Demonstrating chart-plot: Closing the Last Mile of Academic Chart Generation

Yinghao Tang
State Key Lab of CAD&CG, Zhejiang University
Hangzhou, China
yinghaotang@zju.edu.cn

Yupeng Xie
HKUST(GZ)
Guangzhou, China
yxie740@connect.hkust-gz.edu.cn

Yingchaojie Feng
National University of Singapore
Singapore, Singapore
feng.y@nus.edu.sg

Jiale Lao
Cornell University
Ithaca, NY, USA
jiale@cs.cornell.edu

Tingfeng Lan
University of Virginia
Charlottesville, VA, USA
tingfeng@virginia.edu

Wei Chen
State Key Lab of CAD&CG, Zhejiang University
Hangzhou, China
chenvis@zju.edu.cn

ABSTRACT

Large language models can translate a researcher's intent into runnable `matplotlib` code, yet the resulting chart rarely lands in a paper without multiple rounds of manual revision. We argue that the open problem is not chart *code* generation but chart *publication*: making the output look like a top-venue figure, survive the target layout, and respond to precise author edits. We present **chart-plot**, an agentic harness that closes this last mile through three components: (1) a style-aware code generator conditioned on a textual style skill distilled from accepted figures at the target venue, (2) a deployment-aware render loop that compiles the chart inside the target LaTeX context and revises until layout constraints are met, and (3) a structured edit layer that exposes every chart element as a directly manipulable handle. We report early results on three chart-type case studies (grouped bar, scaling line, paired distributions) and a small user study. Project resources are available at https://github.com/blackblue-labs/zeroplot.

VLDB Workshop Reference Format:
Yinghao Tang, Yupeng Xie, Yingchaojie Feng, Jiale Lao, Tingfeng Lan, and Wei Chen. Demonstrating chart-plot: Closing the Last Mile of Academic Chart Generation. VLDB 2026 Workshop: ADS 2026: The Joint Workshop on Agentic Data Systems and Data-Centric AI (The 1st ADS & 3rd DATAI).

1 INTRODUCTION

A modern paper has dozens of figures, many of them charts: ablation bar plots, scaling curves, scatter plots of latent embeddings. Large language models can already translate a researcher's intent and a CSV into runnable `matplotlib` code in seconds. Yet the chart that comes out of the model still rarely makes it into the paper without two or three rounds of manual edits.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

Figure 1: The last mile. A generated chart becomes publication-ready only after three paper-facing concerns are resolved together: matching the visual conventions of the target venue, surviving column-width deployment, and supporting precise author edits.

We argue that the open problem is not chart *code* generation; as model capabilities advance, existing systems handle that step increasingly well [3, 6, 17, 19, 21, 32, 35]. The open problem is chart *publication*: turning a generated chart into a figure that looks at home in a top-venue paper, survives the target layout, and can be fixed with a small targeted edit when the model gets it almost right. We call this gap the **last mile of academic chart generation** and identify three concrete failure modes (Figure 1).

Failure 1: style mismatch. Top-venue charts follow community conventions. AI papers favour low-saturation palettes, sans-serif labels, and confidence ribbons; systems papers favour serif labels, hatched bars, and explicit error bars. A naively generated chart is plausible but visually off-style.

Failure 2: deployment mismatch. A chart rendered at `matplotlib`'s default 6.4×4.8 inches looks fine in isolation. Dropped into a two-column ACM template at 3.3-inch column width, tick labels become

unreadable, legends overlap data, and titles clip. The figure is correct only once it compiles inside the paper.

Failure 3: edit mismatch. Once a chart is mostly right, authors want small geometric edits: move a legend, shrink an annotation, swap two colours. Re-prompting an LLM is unreliable here because the model cannot localise a few-pixel change from a text instruction alone.

Our approach: a harness. We close these gaps with a **harness** [5, 14]: a deterministic scaffold around a generative LLM that injects publication context the model cannot infer from a prompt alone, such as the column width of the target template or the venue's visual conventions. chart-plot organises this scaffold across a spectrum from full automation to author control: (1) a *style* layer conditions generation on the venue's visual conventions; (2) a *deployment* layer revises the figure automatically until it fits the target LaTeX template; and (3) an *edit* layer hands the author direct handles on every figure element when geometric judgment is required. The result is a closed loop that ends only when the figure is publication-ready.

Contributions. We present **chart-plot**, an agentic harness that closes the last mile through three components, each targeting one failure mode.

- A **style-aware code generator** that conditions on a curated corpus of accepted figures, distilled into a textual style skill that is injected at generation time (§4.1).
- A **deployment-aware render loop** that compiles the chart inside the target LaTeX context, detects layout failures, and revises the code until constraints are met (§4.2).
- A **structured edit layer** that exposes every `matplotlib` element as a directly manipulable handle, mapping author gestures back to code mutations (§4.3).

