Bavisitter: Integrating Design Guidelines into Large Language Models for Visualization Authoring

Jiwon Choi*
Sungkyunkwan University

Jaeung Lee*
Sungkyunkwan University

Jaemin Jo*†
Sungkyunkwan University

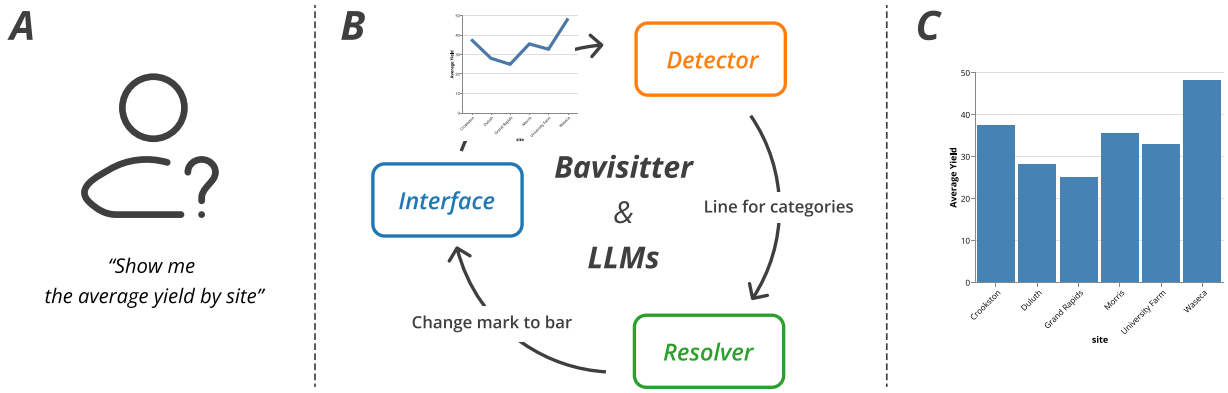

Figure 1: Bavisitter's visualization authoring workflow. A) The user requests a visualization to an LLM by prompting *"Show me the average yield by site."* B) The LLM generates an ineffective visualization design that uses a connection mark for encoding the categorical attribute on the x axis. C) Bavisitter detects the design issue in the generated visualization and gives feedback to the LLM by modifying the original prompt, e.g., appending *"Change mark to bar"*. As a result, the user can author visualization designs that conform to known design guidelines and knowledge while exploiting the flexibility that the LLM provides.

Abstract

Large Language Models (LLMs) have demonstrated remarkable versatility in visualization authoring, but often generate suboptimal designs that are invalid or fail to adhere to design guidelines for effective visualization. We present Bavisitter, a natural language interface that integrates established visualization design guidelines into LLMs. Based on our survey on the design issues in LLM-generated visualizations, Bavisitter monitors the generated visualizations during a visualization authoring dialogue to detect an issue. When an issue is detected, it intervenes in the dialogue, suggesting possible solutions to the issue by modifying the prompts. We also demonstrate two use cases where Bavisitter detects and resolves design issues from the actual LLM-generated visualizations.

Index Terms: Automated Visualization, Visualization Tools, Large Language Model.

1 Introduction

Natural language interfaces (NLIs) for visualization authoring [14, 22, 24, 28, 29, 32] allow the user to create a visualization by articulating their intents in natural language, lowering the visual mapping barrier [9]; the user can author a visualization without strong knowledge on visualization design. Recently, with the advance of large language models (LLMs), a few attempts have been initiated to empower those interfaces with more flexibility and expressiveness by adopting LLMs [7, 19]. In contrast to previous rule-based tokenization and parsing approaches, LLMs provide better flexibility in interpreting the user intent that is not templated in the rules, and they are more expressive in that they can generate codes directly, e.g., Python scripts or Vega-Lite specifications [27], instead of resorting to pre-defined chart templates.

However, we witness that these automated and unsupervised uses of LLMs in visualization authoring often result in suboptimal or invalid visualization designs in terms of design knowledge and guidelines. For example, Figure 2 shows two charts generated by GPT-4. On the left, although the `state` attribute on the x axis is nominal, the chart uses a connection mark for categorical values, raising an expressiveness issue. On the right, there is a single-column heatmap, which could be improved by 1) using the more effective position channel (i.e., a bar chart) and 2) using a linear color scale instead of a diverging one. We discovered that these subtle issues are often missed by previous visualization linters [4, 21].

