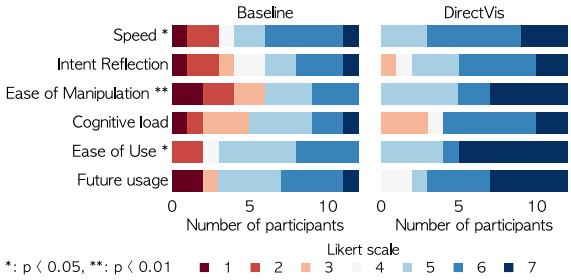


Figure 4: Post-task Likert survey results. All items were rated on a 7-point Likert scale (1 = Strongly disagree, 7 = Strongly agree).

system, while 9 participants successfully completed the task using DirectVis.

**Task completion time.** Task completion time was measured only for successfully completed trials. DirectVis showed faster completion for Subtask 1(Baseline: M = 460.2, SD = 227.2 seconds; DirectVis: M = 371.2, SD = 243.7 seconds), but slightly slower for Subtask 2(Baseline: M = 302.3, SD = 159.0 seconds; DirectVis: M = 328.2, SD = 186.7 seconds).

### 5.3 User behavior

We analyzed user behavior to examine how participants utilized different interaction modalities across systems and subtasks.

#### 5.3.1 Interaction frequency

**Total interactions.** We counted the total number of interactions, defined as the sum of natural language prompts, code edits, direct manipulation actions, and interaction specification actions. Overall, participants performed fewer interactions when using DirectVis compared to the baseline, although this difference did not reach statistical significance (Baseline: M = 17.58, SD = 11.29; DirectVis: M = 11.92, SD = 5.55, W = 11.50, p = .055). For Subtask 1, DirectVis required significantly fewer interactions than the baseline (Baseline: M = 12.58, SD = 10.82; DirectVis: M = 7.50, SD = 4.68; W = 8.00, $p < .05$). In contrast, for Subtask 2, the number of interactions was comparable between the two systems (Baseline: M = 5.00, SD = 4.63; DirectVis: M = 4.42, SD = 3.06; W = 28.50, p = .688).

**Natural language prompts.** Participants relied heavily on natural language prompts in the baseline condition, whereas their usage was substantially lower in DirectVis. Notably, two participants (P2, P6) did not use natural language prompts at all in the DirectVis condition, while all participants used prompts in the baseline condition. Overall, prompt usage was significantly lower in DirectVis than in the baseline (Baseline: M = 11.25, SD = 5.79; DirectVis: M = 3.67, SD = 2.81; W = 5.00, $p < .01$). This difference was observed in both subtasks, including Subtask 1 (Baseline: M = 7.50, SD = 4.76; DirectVis: M = 2.67, SD = 1.78; W = 9.50, $p < .05$) and Subtask 2 (Baseline: M = 3.75, SD = 2.42; DirectVis: M = 1.00, SD = 1.81; W = 4.00, $p < .05$).

**Code editing.** Code editing was used less frequently overall than natural language prompting. Three participants (P1, P3, P12) did not use code editing at all in the DirectVis condition, while two participants (P10, P12) did not use code editing in the baseline condition. Overall, participants performed significantly fewer code edits with DirectVis compared to the baseline (Baseline: M = 6.33, SD = 7.28; DirectVis: M = 3.75, SD = 4.16; W = 11.00, $p < .05$). At the subtask level, differences were not significant for Subtask 1 (Baseline: M = 5.08, SD = 7.30; DirectVis: M = 3.17, SD = 4.34; W = 21.50, p = .301) or Subtask 2 (Baseline: M = 1.25, SD = 2.93; DirectVis: M = 0.58, SD = 1.16; W = 4.00, p = .715).