Together, the three components form a harness that slots into any LLM-backed analysis workflow as the final reporting step: the researcher provides data and intent, and chart-plot delivers a publication-ready figure.

2 RELATED WORK

2.1 Agent Harness Engineering

Early tool-using agents introduced the basic loop of reasoning, action, and observation, making external tools and feedback part of model behavior rather than post-hoc additions [16, 33]. Building on these primitives, recent agent systems increasingly move reliability into the harness: the infrastructure that manages execution, tool interfaces, context, orchestration, verification, and human control. Multi-agent and workflow frameworks make orchestration explicit [7, 10, 27], while coding and web agents couple models with controlled execution environments, tool protocols, and benchmark feedback [9, 18, 24, 31, 36]. Recent harness-specific work makes this shift explicit by arguing that long-horizon agent performance is often bounded by the wrapper around the model, and by studying harness taxonomies, harness scaling, executable code harnesses, and automated harness optimization [5, 11, 14]. chart-plot follows this harness view, but specializes it to academic chart publication: its context injection, execution loop, verification checks, and author controls are organized around the *last mile* from generated chart code to a paper-ready figure.

2.2 Publication-Aware Chart Generation

LLM-based visualization systems have made chart generation increasingly executable, multimodal, and self-correcting. Early systems translate data and intent into visualization code through multi-stage generation pipelines [3], while chart-specific models and benchmarks improve chart generation, chart-to-code recovery, and aesthetic evaluation through instruction tuning, multimodal corpora, and executable program synthesis [6, 19, 29, 34, 35]. More recent agentic systems add execution, debugging, visual critique, specialized critics, and multi-path feedback loops to improve generated visualization scripts [15, 17, 25, 26, 28, 32]. In parallel, visualization authoring work studies how users refine communicative figures through design constraints, direct manipulation, code links, mixed-initiative interfaces, and human-agent collaboration [2, 13, 20–23, 38]; scientific figure tools further emphasize reproducible post-generation editing and image- or vector-level refinement [4, 30, 37]. These lines improve chart code, standalone visual quality, or interactive editing, but they usually do not treat the target paper as the execution environment. chart-plot instead makes publication context first-class: venue style, LaTeX layout constraints, structured chart layout, and author edits are joined in one harness so that generation ends with a reproducible, publication-ready figure.

3 SYSTEM OVERVIEW

Figure 2 shows the chart-plot architecture. The system takes four inputs: a data table, an intent string, a target venue, and a target LaTeX template; the author may optionally select a reference style to steer the venue's visual conventions. It returns a publication-ready figure together with the reproducible `matplotlib` source that produced it.

Design philosophy. We frame chart-plot as a *harness*: infrastructure that channels the raw generative power of a code LLM into a publication-ready artifact by learning the gold standard of each target scenario. The harness operates at three levels, arranged along an automation spectrum.

(1) **Style** (recommended, user-selectable). The harness learns visual conventions from a curated corpus of accepted figures, compiled into a textual style skill that is injected into the codegen system prompt at generation time. The author can swap in a different skill per venue or per lab (§4.1).
(2) **Deployment** (fully automated). The harness learns the layout rules of the target template (column width, font budget, spacing) and enforces them through a compile-and-revise loop. No author intervention is needed unless the constraints are fundamentally unsatisfiable (§4.2).
(3) **Edit** (author-controlled). For preferences that no model can anticipate, the harness exposes click-targets over the rendered figure: selecting an element opens a contextual inspector with controls specific to its kind (legend location, font size, color, line style, etc.). Each control maps to a concrete code mutation (§4.3).