To address such design issues in LLM-generated visualizations, we present Bavisitter, a natural language visualization authoring interface with a feedback process. Bavisitter aims to respect the previous heuristics and guidelines for designing effective visualizations while keeping the flexibility and expressiveness of LLM-based visualization authoring. The visualization authoring workflow in Bavisitter consists of three phases: prompting, detecting, and resolving. In the prompting phase, the user requests a visualization to an LLM, i.e., GPT-4, in natural language. The LLM generates a visualization specification, e.g., a Vega-Lite specification [27]. Then, in the detecting phase, Bavisitter examines the specification and its rendered image to detect potential issues with respect to the known design knowledge. To resolve the detected issues, in the resolving phase, Bavisitter appends additional natural language instructions as a feedback prompt to the original prompt from the user.

The contributions of this work are two-fold:

- We elucidate the design issues in LLM-generated visualizations by investigating a corpus of 200 visualizations and
- We design a mixed-initiative visualization authoring interface, Bavisitter, which supports the user to author visualizations aligned with visualization design knowledge and guidelines.

*e-mail: {jiwnchoi, dlwodnd00, jmjo}@skku.edu
†Jaemin Jo is the corresponding author.

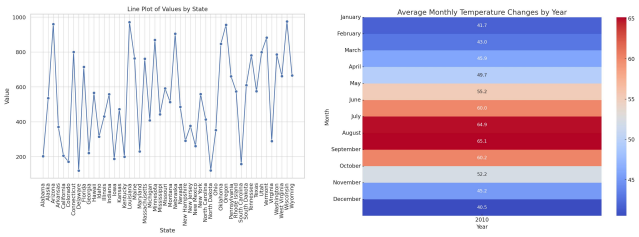

Figure 2: Design issues in visualizations generated by GPT-4. Left: a nominal attribute (state names) is shown on the x axis but a connection mark is used, compromising expressiveness. Right: a quantitative attribute is mapped to the color channel instead of the length channel, making the chart less effective.

2 Related Work

Automated Visualization Authoring and Linting. Researchers have attempted to systematically embed visualization design knowledge and guidelines into the visualization authoring process. For example, Draco [23] represented the visualization design knowledge as explicit constraints to recommend designs that conform to the knowledge. In contrast, there have been deep learning-based systems that took a more implicit approach. For example, Data2Vis [8] and VizML [10] aim to model the visualization knowledge by training deep neural networks on large amounts of human-crafted charts; in these systems, rather than representing the design knowledge as explicit constraints, they are learned as the weights of the networks.

Another line of research aims to diagnose or correct potential issues in visualization, i.e., visualization linter, which was initially proposed by McNutt and Kindlmann [21]. They gathered design guidelines from the web and prototyped a visualization linter that automatically detects issues that violate guidelines. Recently, Chen et al., [4] presented VizLinter that identifies the issues that make chart specifications invalid or unable to render, which were derived by combining Draco [23]'s ruleset and expert interview results. However, we investigated that LLMs create a different kind of problem that is not covered by the above linting systems (Sec. 3.1). To address these issues, we design Bavisitter, a mixed-initiative visualization authoring interface that resolves issues by LLMs with feedback prompts.

Natural Language Interfaces for Visualization Authoring. Natural language interfaces (NLIs) for visualization aim to bridge the gap between humans and visualizations. One group of studies [14, 17, 22, 24, 28, 29, 32] focuses on parsing and interpreting the user's utterance for visualization needs to help them author visualizations without learning design knowledge or guidelines, i.e., producing visualizations. Another group of studies [5, 20, 25, 30] aims to highlight the visualization designer's takeaways and making them more accessible by generating natural language explanations, i.e., consuming visualizations. However, many of these systems resort to heuristics or templates, limiting their adaptability to a broad range of visualization queries.

