An Agentic Drawer–Advisor for Content-Aligned Visualization

Chunggi Lee[*] Harvard University
Woorak Park Brekky Lab
Haejoon Kim Brekky Lab
Jaehwan Lee Brekky Lab
Hanspeter Pfister[†] Harvard University

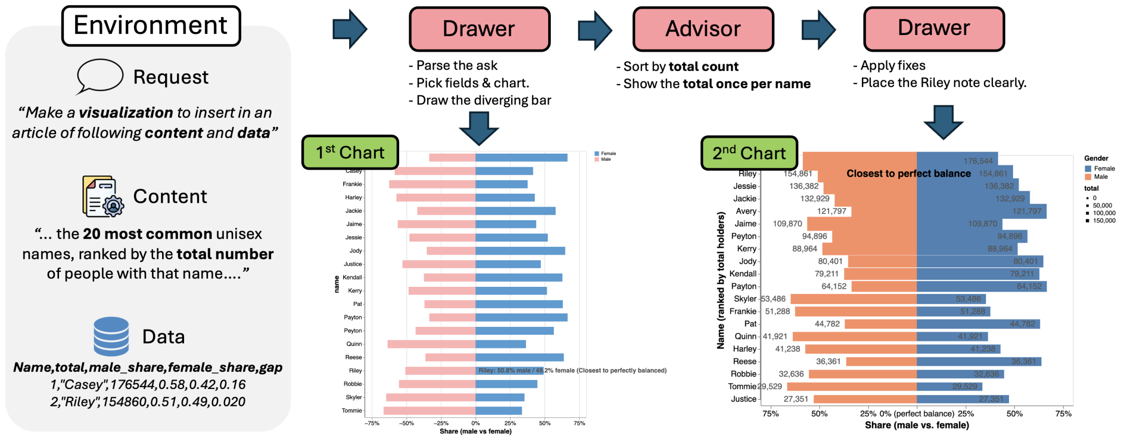
Environment
Request
"Make a visualization to insert in an article of following content and data"
Content
"... the 20 most common unisex names, ranked by the total number of people with that name...."
Data
Name,total,male_share,female_share,gap
1,"Casey",176544,0.58,0.42,0.16
2,"Riley",154860,0.51,0.49,0.020
Drawer
- Parse the ask
- Pick fields & chart.
- Draw the diverging bar
Advisor
- Sort by total count
- Show the total once per name
Drawer
- Apply fixes
- Place the Riley note clearly.
1st Chart
2nd Chart
Closest to perfect balance
Name (ranked by total holders)
Share (male vs. female)
Gender
Female
Male

Figure 1: An agentic AI (Drawer–Advisor) uses the user's question, the data, and article context to auto-draw a chart, then critique and revise it. For "visualize the 20 most common unisex names by total count" with CSV (name, total, male_share, female_share, gap), the Drawer drafts a male–female diverging bar chart. The Advisor recommends ranking by total, showing one total per name, and clarifying the Riley note and the 50% balance; the Drawer applies these fixes to yield the revised figure.

Abstract

Data visualization plays a crucial role in communicating insights, yet both human- and LLM-generated visualizations suffer from issues such as misleading chart types, scales, and encodings. To address these limitations, we propose an agentic AI approach that iteratively refines visualizations through multi-agent orchestration. Our framework coordinates a *Drawer* agent, which generates candidate charts, and an *Advisor* agent, which critiques them based on a checklist of common visualization pitfalls. We compare two orchestration paradigms, imperative internal looping and declarative graph-based state machines, each combined with different plotting libraries. In a user study, participants ranked outputs from these variants. This work contributes empirical insights into orchestration strategies for LLM-driven visualization refinement.

Index Terms: Agentic AI, Agentic Visualization

1 Introduction

Data visualization is essential to uncover, interpret, and communicate complex data patterns. However, human-produced charts are subject to significant limitations, such as mismatched chart types, distorted axes, poor color choices, or excessive clutter, all of which can mislead interpretation or obscure insights. Previous studies [3, 11, 14] show that the misuse of pie charts and improper treatment of size encodings frequently yields misleading visual inferences.