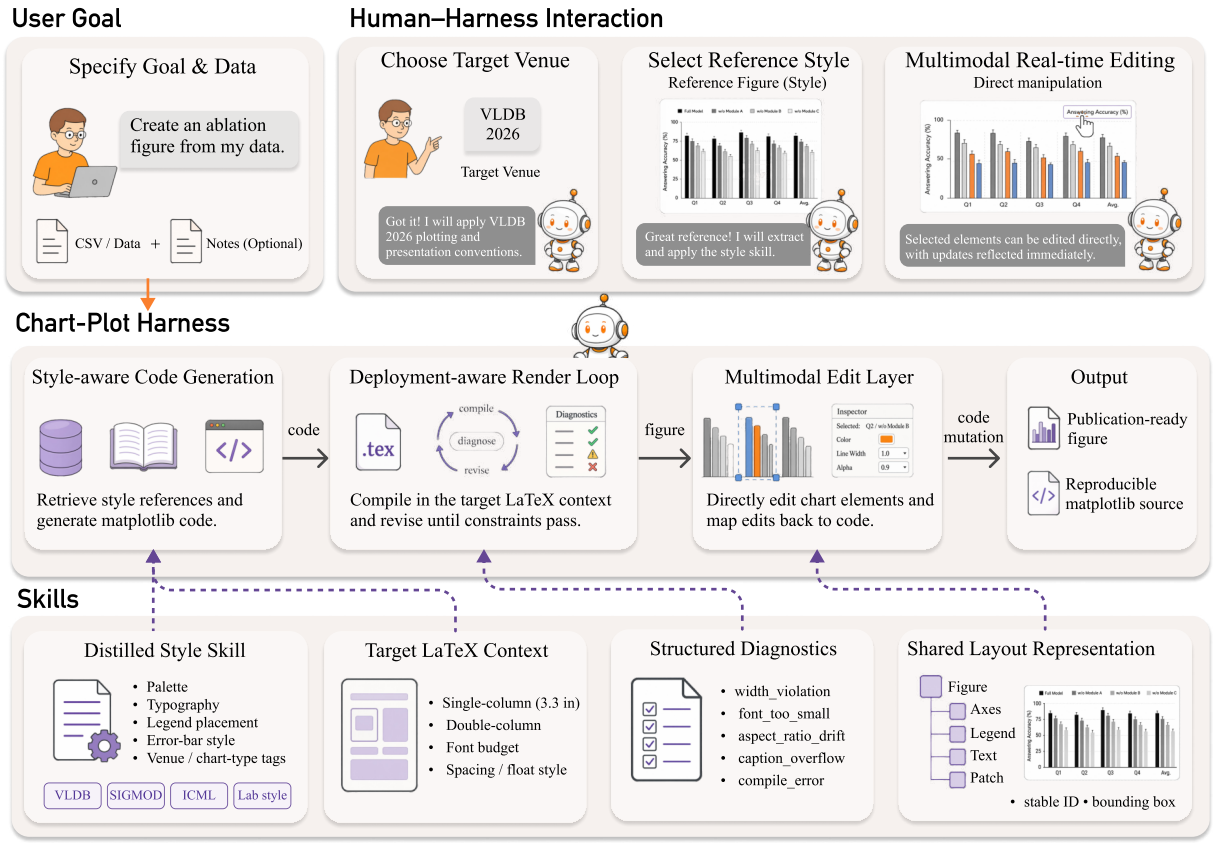

Figure 2: The chart-plot architecture. The author specifies a goal and data, selects a target venue and an optional reference style, and may edit the result through direct manipulation. The harness then runs three stages: style-aware code generation loads the distilled style skill and writes `matplotlib` code (§4.1); the deployment-aware render loop compiles the figure inside the target LaTeX context and revises the code until layout constraints are satisfied (§4.2); and the multimodal edit layer maps direct manipulations back to code mutations (§4.3). A shared layout representation with stable element identifiers and bounding boxes is reused across the deployment and edit stages.

Shared representation. After code generation, the system executes the `matplotlib` program in a sandbox and walks the resulting `Figure` object tree. Every element (axis, legend, text, patch, line) receives a stable identifier and a bounding box. This layout representation is shared across the deployment loop and the edit layer, so automated fixes and manual edits compose without conflict.

chart-plot closes the generation loop: it checks the output against learned standards, revises when constraints fail, and hands control to the author when the remaining gap is a matter of personal preference.

4 METHOD

4.1 Style-aware code generation

Top-venue charts follow community-specific conventions (e.g. low-saturation palettes at AI venues, hatched bars at systems venues), and a single generation prompt cannot capture these norms in sufficient detail. We treat style as a corpus distillation problem.

Our corpus is built by extracting figures from accepted papers at target venues. Each PDF is parsed into per-figure PNGs paired with their captions; the candidate figures are then human-curated for visual quality and chart-likeness, and the surviving entries are tagged with venue and chart type (ablation, scaling curve, training curve, etc.). The corpus grows monotonically as new venues accept papers.

The corpus is distilled into a textual *skill*: a markdown summary of the recurring visual conventions (palette, font sizing, legend placement, error-bar style) that is loaded into the codegen LLM's context at generation time. We follow the published skill convention [1] of attaching a self-contained, version-controlled instruction file alongside an agent's tool surface. The artifact is a single file the author can read, version, or override per venue.

A new venue or a lab-specific palette is captured by a new skill file rather than by retraining. Figure 3 visualises the effect across three chart types: a grouped-bar ablation, a paired-marker scaling line chart, and a paired-condition distribution comparison. In each row we compare the chart Claude writes from the same intent and data with (right) and without (left) the distilled VLDB style skill loaded into its context.