Large language models (LLM) can be employed to reinforce the flexibility and expressiveness of NLIs for visualization authoring. LIDA [7] and Chat2Vis [19] are two early prototypes that generate visualizations based on the given description on datasets (e.g., column names, cardinalities, data type). While these prototypes demonstrate LLM's capability to understand the user's utterance without pre-defined rules, their uses were limited to analyzing or wrangling data. LLM-based automated agents such as Code Interpreter [26] design visualizations in a human-like way, generating and executing the source code of visualizations. AVA [16] showed that agents with multi-modal large language models could optimize the design by adjusting the visualization's parameters in three scenarios, i.e., volume rendering, scatterplots, and dimensionality reduction. In this work, we show our interface can be used to iteratively integrate design guidelines into LLMs for effective visualization beyond tuning limited parameters.

3 Bavisitter

While large language models (LLMs) have the potential to create data visualizations [12], they often produce suboptimal designs that are not aligned with established design knowledge and guidelines. To bridge such a gap, we introduce an LLM-based visualization authoring system, Bavisitter, which intervenes in the visualization authoring dialogue between the user and LLM to detect and correct design issues.

3.1 Design Issues in LLM-Designed Visualizations

To detect design issues in LLM-generated visualizations, we first surveyed systems that detect and explicate design issues in human-designed visualizations [4, 21], but we found out that these systems often do not capture LLM-generated design issues; LLMs rarely make rendering issues, which existing linting systems [4, 7] had focused on. However, issues like `marks-overplotted` cannot be identified with the aforementioned linters because they examine the specifications, not the rendered visualizations.

To enumerate design issues, we created a corpus of 590 LLM-generated chart images. To this end, we used 59 tabular datasets from the Vega-Datasets repository[1], and we instruct GPT-4 [1] to generate ten random visualizations for each dataset. We then randomly sampled 200 chart images with 24 unique chart types from the corpus. Two of the authors independently annotated the chart images to identify potential design issues in each image. Finally, we conduct a thematic analysis [3] to identify common themes in the annotations and ensure consistency in their groupings.

Table 1 lists the design issues we identified. As a result, we identified 23 issues from 104 improper visualizations among the 200 visualizations. There are 7 visualizations (3.5%) that had rendering issues (i.e., without a mark), which we excluded from the survey. Note that each visualization can contain more than one issue. Then, we identified the issues relevant to the **Expressiveness** and **Effectiveness** principles [18]. We also grouped the remaining issues into **Perception** if it was relevant to visual perception, e.g., `marks-overplotted`, **Interpretability** if it could lead misinterpretation, e.g., `no-zero-in-position`, and **Legibility** if it impaired the legibility of the visualization, e.g., `label-overlap`.

3.2 Visualization Authoring with Feedback Prompts

Recently, several systems have been proposed that give repetitive feedback to make LLMs perform various real-world tasks, such as web browsing [6], coding [34], and research [11]. Inspired by the success of these systems, we view the visualization authoring dialogue as a three-phased workflow consisting of 1) **Prompting Phase**, 2) **Detecting Phase**, and 3) **Resolving Phase**. In this section, we outline each phase and the design choices we made for supporting each in Bavisitter.

In the first **Prompting Phase**, the user requests a visualization as a text prompt, and the LLM generates an output source code, e.g., a Vega-Lite specification [27]. The text prompt from the user can have different levels of specificity; for example, it can have the most details about the requested visualization, e.g., *"Show me the average US Gross by Genre,"* or it can be more ambiguous, e.g., *"Show me an interesting visualization from the given movie dataset."* where such different levels of detail are handled by the LLM. During the **Prompting Phase**, Bavisitter monitors the visualization generated by the LLM to check if there is any design issue in the visualization. Once a design issue is detected, it enters the **Detecting Phase**, intervening in the dialogue.

[1] https://vega.github.io/vega-datasets/

Table 1: List of issues identified in the visualizations generated by GPT-4 [1].

