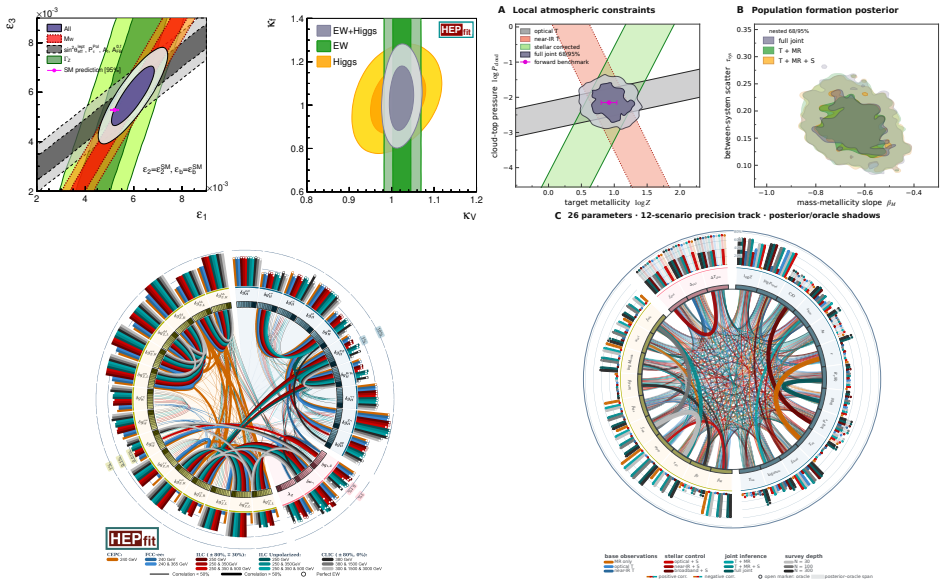
FIGMIRROR: GROUND IT, CODE IT, PLOT IT

Anonymous authors
Paper under double-blind review

ABSTRACT

Converting scientific figures into executable code has gained increasing attention, yet existing methods primarily focus on reproducing the reference figure itself. A more practical setting is to plot new data while preserving the visual style of a reference figure (e.g., color scheme and typography). Prior approaches mimic the reference through pixel-level optimization and struggle to carry its style to new data. We show that the key to this task lies in the coordinate grounding and coding capabilities present in modern computer-use models. We propose `FigMirror`, an agentic framework that unlocks these capabilities through *Grounded Measurement*, which locates visual elements by coordinates and measures their properties through executable code. We further introduce `PlotTwin-Bench`, an expert-curated benchmark with fine-grained code- and image-level style metrics. Experiments show that `FigMirror` consistently outperforms existing methods on reference-conditioned style transfer. All plots in this paper are generated by `FigMirror`, except those produced by other methods for comparison.



(a) Reference (de Blas et al., 2020, Fig. 1) (b) `FigMirror` with new target data

Figure 1: Reference-conditioned style transfer. `FigMirror` preserves the visual style of a reference scientific figure while adapting it to new target data.

1 INTRODUCTION

Producing a publication-quality scientific figure is laborious and challenging. The spacing, panel sizes, and fonts often require multiple rounds of manual adjustment before the figure looks right (Rougier et al., 2014). Researchers have traditionally used well-crafted figures from top venues as references and manually adopted their spacing, colors, and layouts. Multimodal models can now interpret

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

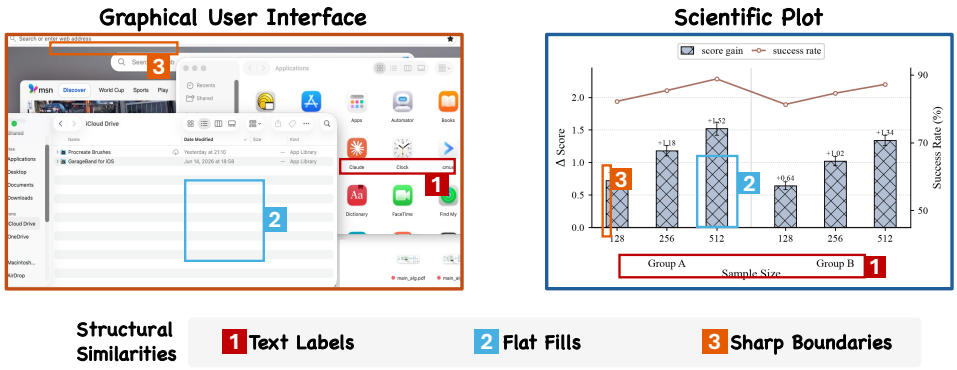


Figure 2: GUIs and scientific figures share localized, code-rendered elements with sharp boundaries, flat fills, and text. This shared structure allows coordinate grounding learned for GUI elements to localize ticks, legends, and marks in scientific figures for measurement and review.

such references and generate the corresponding plotting code. However, most existing chart-to-code methods focus on reproducing the reference figure itself at the pixel level, rather than transferring its visual style and layout to new data.