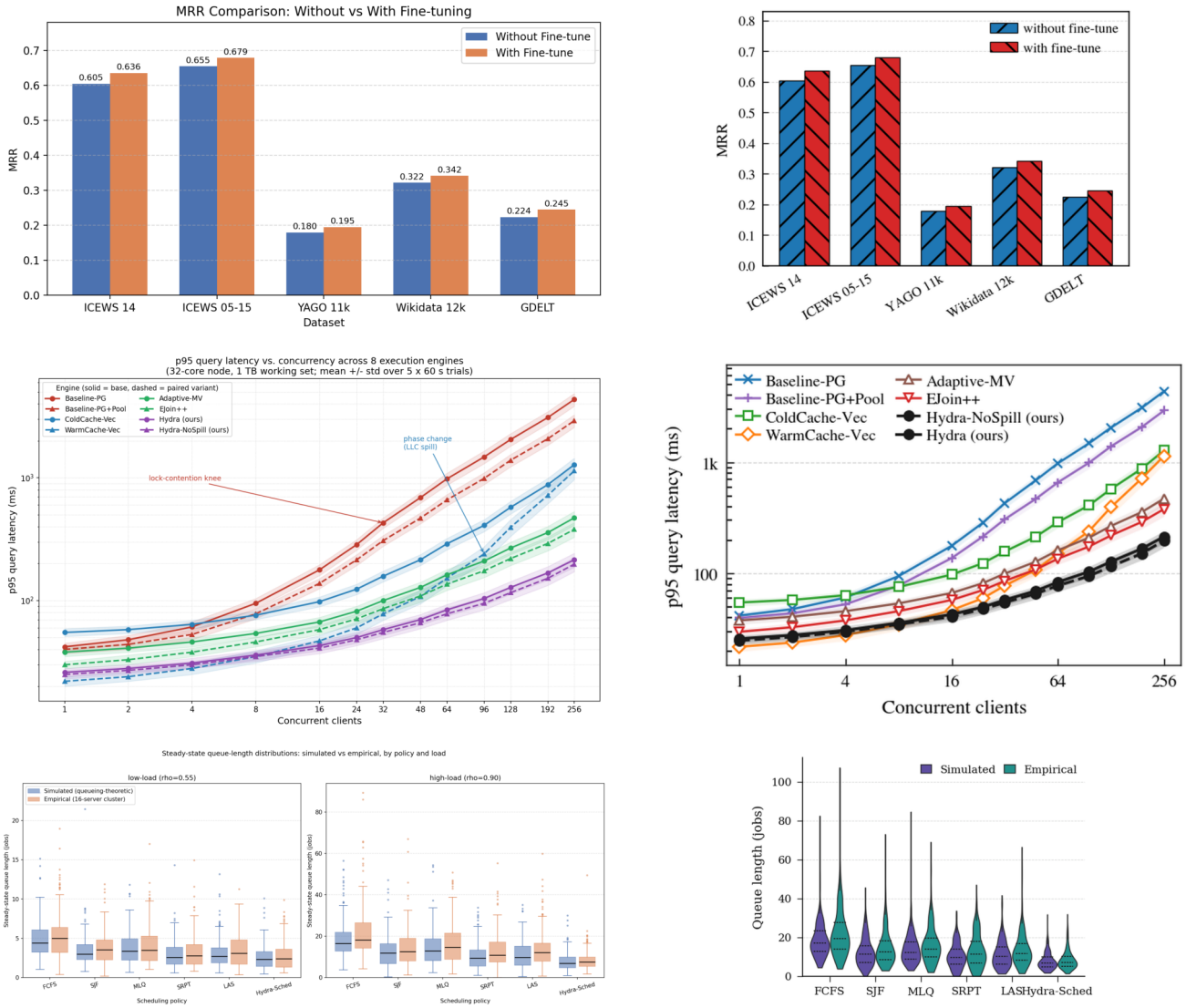

Figure 3: Three style cases (rows, top to bottom: Case 1 grouped-bar ablation; Case 2 scaling line chart; Case 3 paired-condition distribution comparison). Left column: the chart a baseline coding LLM (the default harness, no style skill loaded) produces from the same intent and data. Right column: the chart chart-plot produces with the distilled VLDB style skill loaded. The baseline outputs show recurring publication-quality issues: off-venue palettes, missing hatching or marker discipline, tick labels and legends that become unreadable once the chart is rendered at column width, and inconsistent error-bar styling. The right-column variants inherit the venue's visual conventions and remain legible at the same column width.

4.2 Deployment-aware render loop

Key insight. Charts are produced in isolation but consumed inside a paper column. We close this gap by compiling the chart *inside* the target LaTeX template, which serves two purposes. First, the system observes layout-level errors (column overflow, illegible fonts, aspect-ratio drift) that no standalone `matplotlib` render reveals, and feeds them back into a code-revision loop. Second, the chart now lives in the same rendering context in which the author will eventually edit it (§4.3); a single in-context adjustment produces the camera-ready result, instead of an iteration cycle between an isolated preview and the target paper.