Large Language Models (LLMs) have recently enabled natural-language interfaces for visualization authoring, lowering technical barriers for non-experts. With the growing adoption of LLM-based interfaces, many users now delegate chart creation directly to the model. However, this convenience does not guarantee correctness: LLM-generated figures contain technical faults (e.g., invalid specifications or omitted fields), content–chart misalignment, and stylistic misjudgments. A single-shot LLM inference yields flawed outputs that novices neither detect nor repair. In these scenarios, the risk is not only that human biases are reproduced, but also that users accept suboptimal visualizations at face value.

Agentic AI-multi, agent systems in which specialized components collaborate toward a shared objective, has emerged as a promising paradigm for complex, multi-step reasoning and tool-mediated execution. In contrast to single-pass models, agentic systems engage in explicit planning, tool use, and iterative self-critique, coordinating complementary roles (e.g., planner, executor, critic). Such workflows have shown utility in domains including code generation and debugging [17], data analysis [20], and conversational assistance [16]. Their emphasis on feedback-driven revision aligns naturally with the demands of visualization refinement.

To address the limitations of single-shot visualization, we introduce an agentic AI framework that orchestrates a closed loop between a *Drawer* agent, which generates candidate visualizations using function calls to plotting libraries, and an *Advisor* agent, which evaluates the visualizations and issues plain-text feedback. The system proceeds iteratively until the Advisor returns an "OK" signal or the process reaches a predefined iteration limit. Rather than relying on formal constraint languages, the Advisor is primed with a checklist of known visualization pitfalls (e.g., axis truncation, overplotting, mis-

[*]e-mail: chunggi_lee@g.harvard.edu
[†]e-mail: pfister@g.harvard.edu

leading encodings, and poor chart-type selection) drawn from empirical design guidelines.

We study two coordination strategies for managing this loop: (1) *imperative internal looping*, a procedural control flow embedded within agent logic, suited to tightly coupled and predictable workflows; and (2) *declarative graph-based orchestration*, a state-machine formulation that supports branching, retries, and explicit termination. These orchestration styles are evaluated in combination with two widely used visualization libraries (Plotly and Vega-Lite), resulting in four system configurations. We compare them in a user study that collects quantitative rankings to assess iteration efficiency, error mitigation, and perceived visualization quality.

In summary, our contributions are as follows: 1) a two-agent *Drawer–Advisor* workflow that conditions on the user's question, the data, and article context to auto-draw an initial chart and iteratively revise it via targeted critiques; 2) a controlled comparison of orchestration strategies (imperative loop vs. declarative graph/state machine) and evaluation modalities (image- vs. spec-based) across two code backends (Vega-Lite [13], Plotly [12]), with analysis of iteration efficiency and fix coverage under identical tasks; and 3) a practical checklist and task set for benchmarking visualization refinement.

2 Related Work

2.1 Agentic AI and Visualization.

Agentic AI denotes multi-agent systems that decompose tasks, use tools, and iterate via feedback with minimal supervision. Established frameworks demonstrate effectiveness in complex, tool-mediated workflows (e.g., AutoGen [16] and SWE-agent [17]), while role-playing and deliberate-search methods (e.g., Tree-of-Thoughts [19]) show how planning and critique improve reliability. Within visualization, *agentic visualization* has been proposed as a design lens specifying roles, communication, and coordination while preserving human agency [4]. Systems such as MatPlotAgent and PlotGen implement multi-agent pipelines for scientific plotting and benchmarking [18, 6]. Common design choices include role specialization (query understanding, code generation, critique), tool-grounded execution via plotting libraries, lightweight memory/provenance, and explicit stopping or retry criteria. These choices move the pipeline beyond single-shot generation toward feedback-driven refinement. Our work extends this line by (i) *explicitly conditioning* planning and critique on the triad of *question, data, and article context*, thereby anchoring decisions in communicative intent; and (ii) *treating orchestration*. We instantiate a two-agent *Drawer–Advisor* loop and contrast *imperative internal looping* with *declarative graph/state-machine* control, each paired with two output types (Plotly, Vega-Lite).