**Direct manipulations and interaction specification actions.** Direct manipulation and interaction specification were available only

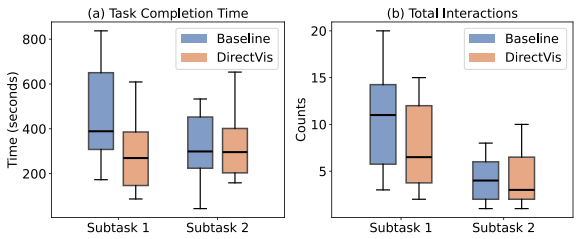


Figure 5: (a) Task completion time and (b) total interaction iterations by system across subtasks.

in the DirectVis condition. For Subtask 1, all participants except P5 and P11 utilized direct manipulation at least once (M = 1.67, SD = 1.23). Direct manipulation was not used in Subtask 2; participants instead relied on interaction specification (M = 2.83, SD = 2.86).

#### 5.3.2 User Strategies

We analyzed post-study interviews to characterize participants' use of different interactions.

**Direct manipulation as a primary strategy.** Participants found direct manipulation intuitive and predictable (P2, P3, P7, P8, P10, P11, P12). DirectVis enabled them to precisely specify intended modifications (P2, P4, P6, P11), while the preview feature allowed them to anticipate outcomes before execution (P2, P5, P6, P7, P8).

**Code editing as a secondary or fallback strategy.** Code editing was used selectively and primarily as a secondary or fallback strategy. Code editing was mainly applied to simple or fine-grained adjustments, such as modifying text labels or making minor positional changes (P2, P4, P7, P8, P9, P10, P11). A few participants reported turning to code editing only when both natural language prompting and direct manipulation failed to produce the desired outcome (P3), whereas others stated that they used code minimally or preferred to avoid it whenever possible (P6). Participants also highlighted the usefulness of the code highlighting feature, which enabled quick access to relevant code segments and reduced the effort required to locate where edits should be made (P4, P6, P9, P10, P11).

**Natural language prompting for expressive or non-direct actions.** With DirectVis, prompting was used more selectively, with some participants not using it at all (P2, P6). Participants reduced prompting because direct manipulation was sufficiently effective for many tasks (P2). When used, prompting typically addressed adjustments difficult to achieve through direct manipulation, such as fine-grained edits (P3), requirements more naturally expressed verbally (P7, P11), or semantic changes like reordering colors (P12).

#### 5.3.3 Natural language prompting strategy

We analyzed participants' natural language prompts and identified three common patterns.

**[Current visual state] + [Desired outcome]** Participants often grounded requests by describing the current visualization before specifying changes. For example: *"Currently, all legend values are shown separately for each month, but I want to convert this into a stacked chart aggregated by month"* or *"Each category in each month is shown horizontally, but I want to change it to be shown vertically."*

**[Target component] + [Action]** Many prompts directly specified a target and action without restating context, such as *"Reduce the width of each bar,"*, or *"Change the chart title to 'Seattle Weather by Month (Jan to Jun),"*

**Constraints** Some prompts included explicit constraints to prevent unintended changes, such as *"Do not change the attribute IDs"* or *"Keep the x-axis unchanged."*

Across both systems, these patterns reflect participants' attempts to reduce ambiguity in natural language prompts by explicitly grounding requests in visual states, components, and constraints.

## 6 Discussion

We discuss design implications for future visualization editing systems, considering both the strengths and limitations of DirectVis.

### 6.1 Design Implications

**Support bidirectional code–output mapping.** Visualization authoring systems should support bidirectional mappings between code and visual output, enabling users to both generate visualizations from code and trace visual elements back to their source code. Prior work on output-directed programming has demonstrated the feasibility of propagating output changes to program updates [6]. Bidirectional mappings can also serve as a code localization mechanism, allowing users to quickly identify which code segments produce specific visual elements. Since visualization refinement typically involves incremental edits and debugging rather than complete rewrites, systems exposing these correspondences can reduce search effort and enable more efficient, targeted modifications.