Worked example. An oversized 8×6 inch chart compiled inside a 3.35-inch single column trips a width violation and a font-size violation. One revision regenerates the figure at 3.35 × 2.34 in and the loop converges (Figure 4).

The loop. The harness compiles the generated chart inside a template that mirrors the target paper's column width, font budget, and float style. After the compile, it measures the rendered figure's bounding box and produces structured diagnostics for any layout

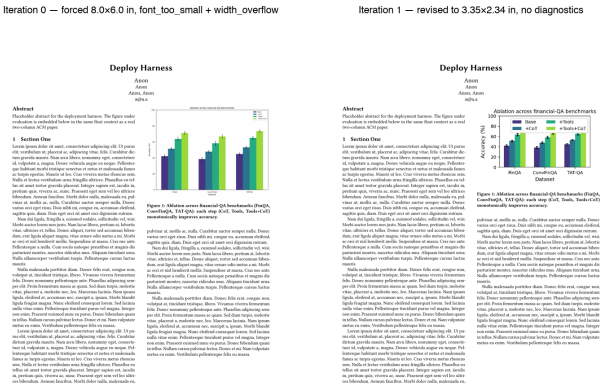

Figure 4: Deploy-loop trace, iteration 0 (left) and iteration 1 (right). The forced 8×6 in chart trips a width violation (8.0 in vs. 3.35 in column) and a font-size violation (effective ≈ 4.2 pt tick labels). One revision regenerates the figure at 3.35×2.34 in; all checks pass.

violation: compile failures, width or font issues, aspect-ratio drift, and caption overflow. Each diagnostic is converted into a concrete code-edit instruction (for instance, "increase tick-label font size to at least 6 pt") and fed back into the next round of code generation. The loop terminates on convergence, iteration cap, compile error, or model failure.

4.3 Multimodal edit layer

Key insight. Re-prompting an LLM with desired changes has two limits. First, free-text instructions like "move the legend a bit" rarely converge on what the author meant, because the preference is geometric, not semantic. Second, the LLM never sees the rendered chart; it sees only the code, so each edit is a hypothesis about the visual outcome, and the divergence compounds across iterations [15, 32]. We provide a multimodal input path instead: the author manipulates chart elements directly inside the same LaTeX environment used for deployment, so what they see during editing is what the camera-ready paper will show.

The interface. The system executes the generated code in a sandbox, walks the `matplotlib` `Figure` object tree, and assigns every element (axis label, legend, line, patch) a stable identifier and bounding box. A web interface overlays click-targets on the rendered figure; selecting an element opens a contextual inspector whose controls depend on the element kind: a legend exposes location, column count, font size, and frame toggle; a line exposes colour, width, style, and marker; a text element exposes content, size, and colour. Each control maps to a concrete code mutation (e.g. `ax.legend(loc='upper left', ncol=2)`); because edits target the source code rather than a rendered bitmap, every change is reproducible and composes with automated deployment fixes (Figure 5).

Figure 5: The edit layer, captured from the live web interface. (a) Element targets are overlaid on the rendered chart; the legend is currently selected (red highlight). (b) The property inspector for the selected legend exposes its addressable controls (location, font size, column count, frame); each control maps to a concrete `matplotlib` mutation that is re-applied through the deployment harness.

5 CASE STUDY AND EARLY RESULTS

We walk three chart cases through the chart-plot pipeline. Each case starts with an intent and data, runs through skill-prefixed codegen, and lands in a target VLDB `acmart` column. The three cases sweep chart types common to systems papers: a grouped-bar ablation (Case 1), a paired-marker scaling line chart (Case 2), and a paired-condition distribution comparison (Case 3). All three are shown side by side as the rows of Figure 3, baseline on the left and chart-plot output on the right.

5.1 Three cases

Case 1: grouped-bar ablation. The input compares MRR with vs. without fine-tuning across five datasets (ICEWS 14, ICEWS 05-15, YAGO 11k, Wikidata 12k, GDELT). The default Claude Code output (Figure 3, row 1 left) reads more like an AI-paper rendering than a systems-paper chart: a saturated palette, sans-serif labels, no hatching, and tick fonts that shrink below legibility once the chart is dropped into a single column. The chart-plot variant (row 1 right) inherits the venue's serif typography, muted Family-A palette, hatched bars, and a frameless compact legend.