2.2 Visualization Pitfalls

A substantial body of work documents recurring visualization pitfalls that impair comprehension and, when noticed, erode trust. Controlled experiments [3] show that truncating the y-axis systematically biases effect-size judgments even when truncation is explicitly signaled. Broader surveys [11] catalog frequent errors in scientific publications, from misleading scales and inappropriate encodings to poor color choices and cluttered layouts. Complementing these accounts, practitioner-facing syntheses such as "Five Ways Visualizations Can Mislead (and How to Fix Them)" consolidate evidence and remedies for common failure modes (e.g., deceptive baselines, unwarranted 3D, and overloaded palettes)

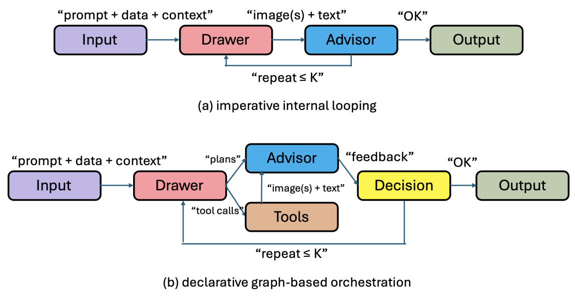

Figure 2: Two orchestration strategies for coordinating the *Drawer* and *Advisor* agents: (a) imperative internal looping and (b) declarative graph-based orchestration.

[14]. Color design is a persistent source of error: the rainbow colormap lacks perceptual ordering and exhibits uncontrolled luminance changes, motivating perceptually uniform alternatives and visualization-specific color-difference models [1, 14]. These findings motivate our checklist-driven *Advisor*, which operationalizes known pitfalls (axes, encodings, color, clutter, labeling) as checkable criteria that guide iterative repair in the Drawer–Advisor loop.

Complementing this empirical evidence, formalizations such as Draco [10] render design knowledge machine-checkable and support checking, ranking, and recommending alternatives. Mixed-initiative recommenders such as Voyager 2 [15] (via CompassQL) propose and rank viable charts from partial specifications. At the same time, work on "visualization mirages" cautions against seemingly meaningful but unreliable patterns that arise from design or data choices [9]. Building on this foundation, our system embeds a curated pitfall checklist as criteria for an *Advisor* that critiques axes, encodings, color, clutter, and labeling, and feeds structured guidance to a *Drawer* that applies targeted revisions in a closed loop. We aim to reduce model-induced visualization errors by converting critique into concrete repairs within an agentic loop, moving beyond single-shot generation.

3 Agentic AI System Design

Our agentic AI system for visualization refinement consists of two specialized agents that operate in a coordinated loop: a *drawer* and an *advisor*. The drawer is responsible for producing candidate visualizations based on the provided dataset and user intent, while the advisor evaluates these visualizations against known pitfalls and best practices, issuing corrective instructions to improve the output. This section describes the system's core components, evaluation modalities, and orchestration strategies.

3.1 Problem Statement and Scope

We study whether an agentic AI workflow, a *Drawer* that drafts a chart and an *Advisor* that critiques and requests revisions, can transform a user request plus data and article context into higher-quality visualizations through iterative refinement. We focus on News article/analysis tasks (bar/line/point charts with annotations) and exclude bespoke infographics and 3D scenes. Inputs are (i) a natural-language prompt, (ii) a tabular dataset, and (iii) optional article context. Outputs are a figure and rationale notes produced after at most K iterations.

3.2 Orchestration Strategies

We compare two orchestration strategies for coordinating the *drawer* and *advisor* agents, as illustrated in Figure 2.

Imperative Internal Looping. In the imperative mode (Figure 2 a), the orchestration is encoded procedurally as a step-by-step loop within the agent logic. The drawer generates a visualization, the advisor evaluates it, and the drawer revises the output based on feedback, repeating until a stopping condition is met (e.g., satisfactory quality or a maximum iteration cap K). This approach is simple and effective for predictable workflows, but the control flow is implicit and rigid, making it difficult to support exceptions or alternate branches without manual code modification.

Declarative Graph-based Orchestration. In the declarative mode (Figure 2 b), the orchestration is represented as a graph or state machine. Each node corresponds to a distinct stage (generation, tool execution, evaluation, decision), and edges explicitly encode possible transitions (e.g., "plans" vs. "tool calls," "OK" vs. "repeat $\leq K$"). This externalizes the control logic (e.g., `should_execute_tools`, `should_continue`) from the agent implementation, enabling flexible branching, retries, and dynamic adaptation. While this design introduces additional setup complexity, it scales more naturally to heterogeneous tools, multiple evaluators, or human-in-the-loop, making it more extensible for complex orchestration.