Issue Type	Code of Issue (#)	Description of Issue
Expressiveness	`line-for-categorical` (5)	A connection mark cannot be used for nominal attributes.
	`cont-colors-for-cat` (4) `cat-colors-for-ordered` (31)	The color scheme does not match the attribute type.
Effectiveness	`cardinality-is-one` (6)	Assigning a visual channel to an attribute of cardinality 1 is ineffective.
	`indistinct-theta` (2)	The angles used to encode data are not visually distinguishable.
	`redundant-encoding` (7)	Unnecessary multiple encodings are used to represent the same data.
	`exccessive-cardinaility` (3)	The cardinality of a categorical attribute is too high.
Interpretability	`no-zero-in-position` (3)	The chart does not include zero on positional scales.
	`ordinal-not-sorted` (2)	The ordinal attribute is not sorted properly.
	`layered-bin-mismatch` (1)	Binning is inconsistent across layered histograms.
	`wrong-title` (5)	The title does not accurately reflect the specification.
Legibility	`bin-is-not-nice` (10)	The boundaries of bins are not nice numbers.
	`annotations-unreadable` (3)	Annotations are not distinguishable from marks or backgrounds.
	`small-marks` (2)	Marks or ticks are too small.
	`no-ticks` (1) `no-labels` (1) `no-legend` (7)	Ticks, labels, or legends are missing.
	`annotation-overlap` (3) `label-overlap` (2)	Annotations or labels overlap each other.
Perception	`shapes-with-size` (3)	Shapes other than circles are used to encode size.
	`too-many-bins` (1)	There are too many bins.
	`overplotted-marks` (13)	Marks are overplotted.
	`sparse-mark` (5)	The density of marks is too sparse.

In the next **Detecting Phase**, Bavisitter warns the user of the design issues detected by showing a summary of the issues in the middle of the dialogue (Fig. 3B). To this end, we identified 23 design issues by investigating the 200 visualizations that GPT-4 generated (Tab. 1), which can be grouped into five categories: Expressiveness, Effectiveness, Interpretability, Legibility, and Perception. For example, the `line-for-categorical` issue is raised when a connection mark is used to encode categorical attributes.

In the **Resolving Phase**, Bavisitter recommends possible solutions to the identified design issues. Since there can be multiple design issues or multiple solutions to a single issue, we provide a multiple-select interface to the user where the user can opt for each solution (Figure 3B). Once a solution is chosen, the detected issues and a description of the solution are given to the LLM as a feedback prompt, guiding the LLM to correct the design issue.

For example, suppose the user entered an initial prompt, *"Show me the average US Gross by Genre,"* and the LLM answered a line chart instead of a bar chart. In this case, Bavisitter raises the `line-for-categorical` issue in the **Detecting Phase** and suggests the feedback prompt to *"Current visualization has issues that: Line chart is used for the categorical attribute.* ***Change line chart to bar chart.****"* in **Resolving Phase**.

Different design choices are possible for detecting and resolving design issues. For example, one can use pre-defined rules and heuristics to detect design issues [4,21,23,33], which is grounded in visualization theories and practices. This would be the most simple and transparent approach but can compromise generalizability. On the other hand, one can use an automated approach by employing LLMs by giving a prompt such as *"Carefully examine the following Vega-Lite specification and find any problems."* as in [12]. This would provide more flexibility but there is no guarantee that these models enumerate all possible issues.

A similar tension exists in the **Resolving Phase**. Modifying a chart specification is one of the most imperative ways to correct a design issue; for example, one can change the `mark` attribute of a Vega-Lite specification to change the chart type, i.e., from `mark:"line"` to `mark:"bar"`. Another way is to add extra prompts to the original prompt, e.g., *"Use a bar chart."* or provide general guidelines in the prompt, e.g., *"A line chart cannot be used for a nominal attribute on the x-axis."*

Unfortunately, each choice has trade-offs, and there is no one-size-fits-all approach. In Bavisitter, we take a hybrid approach. In **Detecting Phase**, we chose to use pre-defined heuristics to keep the detection logic interpretable and detect design issues since we wanted to ensure that we did not miss the issues we surveyed (detailed in Section 3.1). The heuristics we developed for **Detecting Phase** are detailed in supplementary materials. In **Resolving Phase**, we chose to append the additional feedback prompt to the original prompt with feedback prompts to exploit the generalizability of LLMs and better keep the user informed of the changes made to their prompts by the system.