Case 2: paired-marker scaling line chart. The intent asks how p95 query latency for eight execution engines changes as concurrent clients scale from 1 to 256. The baseline (row 2 left) draws eight equally weighted lines that compete for attention and overflow tick labels at column width. The chart-plot variant (row 2 right) pairs the two proposed variants (filled black) against six baselines (open coloured markers), uses log-log axes with mean-±-std error bands, and selects a column-aware `figsize`.

Case 3: paired-condition distribution comparison. The intent compares queue-length distributions from a simulator against empirical traces under high load for six scheduling policies. The baseline (row 3 left) emits a default boxplot per policy that obscures distribution shape and tail behaviour. The chart-plot variant (row 3 right) renders paired violins per policy (sim purple, emp teal) with embedded quartile ticks, surfacing the sim-vs-emp gap directly.

Together the three cases share a single observation: a default coding-LLM harness produces plausible `matplotlib` code, yet the resulting figure misses both the venue's visual conventions and the column-width legibility budget. The chart-plot harness fixes both: the style skill encodes the right defaults and the deployment loop keeps the result legible.

5.2 Observations

- **Default code-LLM output is plausible but not publication-ready.** Across all three cases the baseline chart compiles cleanly yet fails both the venue conventions and the column-width legibility test (Figure 3, left column). The chart-plot variant addresses both in a single pass.
- **Constraint violations are diagnosable.** When upstream code does fail a layout check, the deployment loop surfaces a named diagnostic with measured offending values (font size, width, aspect ratio) rather than a free-text vision-model critique, so the next codegen iteration knows what to fix.
- **Geometric edits cost zero LLM tokens.** Author edits in the inspector go through AST mutation; the alternative would be a full re-prompt of ~3000 tokens to describe the chart and the desired change.
- **Skill and deployment are a co-design.** When the skill already encodes a column-aware figure budget, the deployment loop converges immediately; it fires only when upstream code violates a constraint.

5.3 User study

To complement the case studies with feedback from real authors, we ran a small user study with four computer science researchers familiar with academic publishing. Participants rated chart-plot against a baseline coding LLM on a 5-point Likert scale across five questions, organised as one overall question and three component-specific groups:

- **Overall.** Q1: would you continue using the system for future submissions?
- **Style component.** Q2: does the figure match top-venue VLDB visual conventions? Q3: how aesthetically appealing is the figure overall?
- **Deploy component.** Q4: are tick labels, legends, and titles readable at column width?
- **Edit component.** Q5: how easy and precise is it to apply small geometric edits to the figure?

Q1–Q4 were rated by direct inspection of the three cases in §5 (the same figures the reader sees here), comparing each chart-plot output against a no-skill baseline rendering. Q5 was rated after a short live demo of the edit interface in which each participant performed a few precise edits on a chart-plot output in the browser.

Figure 6 summarises the results. chart-plot lifts mean ratings by 1.50–2.50 Likert points across all five questions; the largest gaps fall on column-width readability (Q4, 4.5 vs. 2.00) and continued-use intent (Q1, 4.5 vs. 2.50), and the smallest on aesthetic appeal (Q3, 4.25 vs. 2.75). The edit dimension (Q5) lands at 4.5 vs. 2.75 once participants have spent a few minutes inside the click-target inspector.

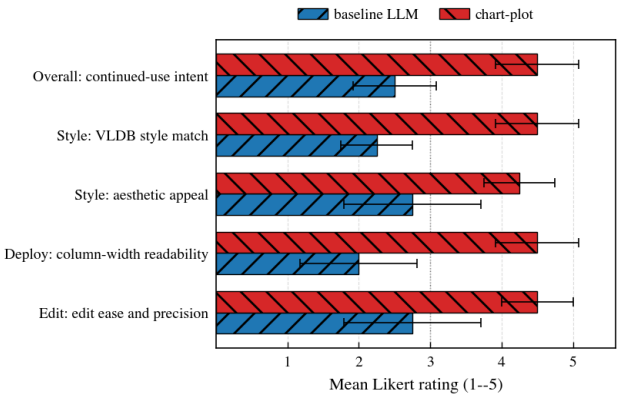

Figure 6: User study results (N=4 computer science researchers). Mean Likert rating per question with one-standard-deviation error bars, comparing a baseline coding LLM against chart-plot across five questions in four groups (Overall, Style, Deploy, Edit). This figure was itself generated end-to-end by the chart-plot skill under the same single-column `acmart` target as the rest of the paper.

Interview insights. A short post-task interview with each participant surfaced two recurring threads worth documenting. First, participants converged on a hybrid editing pattern: chat-style natural-language instructions worked best for *large* changes such as rewriting an annotation or restructuring a multi-axis layout, while the click-target inspector worked best for *small, precise* geometric edits. We accordingly recommend a hybrid multimodal editing workflow that combines both. Second, participants observed that chart-plot ships two harness environments (the standalone implementation used for the demo and a Claude Code skill packaging) and asked for parity with other coding agents such as OpenAI Codex; broadening the supported set of agent harnesses is the most direct lever for community adoption.