Chart-to-code generation predates capable general-purpose large models, and early work fine-tuned specialized models for this task. Reproduction provides a natural training signal: the discrepancy between the rendered figure and the reference. ChartLlama (Han et al., 2023) is trained in this manner. General-purpose models have since made substantial progress in visual understanding and reasoning, yet the same recipe persists. METAL (Li et al., 2025) adopts a multi-agent framework in which a generator writes the code and a critic repeatedly compares the rendered output against the reference and revises the code accordingly. This effectively moves the same optimization process to inference time. In either case, the objective remains reproduction. By contrast, transferring the visual style of a reference figure to a researcher's own data has received little attention as a task in its own right.

A figure entangles its data with its style. We argue that transferring the style requires disentangling the two in two steps. The first is *identification*: locating the visual attributes that carry the transferable style, e.g., a color series or the spacing between subplots. The second, more critical, is *measurement*: computing each attribute's exact value, e.g., a hex code or a width in points, from the element's pixels. Modern general-purpose models have already learned the first step, just not from plotting. They are trained at scale for computer use (Anthropic, 2026; OpenAI, 2026; Bai et al., 2025; Wang et al., 2025). There, an agent operates software on behalf of a user, and every action begins with the coordinates of its on-screen target, a button or a menu entry. This demand instills coordinate grounding. A scientific plot, built from discrete, sharp-edged elements rendered by code (Figure 2), is as amenable to coordinate grounding, and as readily as a GUI. Yet existing plotting tasks do not explicitly invoke this skill or capability. When asked to match a reference figure, the model still tends to estimate visual attributes by eye.

To address this, we propose *Grounded Measurement* to redirect the skill from acting to measuring. The model grounds the element that carries a style attribute, and where a click would follow, a short program reads the exact value, a line's width or a series' color. One measurement, however, resolves one attribute. A transfer needs every attribute found, measured, applied, and re-checked on the rendered figure. Our proposed framework, `FigMirror`, organizes this work as a **Drawer–Reviewer** loop. The Drawer decomposes the reference into style attributes, measures each, and renders the user data into a candidate. The Reviewer checks the candidate against the reference visually. The two iterate until the candidate matches the reference style.

As no benchmark is dedicated to figure style transfer, we build `PlotTwin-Bench`. Its references come from two sources: figures replotted by hand from top venues and journals, and existing plotting code that a pipeline enriches into more complex and visually appealing figures. We extract scoring criteria from each reference and score candidates through two channels. The code channel runs an exact check on the attributes where the reference departs from a default plot. The vision channel catches problems that only appear once drawn, such as overlapping text or clipped labels.

Our contributions are:

- We formulate reference-conditioned style transfer for scientific figures and identify precise style measurement as its core challenge.
- We propose *Grounded Measurement*, which repurposes coordinate grounding to read each style attribute from the reference with code, and realize it in `FigMirror`, an agentic Drawer–Reviewer framework.
- We build `PlotTwin-Bench`, the first benchmark designed for figure style transfer, with expert-curated figure–code pairs and per-reference scoring criteria that evaluate style from both code and rendered image. `FigMirror` consistently outperforms prior methods.

2 Related Work

Chart-to-code generation produces plotting code from a chart image. The task grew out of chart understanding, which treated figures as question-answering and derendering targets (Kafle et al., 2018; Masry et al., 2022; Kantharaj et al., 2022; Liu et al., 2023), and turned generative once multimodal models could emit executable programs. ChartLlama (Han et al., 2023) LoRA-tunes (Hu et al., 2021) a multimodal LLaMA (Touvron et al., 2023) to redraw a chart from its image, and successors scale the recipe with larger corpora, code-oriented backbones, and reinforcement learning (Zhao et al., 2025; Tan et al., 2025). As general-purpose models matured, training gave way to inference-time self-refinement (Madaan et al., 2023; Shinn et al., 2023): a reviewer compares the render against the reference, and a drawer revises the code until the two agree (Li et al., 2025; Xu et al., 2025). Benchmarks co-evolved with the methods, pairing a reference figure with its code and scoring how faithfully a candidate reproduces it (Yang et al., 2025; Wu et al., 2025a).