We developed Bavisitter to address the design issues identified in Section 3.1. Although Bavisitter covers many of these issues, there may be undiscovered design issues associated with different types and scales of datasets (e.g., graph datasets) that were not covered in Section 3.1. To accommodate such cases, the system can be easily extended by registering JavaScript functions that detect issues from visualizations and resolve issues with natural language feedback prompts. Detailed implementations and registration procedures are provided in the supplementary materials.

3.3 Interface Design

The user interface of Bavisitter consists of four components. In the Chat View (Fig. 3A), the user can see the visualization authoring dialogue, including the prompts they entered or the feedback prompts Bavisitter added. The Feedback Panel (Fig. 3B) below the Chat View shows the possible solutions to the detected design issues recommended by Bavisitter. On the right, the Chart View (Fig. 3C) renders the Vega-Lite code generated by the LLM, and the Thumbnail View (Fig. 3D) shows the snapshots of the generated visualizations. Bavisitter runs on a Python environment as

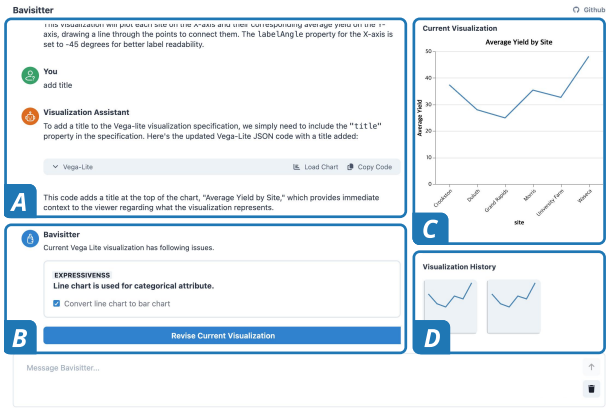

Figure 3: The User Interface of Bavisitter

an IPython widget that can be easily installed via PyPI[2]. This design facilitates the integration with external libraries, such as Scikit-learn, and LLMs, such as GPT [1] or LLaMA [31]. The source code of Bavisitter is available at GitHub[3].

4 Use Cases

4.1 Use Case 1: `flights-airport`

Suppose a scenario where a data analyst, Jason, explores the `flights-airport` dataset consisting of three columns: `origin`, `destination`, and `count`. To author a visualization that shows the relative share of destinations, he requests *"Show me the relative share of destinations"* (Fig. 4A). As a result, the LLM generates a bar chart with `destination` on the x axis and the summed `count` on the y axis (Fig. 4B). However, the bar chart has a legibility issue in that it has too many categories on the x axis due to the high cardinality of the `destination` attribute. Bavisitter automatically detects the `label-overlap` and `cardinality-excessive` issues and suggests *"Rotate labels to avoid overlap."* and *"Filter 20 unique values and group the rest."* as a solution (Fig. 4C). Once Jason confirms the solution, the feedback prompts are given to the LLM (Fig. 4D), and the LLM modifies the previous chart with better effectiveness and legibility (Fig. 4E).

4.2 Use Case 2: `barley`

Here, Jason explores the `barley` dataset with four columns: `yield`, `variety`, `year`, and `site`. To compare the average yield across different sites, he requests *"Show me the average yield by site"* (Fig. 1A). The LLM generates a line chart with `site` on the x axis and average `yield` on the y axis (Fig. 1B). Then, Bavisitter finds out that a connection mark is used to encode nominal values, raising an effectiveness issue, i.e., `line-for-categorical`, and suggests using bar marks. Jason confirms the suggestion, the feedback prompt is provided to the LLM, and the LLM updates the visualization to a bar chart (Fig. 1C).

5 Limitations and Future Work

Our system incorporates a variety of heuristic algorithms for **Detecting Phase** and manually designed prompts for **Resolving Phase**. These design choices aim to balance interpretability and flexibility. However, it is important to note that these decisions were not informed by a systematic comparison of LLMs with existing visualization tools. It would be interesting to identify the shortcomings of LLMs in various visualization tasks and address them by integrating well-established visualization authoring tools. Also, our survey is limited to 59 tabular data in Vega datasets. We plan to extend our survey to different types and scales of data (e.g., graph data, large-scale data) to be able to generalize Bavisitter to a wider variety of visualization tasks. Approaches like Bavisitter, which integrate existing heuristics or tools with LLMs, necessitate manual development of heuristics for each rule, and its impact heavily relies on heuristic performance. Moreover, certain issues, such as detecting `wrong-title`, are challenging to address using heuristics alone. To address this issue, we plan to develop a large-scale instruction tuning [15] dataset designed to train and evaluate the visualization literacy [2, 13] of vision-language models. Lastly, we are also interested in evaluating our interface through a quantitative user study to understand how our system can facilitate visualization authoring, especially for novices.