6 DISCUSSION AND CONCLUSION

We presented chart-plot, an agentic harness that closes the last mile of academic chart generation through three layers: a style-aware code generator, a deployment-aware render loop, and a multimodal edit layer.

What chart-plot distils today is the visual idiom of one broad community (computer-science papers). Every academic field carries its own conventions that a general-purpose LLM does not reproduce reliably from training data alone; financial charts, for example, annotate inflection points, regime changes, and event windows with ad-hoc text labels and shaded spans [8, 12]. Our next step is an open-source library of expert-distilled style skills across additional domains, packaged in the same skill format introduced in §4.1, so that academic chart generation does not stop at one community's gold standard.

REFERENCES

[1] Anthropic. 2025. Claude Skills: Reusable Capability Bundles for AI Agents. https://www.anthropic.com/news/skills. Online.

[2] Yiyu Chen, Yifan Wu, Shuyu Shen, Yupeng Xie, Leixian Shen, Hui Xiong, and Yuyu Luo. 2025. ChartMark: A Structured Grammar for Chart Annotation. *arXiv preprint arXiv:2507.21810* (2025).

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

[4] Richard Gerum. 2019. Pylustrator: code generation for reproducible figures for publication. *arXiv preprint arXiv:1910.00279* (2019).

[5] Shangding Gu. 2026. From Model Scaling to System Scaling: Scaling the Harness in Agentic AI. *arXiv preprint arXiv:2605.26112* (2026).

[6] Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. 2023. ChartLlama: A Multimodal LLM for Chart Understanding and Generation. *arXiv preprint arXiv:2311.16483* (2023).

[7] Sirui Hong et al. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *International Conference on Learning Representations*.

[8] Qi Jiang, Guodao Sun, Tong Li, Jingwei Tang, Wang Xia, Yunchao Wang, Li Jiang, and Ronghua Liang. 2025. AutoMA: Automated Generation of Multi-level Annotations for Time Series Visualization. In *IEEE Pacific Visualization Symposium (PacificVis)*. 80–90. https://doi.org/10.1109/PACIFICVIS64226.2025.00014

[9] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *International Conference on Learning Representations*.

[10] Boyan Li, Yiran Peng, Yupeng Xie, Sirong Lu, Yizhang Zhu, Xing Mu, Xinyu Liu, and Yuyu Luo. 2026. Deepeye: A steerable self-driving data agent system. *arXiv preprint arXiv:2603.28889* (2026).

[11] Junjie Li, Xi Xiao, Yunbei Zhang, Chen Liu, Lin Zhao, Xiaoying Liao, Yingrui Ji, Janet Wang, Jianyang Gu, Yingqiang Ge, Weijie Xu, Xi Fang, Xiang Xu, Tianchen Zhao, Youngeun Kim, Tianyang Wang, Jihun Hamm, Smita Krishnaswamy, Jun Huan, and Chandan K. Reddy. 2026. Agent Harness Engineering: A Survey. https://openreview.net/forum?id=3hXEPbG0dh Under review for TMLR.

[12] Ji-Feng Luo, Yuzhen Chen, Kaixun Zhang, Xudong An, Menghan Hu, Guangtao Zhai, and Xiao-Ping Zhang. 2025. Human-Centered Financial Signal Analysis Based on Visual Patterns in Stock Charts. *IEEE Transactions on Multimedia* 27 (2025), 4193–4205. https://doi.org/10.1109/TMM.2025.3535278

[13] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. 2018. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization and Computer Graphics* (2018).

[14] Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, et al. 2026. Code as Agent Harness. *arXiv preprint arXiv:2605.18747* (2026).

[15] Bo Pan, Yixiao Fu, Ke Wang, Junyu Lu, Lunke Pan, Ziyang Qian, Yuhan Chen, Guoliang Wang, Yitao Zhou, Li Zheng, Yinghao Tang, Zhen Wen, Yuchen Wu, Junhua Lu, Biao Zhu, Minfeng Zhu, Bo Zhang, and Wei Chen. 2025. VIS-Shepherd: Constructing Critic for LLM-based Data Visualization Generation. *arXiv preprint arXiv:2506.13326* (2025).

[16] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*.

[17] Wonduk Seo et al. 2025. VisPath: Automated Visualization Code Synthesis via Multi-Path Reasoning and Feedback-Driven Optimization. *arXiv preprint arXiv:2502.11140* (2025).

[18] Yinghao Tang, Tingfeng Lan, Xiuqi Huang, Hui Lu, and Wei Chen. 2025. SCORPIO: Serving the Right Requests at the Right Time for Heterogeneous SLOs in LLM Inference. *arXiv preprint arXiv:2505.23022* (2025).