3.3 System Components

Drawer Agent. The drawer agent receives the user's request and dataset, along with any contextual information about preferred chart types or design constraints. Its primary role is to generate candidate visualizations, either by directly invoking a plotting backend (e.g., producing a Vega-Lite JSON schema that specifies data mappings, encodings, and styling) or, in some cases, by proposing a high-level plan of which chart type to use and why. The latter "planning" pathway allows the advisor to evaluate the rationale before committing to tool execution. This dual capability aligns with the graph-based orchestration, where the drawer either issues immediate tool calls or outputs plans that trigger a review step.

Prompt for Drawer Agent (Visualization Creator)

The drawer agent generates visualizations from the user's prompt, data, and context. **Instructions:**

- Select chart types appropriate for the goal and data (bar, line, scatter, etc.).
- Always produce an actual chart by calling a plotting tool with real values.
- Add informative titles, axis labels, and use clear, consistent colors.
- Avoid clutter: keep legends concise and highlight key comparisons.
- Save each chart with a unique filename; revise if advisor feedback is given.

Advisor Agent. The advisor agent serves as the quality-control mechanism in the loop. It is equipped with rules and heuristics derived from empirical studies on visualization pitfalls (e.g., inappropriate chart types, misleading scales, non-perceptual color maps, excessive clutter). Depending on the drawer's output, the advisor operates in two modalities: (i) when given a visualization plan (a high-level proposal of chart type and rationale), it evaluates the appropriateness of the choice before execution; (ii) when given a rendered visualization, it inspects the result for clarity and effectiveness. In both cases, the advisor returns structured feedback or direct modification instructions to the drawer, closing the loop.

Prompt for Advisor Agent (Visualization Reviewer)

The advisor agent reviews charts created by the drawer agent. **Instructions:**

- Judge if the visualization effectively supports the intended message.
- Check for pitfalls: wrong chart choice, misleading axes, poor color use, clutter.
- Suggest only meaningful, concrete improvements (clarity, labels, encoding).
- Keep feedback short (1–5 plain-text sentences).
- If the chart is acceptable, respond exactly with `OK`.

3.4 Evaluation Modalities

The advisor can evaluate the drawer's output in one of two modes: **Image-based Evaluation.** In this mode, the advisor inspects a rendered image of the visualization. This approach allows direct assessment of perceptual qualities (e.g., readability, color distinguishability, label clarity, and overall visual balance) that is not easily inferred from code alone. However, it requires additional rendering and loses certain semantic details about the underlying specification. **Specification-based Evaluation.** In this mode, the advisor examines the Vega-Lite JSON specification generated by the drawer. This allows precise checking of structural and semantic aspects of the visualization, such as encoding channels, scale domains, and axis configurations. Specification-based evaluation enables the detection of issues like truncated axes or inappropriate aggregation, and also facilitates automated repair by modifying the JSON directly.

3.5 Implementation Details

We use the same LLM across conditions with temperature $\tau = 0$. All system prompts are kept identical across settings. We implement tool adapters for Vega-Lite[13] and Plotly[12]. Advisor rules instantiate the checklist with auto-fixes for SPEC mode (which modifies JSON) and textual critiques for IMG mode (where the drawer applies edits). We set the maximum number of iterations to $K = 2$ with early stopping if advisor generated "OK". For orchestration, we implemented *Imperative Internal Looping* using Ailoy [2] and *Declarative Graph-based Orchestration* using LangGraph [7].

4 User Study

Research Questions. We investigate two main research questions: **RQ1:** Does an iterative Drawer–Advisor framework outperform a single-pass baseline in producing more contextually appropriate visualizations? **RQ2:** Within the iterative framework, how do orchestration (*imperative vs. graph*) and backend (*Vega-Lite vs. Plotly*) influence the quality and suitability of the resulting visualizations?