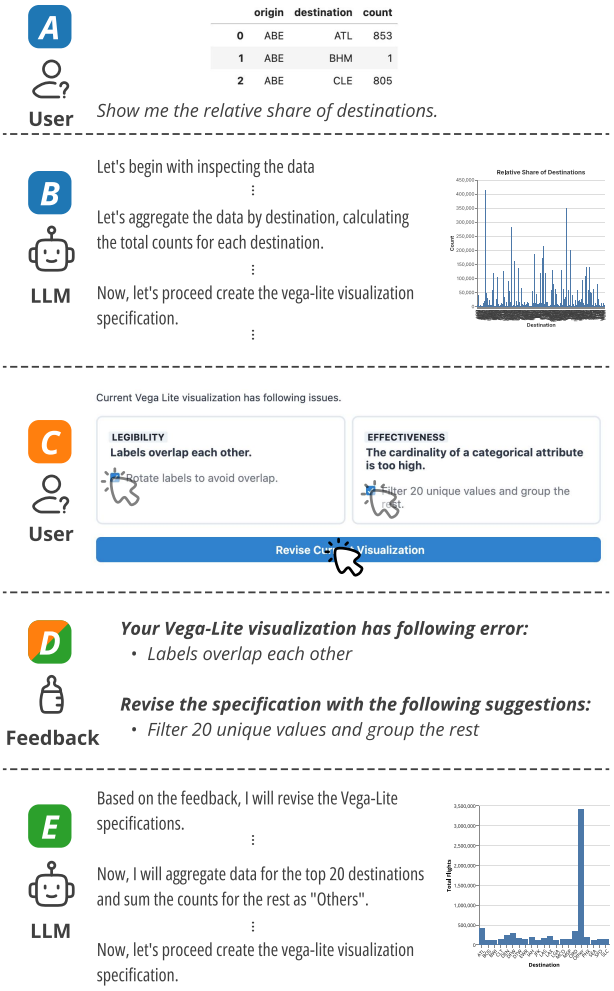

Figure 4: Use Case of Bavisitter

6 Conclusion

In this paper, we investigate an approach to integrating established visualization design guidelines into Large Language Models (LLMs). Despite the versatility of LLMs, they often generate invalid or suboptimal visualizations that fail to adhere to best practices in the field. To address this, we first identified issues LLMs raise while generating visualizations from the LLM-generated visualization corpus. Based on these issues, we designed Bavisitter, a natural language interface optimized for generating visualizations. Bavisitter detects the issues we identified and resolves them through feedback prompts, guiding the LLMs toward authoring visualizations that align with established design guidelines. By two usage examples of Bavisitter, we demonstrate how Bavisitter detects issues and resolves them to follow existing design guidelines.

[2] `pip install bavisitter`
[3] https://github.com/jiwnchoi/Bavisitter

ACKNOWLEDGMENTS

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-00421, AI Graduate School Support Program (Sungkyunkwan University)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00221186).

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 2, 3, 4

[2] J. Boy, R. A. Rensink, E. Bertini, and J.-D. Fekete. A principled way of assessing visualization literacy. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1963–1972, 2014. doi: 10.1109/TVCG.2014.2346984 4

[3] V. Braun and V. Clarke. *Thematic analysis.* American Psychological Association, 2012. 2

[4] Q. Chen, F. Sun, X. Xu, Z. Chen, J. Wang, and N. Cao. Vizlinter: A linter and fixer framework for data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):206–216, 2022. doi: 10.1109/TVCG.2021.3114804 1, 2, 3

[5] J. Choi and J. Jo. Intentable: A mixed-initiative system for intent-based chart captioning. In *2022 IEEE Visualization and Visual Analytics (VIS)*, pp. 40–44. IEEE, 2022. 2