[19] Yinghao Tang, Xueding Liu, Boyuan Zhang, Tingfeng Lan, Yupeng Xie, Jiale Lao, Yiyao Wang, Haoxuan Li, Tingting Gao, Bo Pan, Luoxuan Weng, Xiuqi Huang, Minfeng Zhu, Yingchaojie Feng, Yuyu Luo, and Wei Chen. 2026. IGenBench: Benchmarking the Reliability of Text-to-Infographic Generation. *arXiv preprint arXiv:2601.04498* (2026).

[20] Yinghao Tang, Yupeng Xie, Yingchaojie Feng, Tingfeng Lan, and Wei Chen. 2026. Demonstrating ViviDoc: Generating Interactive Documents through Human-Agent Collaboration. *arXiv preprint arXiv:2603.01912* (2026).

[21] Yinghao Tang, Yupeng Xie, Yingchaojie Feng, Tingfeng Lan, Jiale Lao, Yue Cheng, and Wei Chen. 2026. ViviDoc: Generating Interactive Documents through Human-Agent Collaboration. *arXiv preprint arXiv:2603.27991* (2026).

[22] Priyan Vaithilingam, Elena L Glassman, Jeevana Priya Inala, and Chenglong Wang. 2024. Dynavis: Dynamically synthesized ui widgets for visualization editing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–17.

[23] Chenglong Wang, Bongshin Lee, Steven M Drucker, Dan Marshall, and Jianfeng Gao. 2025. Data formulator 2: Iterative creation of data visualizations, with ai transforming data along the way. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–17.

[24] Xingyao Wang et al. 2025. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *International Conference on Learning Representations*.

[25] Zelin Wang, Yuanyuan Yin, Jien Wang, Haiyan Yan, Xuan Xie, and Yiqing Zheng. 2026. ggplotAgent: a self-debugging multi-modal agent for robust and reproducible scientific visualization. *Bioinformatics Advances* 6, 1 (2026), vbaf332.

[26] Luoxuan Weng, Yinghao Tang, Yingchaojie Feng, Zhuo Chang, Ruiqin Chen, Haozhe Feng, Chen Hou, Danqing Huang, Yang Li, Huaming Rao, Haonan Wang, Canshi Wei, Xiaofeng Yang, Yuhui Zhang, Yifeng Zheng, Xiuqi Huang, Minfeng Zhu, Yuxin Ma, Bin Cui, Peng Chen, and Wei Chen. 2025. DataLab: A Unified Platform for LLM-Powered Business Intelligence. In *IEEE International Conference on Data Engineering (ICDE)*. https://doi.org/10.1109/ICDE65448.2025.00326

[27] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155* (2023).

[28] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. HAIChart: Human and AI Paired Visualization System. *Proceedings of the VLDB Endowment* (2024). https://doi.org/10.14778/3681954.3681992

[29] Yupeng Xie, Zhiyang Zhang, Yifan Wu, Sirong Lu, Jiayi Zhang, Zhaoyang Yu, Jinlin Wang, Sirui Hong, Bang Liu, Chenglin Wu, and Yuyu Luo. 2025. VisJudge-Bench: Aesthetics and Quality Assessment of Visualizations. *arXiv preprint arXiv:2510.22373* (2025).

[30] Pengyu Yan et al. 2024. ChartReformer: Natural Language-Driven Chart Image Editing. In *ICDAR*.

[31] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*.

[32] Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, Zhiyuan Liu, Xiaodong Shi, and Maosong Sun. 2024. MatPlotAgent: Method and Evaluation for LLM-based Agentic Scientific Data Visualization. *arXiv preprint arXiv:2402.11453* (2024).

[33] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*.

[34] Fatemeh Pesaran Zadeh, Juyeon Kim, Jin-Hwa Kim, and Gunhee Kim. 2024. Text2Chart31: Instruction tuning for chart generation with automatic feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 11459–11480.

[35] Xuanle Zhao et al. 2025. ChartCoder: Advancing Multimodal Large Language Model for Chart-to-Code Generation. *arXiv preprint arXiv:2501.06598* (2025).

[36] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations*.

[37] Minjun Zhu, Zhen Lin, Yixuan Weng, Panzhong Lu, Qiujie Xie, Yifan Wei, Sifan Liu, Qiyao Sun, and Yue Zhang. 2026. AutoFigure: Generating and Refining Publication-Ready Scientific Illustrations. *arXiv preprint arXiv:2602.03828* (2026).

[38] Jonathan Zong, Dhiraj Barnwal, Rupayan Neogy, and Arvind Satyanarayan. 2020. Lyra 2: Designing interactive visualizations by demonstration. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2020), 304–314.