Computer-use agents complete tasks by operating software through its interface, on the web (Deng et al., 2023; Zhou et al., 2024) and across full desktops (Xie et al., 2024). Acting on a screen requires knowing where its elements are, so the line invests heavily in GUI grounding: SeeClick (Cheng et al., 2024) pre-trains on grounding data, UGround (Gou et al., 2025) scales a universal grounding model on synthetic web pages, OS-ATLAS (Wu et al., 2025b) builds a grounding corpus across platforms, and agent models such as CogAgent (Hong et al., 2024) and UI-TARS (Qin et al., 2025) fold grounding into end-to-end action, the predicted coordinates feeding clicks, drags, and keystrokes. The direction is now mainstream: the latest frontier models ship computer use as a built-in capability and benchmark it on OSWorld alongside coding and reasoning (Anthropic, 2026; OpenAI, 2026).

Neither line reaches our setting. Chart-to-code methods and their benchmarks target reproduction; plotting new data appears at most as an auxiliary test case (Yang et al., 2025). No method reads a reference's style attribute by attribute, and no benchmark scores how faithfully a style carries to new data. Computer-use agents train the grounding that such reading needs, but apply it only to on-screen actions. Reference-conditioned style transfer has yet to be studied as a task in its own right.

3 Method

`FigMirror` is an agentic framework for this setting. A Style Checklist organizes which style attributes to inspect on a reference; Grounded Measurement then locates the visual element that carries each attribute and reads its exact value from the reference with code (Section 3.1). A Drawer–Reviewer loop organizes the generation (Section 3.2): the Drawer resolves every open attribute this way and renders the user data into a template, the Reviewer checks the template against the reference visually and routes attributes into a Revision and a Preserve List, and the lists update the checklist for the next round. The loop ends when the Revision List is empty; Figure 3 summarizes the process.

3.1 Grounded Measurement

A scientific figure shares its anatomy with a graphical interface (Figure 2): both are rendered from code into discrete elements with sharp boundaries and flat fills. On such pixels, general-purpose models are trained to ground a referred element (Cheng et al., 2024; Gou et al., 2025; Xie et al., 2024): given an image I and an element e, such as a button or a menu entry, they return its image

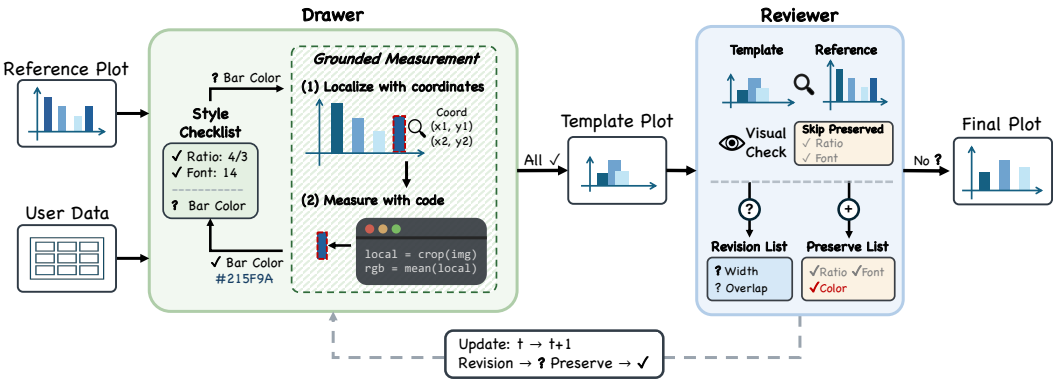


Figure 3: The FigMirror pipeline. The Drawer uses Grounded Measurement to resolve open style attributes and render the target data. The Reviewer compares the draft with the reference and assigns attributes to the Revision or Preserve List. These lists guide subsequent iterations until no revisions remain. The symbols ? and ✓ denote open and resolved attributes, respectively.

coordinates $(x, y) = g(I, e)$. An axis tick or a legend entry grounds just as well, so we reuse this coordinate interface on the reference figure.

Grounded Measurement builds on the reused interface. ***(1) Localize.*** For a style attribute a, the model is queried for the coordinates that delimit the visual element carrying a, the two corners (x_1, y_1) and (x_2, y_2) that span a region r_a of the reference. ***(2) Measure.*** A short program ρ_a computes the attribute value v_a from that region,

$$v_a = \rho_a\big(I[r_a]\big), \tag{1}$$

where $I[r_a]$ denotes the local crop used to measure a. The two steps appear as steps (1) and (2) in the Drawer part of Figure 3.