**Enable interaction-based authoring beyond natural language.** Natural language specifications require explicit verbalization of visualization state, targets, and outcomes, introducing ambiguity and repeated iterations. Prior work has shown that augmenting natural language with interaction [3, 8] reduces such ambiguity. DirectVis demonstrates this approach by enabling users to specify intent through direct manipulation and code selection, reducing reliance on verbose descriptions. Future visualization authoring systems should similarly support interaction-based authoring to mitigate failures from underspecified prompts.

**Design intuitive interactions with immediate feedback.** Effective interaction-based authoring relies on intuitive interaction design coupled with immediate feedback, a core principle of direct manipulation [24]. When interactions align with users' mental models, available actions become discoverable through exploration rather than instruction [14]. Immediate visual feedback, such as previews of interaction outcomes, supports this process by allowing users to anticipate the effects of their actions. These properties reduce cognitive load and interaction failures, suggesting that visualization authoring systems should prioritize discoverable interactions and rapid feedback to support efficient learning and use.

### 6.2 Limitations and future work

We found no significant differences for interaction authoring tasks, possibly because tasks involved familiar interactions (hovering, clicking) accessible to novices. Future work could examine specialized interactions (zooming, filtering) and explore outcome specification methods beyond natural language.

DirectVis supports fundamental operations (drag-and-drop, resizing) for common charts (bar, line). Complex visualizations (networks, hierarchies, maps) and abstract operations (statistical transformations, aggregations) may not map to spatial manipulations. Future work should investigate which types and tasks benefit most from direct manipulation versus alternative modalities. Although our current system exclusively supports D3.js [2], the core ideas of DirectVis can be extended to other visualization authoring libraries [7, 19] through systematic modifications such as reverse-engineering methods [15].

## 7 Conclusion

While LLMs make code-based visualization generation accessible, iterative refinement remains challenging due to unclear code-to-visual mappings, ambiguous instructions, and lack of immediate feedback. We address this by integrating direct manipulation with LLM-based editing, enabling explicit mappings and immediate visual feedback. Our user study indicates that this approach supports more effective and efficient refinement compared to natural language prompting, highlighting the benefits of combining generative AI with direct manipulation for data visualization authoring.

## Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2023R1A2C200520911) and the SNU-Global Excellence Research Center establishment project. This work was also supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)]. The ICT at Seoul National University provided research facilities for this study.

## References

[1] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1121–1128, Nov. 2009. doi: 10.1109/TVCG.2009.174 2

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/TVCG.2011.185 1, 2, 5

[3] J. J. Y. Chung and E. Adar. Promptpaint: Steering text-to-image generation through paint medium-like interactions. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, 2023. doi: 10.1145/3586183.3606777 5

[4] V. Dibia. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models, 2023. doi: 10.48550/arXiv.2303.02927 2

[5] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, p. 489–500, 2015. doi: 10.1145/2807442.2807478 2

[6] B. Hempel, J. Lubin, and R. Chugh. Sketch-n-sketch: Output-directed programming for svg. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, p. 281–292, 2019. doi: 10.1145/3332165.3347925 5

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55 2, 3, 5

[8] P. Jiang, J. Rayan, S. P. Dow, and H. Xia. Graphologue: Exploring large language model responses with interactive diagrams. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, 2023. doi: 10.1145/3586183.3606737 5

[9] C. Liu, Y. Zhang, C. Wu, C. Li, and X. Yuan. A spatial constraint model for manipulating static visualizations. *ACM Trans. Interact. Intell. Syst.*, 14(2), June 2024. doi: 10.1145/3657642 2

[10] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, p. 1–13, 2018. doi: 10.1145/3173574.3173697 2

[11] P. Maddigan and T. Susnjak. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *IEEE Access*, 11:45181–45193, 2023. doi: 10.1109/ACCESS.2023.3274199 2

[12] Microsoft. Microsoft Power BI. Accessed January 6, 2026. 2

[13] A. Narechania, A. Srinivasan, and J. Stasko. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379, 2021. doi: 10.1109/TVCG.2020.3030378 2