Study Design. We adopted a within-subjects design with six participants. Each participant completed five conditions: one *single-pass baseline* and four *framework variants* (imperative–Vega-Lite, imperative–Plotly, graph–Vega-Lite, graph–Plotly). Evaluation mode (*spec vs. image*) was not introduced as a separate factor but followed naturally from the backend: Vega-Lite conditions produced JSON specifications while Plotly conditions generated rendered images. We

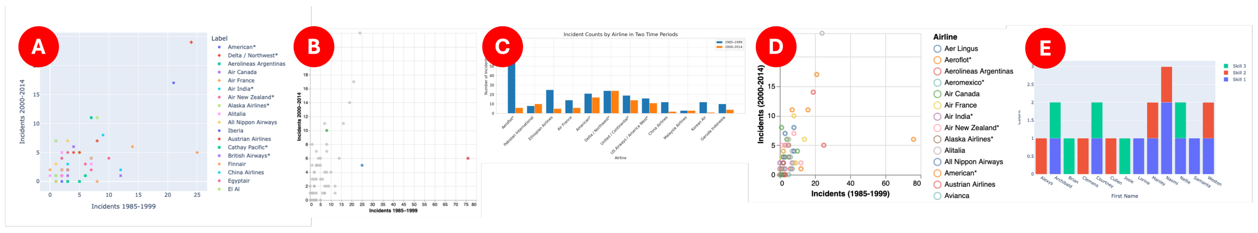

Figure 3: Representative outputs for an article-driven task (Q10). Condition E (Graph + Vega-Lite, left) produces mismatched encodings and irrelevant groupings, leading to poor interpretability. Condition C (Baseline single-shot, middle) produces a simple output but fails to capture key comparisons. Iterative refinement (e.g., A, B and D) yields more balanced and interpretable charts.

counterbalanced the order of framework conditions across participants to mitigate ordering effects.

For each condition, participants were given identical tasks. We curated the task inputs from two sources: (i) five tasks drawn from established NL2VIS benchmarks [8] (adapted from NL2SQL tasks), and (ii) five tasks derived from real-world articles on the FiveThirtyEight open dataset repository[1]. Each task included the dataset, context, and a visualization prompt. After viewing the outputs, participants ranked the five visualizations from 1 (best) to 5 (worst) based on how well each matched the intended context.

Results. We first summarize the overall distribution of participant rankings across the five conditions. The mean ranks (lower is better) were: A=2.80, B=2.87, C=3.10, D=2.87, and E=2.93. Overall, the four iterative variants (A, B, D, and E) achieved better average ranks than the single-shot baseline (C). In terms of top-1 votes, A and C led with 14 each, while B and D followed closely with 13. However, C's higher mean rank and 14 bottom-rank votes indicate inconsistency. E also underperformed, with the weakest mean rank among iterative variants and 12 bottom-rank votes. These results support **RQ1**, showing that iterative approaches yield more reliable outcomes overall. Yet, as the task-type analysis below reveals, the relative strengths of the baseline and iterative variants depend on task complexity.

Task-type Analysis. We divided the tasks into two categories: *Simple Queries (Questions 1–5)* involved straightforward query-to-chart mappings, while *Article-based Context (Questions 6–10)* required interpreting content from real articles and synthesizing context into the visualization. Tables 1 and 2 show the updated ranking distributions.

Condition	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
A	8	10	6	1	5
B	3	5	3	13	6
C	11	3	8	3	5
D	4	4	8	5	9
E	9	11	6	3	1

Table 1: Ranking distribution for Simple Queries (Q1–Q5). Condition mapping: A = Imperative + Plotly, B = Imperative + Vega-Lite, C = single-shot, D = Graph + Plotly, E = Graph + Vega-Lite.

In *Simple Queries* (Table 1), where tasks were more straightforward, performance was relatively even: all systems occasionally received higher ranks, and the baseline (C) even led in top-1 votes. This suggests that direct NL2VIS mappings can be handled without iterative refinement. By contrast, *Article-based Context* tasks (Table 2) revealed sharper distinctions. The imperative–Vega-Lite variant (B) obtained the most Rank 1 votes (10), with the graph–Plotly variant (D) close behind (9), showing stronger alignment when contextual interpretation was required. Meanwhile, the graph–Vega-Lite variant (E) accumulated the most Rank 5 votes (11), reflecting notable struggles with context-rich tasks.

[1]`https://data.fivethirtyeight.com/`

Condition	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
A	6	4	6	9	5
B	10	10	6	0	4
C	3	1	10	7	9
D	9	10	3	7	1
E	2	5	5	7	11

Table 2: Ranking distribution for Article-based Context (Q6–10).

This divergence highlights that while single-shot approaches suffice for simple queries, article-driven synthesis tasks better expose the strengths of iterative orchestration. The qualitative example in Figure 3 illustrates this trend: Condition E fails due to irrelevant encodings, while Condition C underperforms by oversimplification. These findings reinforce **RQ2**: orchestration and backend choices influence outcomes, particularly under complex, context-driven scenarios, with Condition D emerging as the most balanced variant.