We then observe that the measurement programs are supplied just as readily. Code is the modality frontier models optimize hardest (Wang et al., 2024), and the few lines that read one attribute sit well within this capability. The lines differ by attribute, and the model fits them to what it sees: a mean over a flat fill recovers a bar's color, a thin anti-aliased stroke asks instead for the modal pixel, a run test along a stroke tells dashed from solid. One primitive thus spans the attribute space, from continuous values such as colors, widths, and aspect ratios to discrete ones such as the presence of grid lines. The same coordinate interface also supports review. After a draft is rendered, the model can mark the region where a style attribute was measured or applied incorrectly. This turns a vague visual critique into a local repair target.

3.2 The Drawer–Reviewer Loop

A single measurement reads one attribute; a figure has dozens. The Drawer therefore maintains a Style Checklist, composed of an open set $\mathcal{O}$ and a resolved set $\mathcal{R}$. The checklist is initialized under prompt guidance, where the Drawer scans the reference against a predefined attribute list and collects the attributes the figure exhibits, all open at the start. Each open attribute is then resolved by Grounded Measurement and moved to $\mathcal{R}$ with its value v_a. Once $\mathcal{O}$ is empty, the Drawer writes plotting code for the user data using the resolved style values and renders a draft figure.

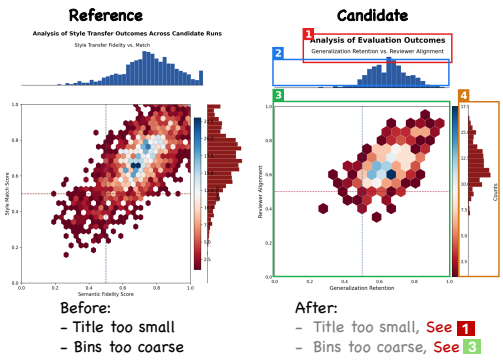


Figure 4: Grounded Reviewer feedback. Plain-text feedback can leave the repair target ambiguous. Linking each comment to a marked region identifies the relevant element and guides the Drawer to correct the local mismatch while preserving attributes that already match the reference.

The Reviewer gives this draft a fresh visual check. It receives a side-by-side view of the reference and the draft for global comparison, together with separate high-resolution views for local details.

The side-by-side view exposes layout and balance errors. The high-resolution views expose small failures such as text overlap or misplaced guides. As illustrated in Figure 4, the Reviewer returns feedback items (b_j, n_j), where b_j is a marked region and n_j is a short note. A sentence such as "the label is too close" leaves the Drawer guessing which label and which mark are involved. A marked region tells the Drawer exactly what the critique refers to.

The loop alternates a stateful Drawer and a stateless Reviewer. In each iteration, the Drawer edits the previous code using two lists from the last review. The Preserve List contains accepted style choices and grows monotonically across iterations. The Revision List contains localized mismatches and returns the corresponding attributes to $\mathcal{O}$. The Reviewer re-audits the full draft in every round. This separation lets the Drawer preserve useful construction state while the Reviewer judges only the rendered result. The loop stops when a review produces an empty Revision List, and the current draft becomes the final figure.

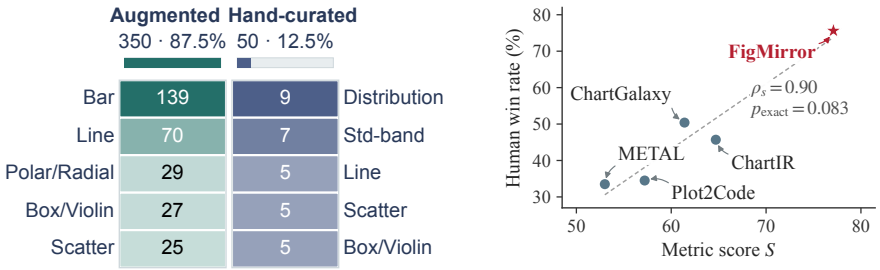
Implementation. `FigMirror` is packaged as a *skill*, a self-contained instruction bundle that an agentic coding harness loads at run time. The skill carries the full procedure (the Style Checklist, Grounded Measurement, and the Drawer–Reviewer loop); the harness supplies the model, code execution, and image access. This separation keeps `FigMirror` portable across harnesses.