[6] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[7] V. Dibia. Lida: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 113–126, 2023. 1, 2

[8] V. Dibia and Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications*, 39(5):33–46, 2019. doi: 10.1109/MCG.2019.2924636 2

[9] L. Grammel, M. Tory, and M.-A. Storey. How information visualization novices construct visualizations. *IEEE transactions on visualization and computer graphics*, 16(6):943–952, 2010. 1

[10] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, p. 1–12. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3290605.3300358 2

[11] Q. Huang, J. Vora, P. Liang, and J. Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation, 2024. 2

[12] N. W. Kim, G. Myers, and B. Bach. How good is chatgpt in giving advice on your visualization design? *arXiv preprint arXiv:2310.09617*, 2023. 2, 3

[13] S. Lee, S.-H. Kim, and B. C. Kwon. Vlat: Development of a visualization literacy assessment test. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):551–560, 2017. doi: 10.1109/TVCG.2016.2598920 4

[14] C. Liu, Y. Han, R. Jiang, and X. Yuan. Advisor: Automatic visualization answer for natural-language question on tabular data. In *2021 IEEE 14th Pacific Visualization Symposium (PacificVis)*, pp. 11–20. IEEE, 2021. 1, 2

[15] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds., *Advances in Neural Information Processing Systems*, vol. 36, pp. 34892–34916. Curran Associates, Inc., 2023. 4

[16] S. Liu, H. Miao, Z. Li, M. Olson, V. Pascucci, and P.-T. Bremer. AVA: Towards Autonomous Visualization Agents through visual perception-driven decision-making. *arXiv [cs.HC]*, Dec. 2023. 2

[17] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226, 2021. 2

[18] J. Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986. 2

[19] P. Maddigan and T. Susnjak. Chat2vis: Generating data visualisations via natural language using chatgpt, codex and gpt-3 large language models. *Ieee Access*, 2023. 1, 2

[20] A. Masry, P. Kavehzadeh, X. L. Do, E. Hoque, and S. Joty. Unichart: A universal vision-language pretrained model for chart comprehension and reasoning. *arXiv preprint arXiv:2305.14761*, 2023. 2

[21] A. McNutt and G. Kindlmann. Linting for visualization: Towards a practical automated visualization guidance system. In *VisGuides: 2nd Workshop on the Creation, Curation, Critique and Conditioning of Principles and Guidelines in Visualization*, vol. 1, p. 2, 2018. 1, 2, 3

[22] R. Mitra, A. Narechania, A. Endert, and J. Stasko. Facilitating conversational interaction in natural language interfaces for visualization. In *2022 IEEE Visualization and Visual Analytics (VIS)*, pp. 6–10. IEEE, 2022. 1, 2

[23] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE transactions on visualization and computer graphics*, 25(1):438–448, 2018. 2, 3

[24] 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, 2020. 1, 2

[25] J. Obeid and E. Hoque. Chart-to-text: Generating natural language descriptions for charts by adapting the transformer model. *arXiv preprint arXiv:2010.09142*, 2020. 2

[26] OpenAI. Code Interpreter - OpenAI API. https://platform.openai.com/docs/assistants/tools/code-interpreter. [Accessed 18-04-2024]. 2

[27] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics*, 23(1):341–350, 2016. 1, 2

[28] 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*, pp. 365–377, 2016. 1, 2

[29] V. Setlur, M. Tory, and A. Djalali. Inferencing underspecified natural language utterances in visual analysis. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pp. 40–51, 2019. 1, 2

[30] A. Srinivasan, S. M. Drucker, A. Endert, and J. Stasko. Augmenting visualizations with interactive data facts to facilitate interpretation and communication. *IEEE transactions on visualization and computer graphics*, 25(1):672–681, 2018. 2

[31] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 4

[32] Y. Wang, Z. Hou, L. Shen, T. Wu, J. Wang, H. Huang, H. Zhang, and D. Zhang. Towards natural language-based visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):1222–1232, 2022. 1, 2

[33] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658, 2015. 3

[34] J. Yang, A. Prabhakar, K. Narasimhan, and S. Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36, 2024. 2