[14] D. Norman. Design of everyday things, 1988. 2, 5

[15] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. *Computer Graphics Forum*, 36(3):353–363, 2017. doi: 10.1111/cgf.13193 5

[16] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):789–799, 2019. doi: 10.1109/TVCG.2018.2865158 2

[17] B. Saket, S. Huron, C. Perin, and A. Endert. Investigating direct manipulation of graphical encodings as a method for user interaction. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):482–491, 2020. doi: 10.1109/TVCG.2019.2934534 2

[18] A. Satyanarayan and J. Heer. Lyra: an interactive visualization design environment. In *Proceedings of the 16th Eurographics Conference on Visualization*, EuroVis '14, p. 351–360. Eurographics Association, 2014. doi: 10.1111/cgf.12391 2

[19] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vegalite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, Jan. 2017. doi: 10.1109/TVCG.2016.2599030 1, 2, 3, 5

[20] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):659–668, 2016. doi: 10.1109/TVCG.2015.2467091 2

[21] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. Declarative interaction design for data visualization. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, p. 669–678, 2014. doi: 10.1145/2642918.2647360 1, 2

[22] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, p. 365–377, 2016. doi: 10.1145/2984511.2984588 2

[23] L. Shen, E. Shen, Y. Luo, X. Yang, X. Hu, X. Zhang, Z. Tai, and J. Wang. Towards natural language interfaces for data visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 29(6):3121–3144, 2023. doi: 10.1109/TVCG.2022.3148007 2

[24] B. Shneiderman. Direct manipulation: A step beyond programming languages (abstract only). *SIGSOC Bull.*, 13(2–3):143, May 1981. doi: 10.1145/1015579.810991 2, 5

[25] Z. Shuai, B. Li, S. Yan, Y. Luo, and W. Yang. Deepvis: Bridging natural language and data visualization through step-wise reasoning. *IEEE Transactions on Visualization and Computer Graphics*, p. 1–11, 2025. doi: 10.1109/tvcg.2025.3634645 2

[26] L. S. Snyder and J. Heer. Divi: Dynamically interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):403–413, 2024. doi: 10.1109/TVCG.2023.3327172 2

[27] A. Srinivasan and J. Stasko. Natural language interfaces for data analysis with visualization: considering what has and could be asked. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers*, EuroVis '17, p. 55–59. Eurographics Association, 2017. doi: 10.2312/eurovisshort.20171133 2

[28] Y. Sun, J. Leigh, A. Johnson, and S. Lee. Articulate: A semi-automated model for translating natural language queries into meaningful visualizations. In R. Taylor, P. Boulanger, A. Krüger, and P. Olivier, eds., *Smart Graphics*, pp. 184–195. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-13544-6_18 2

[29] Tableau Software. Tableau: Business Intelligence and Analytics Software. https://www.tableau.com/. Accessed January 6, 2026. 2

[30] Y. Tian, W. Cui, D. Deng, X. Yi, Y. Yang, H. Zhang, and Y. Wu. Chartgpt: Leveraging llms to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics*, 31(3):1731–1745, Mar. 2025. doi: 10.1109/TVCG.2024.3368621 2

[31] H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010. doi: 10.1198/jcgs.2009.07098 2

[32] J. Yang, A. Bäuerle, D. Moritz, and c. Demiralp. Vegaprof: Profiling vega visualizations. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, 2023. doi: 10.1145/3586183.3606790 2

[33] J. Zong, D. Barnwal, R. Neogy, and A. Satyanarayan. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):304–314, 2021. doi: 10.1109/TVCG.2020.3030367 2

[34] J. Zong, J. Pollock, D. Wootton, and A. Satyanarayan. Animated vegalite: Unifying animation with a grammar of interactive graphics, 2022. doi: 10.48550/arXiv.2208.03869 2