5 Limitations, Future Work, and Lessons Learned

Limitations. Our study has several limitations. The participant pool was small ($N = 6$), limiting generalizability, and the two task types (simple queries vs. article-based context) do not capture the full range of NL2VIS scenarios. We also drew queries from NL2SQL datasets and articles from FiveThirtyEight, which do not represent other domains. Finally, short-session exposure to multiple systems introduced learning or carryover effects.

Lessons Learned. We highlight four lessons. First, simple queries can be adequately solved even with single-shot approaches. Second, article-based tasks reveal clearer distinctions, with iterative orchestration offering stronger contextual reasoning. Third, well-structured frameworks (e.g., Condition D) balance accuracy and interpretability, yielding higher preference. Fourth, failure to integrate external context (e.g., Condition E) was penalized, underscoring the importance of contextual grounding. Overall, orchestration matters more than optimizing only for direct query-to-chart mappings.

Future Work. Future work should validate our framework in real authoring workflows with human-in-the-loop critique. Promising directions include adaptive orchestration that varies by task complexity, learning repair strategies from logs, and integrating structured critique planning [19, 10]. Benchmarking against emerging agentic visualization systems [18, 6, 5] will further situate our approach within broader design patterns.

References

[1] D. Borland and R. M. T. Ii. Rainbow color map (still) considered harmful. *IEEE computer graphics and applications*, 27(2):14–17, 2007. 2

[2] Brekky Lab. Ailoy: Lightweight multi-agent runtime for llm applications, 2024. Accessed: 2025-08-18. 3

[3] M. Correll, E. Bertini, and S. Franconeri. Truncating the y-axis: Threat or menace? In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pp. 1–12, 2020. 1, 2

[4] V. Dhanoa, A. Wolter, G. M. León, H.-J. Schulz, and N. Elmqvist. Agentic visualization: Extracting agent-based design patterns from visualization systems. *arXiv preprint arXiv:2505.19101*, 2025. 2

[5] V. Dhanoa, A. Wolter, G. M. León, H.-J. Schulz, and N. Elmqvist. Agentic visualization: Extracting agent-based design patterns from visualization systems. *arXiv preprint arXiv:2505.19101*, 2025. 4

[6] K. Goswami, P. Mathur, R. Rossi, and F. Dernoncourt. Plotgen: Multi-agent llm-based scientific data visualization via multimodal feedback. *arXiv preprint arXiv:2502.00988*, 2025. 2, 4

[7] LangChain Inc. Langgraph: Build multi-agent workflows with llms, 2024. Accessed: 2025-08-18. 3

[8] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin. Synthesizing natural language to visualization (nl2vis) benchmarks from nl2sql benchmarks. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 1235–1247, 2021. 4

[9] A. McNutt, G. Kindlmann, and M. Correll. Surfacing visualization mirages. In *Proceedings of the 2020 CHI Conference on human factors in computing systems*, pp. 1–16, 2020. 2

[10] 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, 4

[11] V. T. Nguyen, K. Jung, and V. Gupta. Examining data visualization pitfalls in scientific publications. *Visual Computing for Industry, Biomedicine, and Art*, 4(1):27, 2021. 1, 2

[12] Plotly Technologies Inc. Plotly: Collaborative data science and visualization platform, 2015. Accessed: 2025-08-18. 2, 3

[13] 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, 2017. 2, 3

[14] D. A. Szafir. The good, the bad, and the biased: Five ways visualizations can mislead (and how to fix them). *interactions*, 25(4):26–33, 2018. 1, 2

[15] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 chi conference on human factors in computing systems*, pp. 2648–2659, 2017. 2

[16] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. 1, 2

[17] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024. 1, 2

[18] Z. Yang, Z. Zhou, S. Wang, X. Cong, X. Han, Y. Yan, Z. Liu, Z. Tan, P. Liu, D. Yu, et al. Matplotagent: Method and evaluation for llm-based agentic scientific data visualization. *arXiv preprint arXiv:2402.11453*, 2024. 2, 4

[19] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. *URL https://arxiv.org/abs/2305.10601*, 3:1, 2023. 2, 4

[20] Z. You, Y. Zhang, D. Xu, Y. Lou, Y. Yan, W. Wang, H. Zhang, and Y. Huang. Datawiseagent: A notebook-centric llm agent framework for automated data science. *arXiv preprint arXiv:2503.07044*, 2025. 1

Appendix

Prompt for Drawer Agent

The drawer agent generates data visualizations from the user's prompt, data, and context.
Instructions

- Choose chart types that fit the goal and data (bar/-line/scatter/...) using the typology guide.
- **Always call a plotting function with actual values.** Do not only describe the plan.
- Add informative titles, axis labels, and readable ticks; avoid clutter.
- Use consistent, meaningful color encodings; avoid rainbow for ordered data.
- Save each chart with a unique filename (include an index).
- If the advisor provides feedback, *revise* the chart and re-emit a new file.

Available knowledge

- Chart typology cheat-sheet (scatter, bar, stacked bar, line, violin, box, histogram, radar, pie, gantt).

Critical constraint: You *must* execute one of the registered plotting tools and produce an image file.

Iteration & Orchestration Policy

Loop: Drawer → (tool call) → Advisor → Drawer ...
Stopping: early stop on `OK` or when $K=3$ iterations are reached.
Decision rules:

- If no tool output is produced, request a concrete tool call with extracted values.
- Prefer simpler charts that directly answer the prompt; annotate key comparisons.
- Enforce filename uniqueness across iterations.

Advisor Checklist (Abbrev.)

General design (ordering, avoid clutter); scales/axes (no deceptive truncation, sensible bins, aspect ratios); encoding (avoid overplotting; use jitter/alpha; avoid radar unless cyclic); color (no rainbow for ordered data; meaningful palettes; concise legends); interpretability (titles/units/annotations; reduce mental arithmetic).

Prompt for Advisor Agent

The advisor evaluates drawer outputs and returns concise, actionable feedback.
Goal Ensure the visualization serves the intended message and avoids common pitfalls.
Inputs

- Original query/context and any drawer text.
- One or more rendered chart images (or a spec, if provided).

Evaluation checklist (abbrev.)

- *Chart choice*: mismatch to task/data; avoid needless 3D; watch stacking.
- *Scales/axes*: truncation, wrong aspect, misleading dual axes, poor binning.
- *Encoding*: overplotting without jitter/alpha; inappropriate size/area; too many series.
- *Color/legend*: non-perceptual ramps; inconsistent mapping; oversized legends.
- *Clarity*: titles/labels/units; annotation for key points; reduce clutter.

Output format

- If acceptable: reply exactly `OK`.
- Otherwise: 1–5 short sentences with specific edits the drawer can apply.

plot_scatter_chart

Purpose. Create 2D scatter plots with optional log scale, marginal plots, and trendlines.
Signature.
```
plot_scatter_chart(x_values, y_values,
 labels=None, x_title="x", y_title="y",
 label_title="Label", log_x=False, log_y=False,
 color_map="qualitative", marginal_x=None,
 marginal_y=None, trendline=False,
output_path="scatter.png") -> str
```
When to use. Explore relationships, clusters, or outliers between two quantitative variables.
Options. `marginal_x` and `marginal_y` can be `rug`, `box`, `violin`, or `histogram`.
Tips. For dense data, consider transparency or sampling. Log axes require non-zero positive values.

plot_bar_chart

Purpose. Compare categorical values with vertical or horizontal bars. Supports grouping, stacking, and broken axes.
Signature.
```
plot_bar_chart(x_values, y_values, labels=None,
  x_title="x", y_title="y", log_y=False,
  barmode="relative", color_map="qualitative",
  orientation="v", text_auto=False,
  broken_axis_ranges=None, axis_to_break="y",
  output_path="bar.png") -> str
```
When to use. Compare discrete categories across one or multiple series.
Options. `barmode` can be `group`, `stack`, or `relative`.
Tips. Order categories meaningfully. Use `text_auto` for quick labels. Broken axis ranges help when values differ by magnitude.

The remaining functions (`plot_stacked_bar_chart`, `plot_line_chart`, `plot_violin_chart`, `plot_gantt_chart`, `plot_box_chart`, `plot_histogram`, `plot_radar_chart`, `plot_pie_chart`) follow the same format: each card specifies (1) *Purpose*, (2) *Function signature*, (3) *When to use*, (4) *Options*, and (5) *Tips*.