4 Benchmark and Evaluation

4.1 Benchmark Construction

Data sources. Existing chart-to-code benchmarks mostly focus on reproduction, so we build `PlotTwin-Bench`. It targets real-world use: a reference figure is worth mirroring when its visual design is worth reusing and its structure is easier to specify with an image than with a text prompt. We therefore select references that are visually appealing and structurally rich. We draw on two sources. The first is hand-curated. We select complex, visually appealing figures from papers in top venues, then replot each figure by hand to obtain an aligned figure–code pair. This source contains 50 figures. The second source scales this up. We start from existing plotting code from ChartMimic (Yang et al., 2025) and use an LLM-based rewriting pipeline to produce more complex and visually appealing references (details and examples in Appendix C.2). A filter then removes the figures that fall short in either property, leaving 350. Together the two sources give 400 references across 12 chart types, including grouped bars, multi-panel line plots, and heatmaps. Figure 5a shows the five most common chart types in each source. Appendix C.1 reports the full composition.

Style-transfer task. Each task pairs a reference figure with target data and asks the model to visualize the data in the reference's style. To construct the target data, a generator sees only the reference image, invents a plausible scientific story in a different domain, and derives a dataset from that story. It then changes one to three aspects of the dataset while keeping it compatible with the reference's chart and panel structure. The generator performs a final check for structural consistency and numerical plausibility; Appendix C.3 gives the full procedure.



(a) (b)

Figure 5: Benchmark composition and evaluator alignment. (a) Five most common chart types in each reference source. (b) Automated score S versus human win rate across the five methods.

4.2 EVALUATION

Direct VLM scoring tends to be lenient for scientific figure style transfer (Zheng et al., 2023). Most figures share standard axes, marks, and labels, so a holistic comparison can underweight the few choices that distinguish a reference and give visibly different candidates similar scores. Our evaluator focuses on these reference-specific choices through two complementary channels. The code channel identifies departures from an average plotting style, while the vision channel finds important mismatches in the rendered figure.

The code channel scores the attributes that set a reference apart from an average figure. We use the Matplotlib (Hunter, 2007) defaults as a fixed and reproducible stand-in for this average. For a style attribute a, let v_a be its reference value, d_a its default value, and $\hat{v}_a$ its candidate value. `PlotTwin-Bench` pairs every reference with its code. The evaluator therefore reads v_a from the reference code and $\hat{v}_a$ from the candidate code, and both values are exact. During generation, the method sees only the rendered reference and measures from pixels (Section 3.1). The scored set is

$$\Delta(I) = \{a \mid v_a \neq d_a\}. \tag{2}$$

For each $a \in \Delta(I)$, the candidate receives full credit when $\hat{v}_a$ matches v_a, zero credit when it is no closer to v_a than d_a, and proportional credit in between. We use exact agreement for categorical choices and normalized distance for continuous values. Let $q_a \in [0, 1]$ denote this credit. The code score is

$$S_{\text{code}} = \frac{100}{|\Delta(I)|} \sum_{a \in \Delta(I)} q_a. \tag{3}$$

Code does not capture every distinctive choice. Layout, spacing, alignment, and visual hierarchy are often read more clearly from the rendered figure. The vision channel uses a VLM to identify important reference choices that the candidate failed to reproduce and records them as evidence-grounded visual defects. Each finding must cite visible evidence through bounding boxes in the reference and candidate images and, when applicable, the relevant code lines. The vision score S_{vision} starts at 100 and deducts 5, 10, or 25 points for each minor, major, or critical defect. The evidence requirement constrains unsupported findings and makes every deduction auditable.

The two scores cover complementary evidence on the same scale. We combine them as

$$S = 0.35 S_{\text{code}} + 0.65 S_{\text{vision}}. \tag{4}$$

The weights are fixed a priori: readers judge the rendered figure, so the vision channel receives the larger weight. We use the same weights in all experiments and report both channel scores alongside S. A human study suggests S aligns with human judgment (Figure 5b); Appendix B.2 reports the protocol and statistics.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate on 150 of the 400 references: all 50 hand-curated figures and 100 sampled at random from the augmented source; the exact subset is included in the benchmark release. Each method produces a self-contained plotting script, which the evaluator in Section 4 takes as input. In the main experiments, both the evaluator and every method's underlying model are GPT-5.5 at x-high reasoning effort, and `FigMirror` runs as a skill in the Codex harness (Section 3). Appendix A repeats the comparison with Claude Opus 5 in Claude Code and Grok 4.6 in Grok Build, and with GPT-5.6 (sol, x-high) in Codex. Appendix B.3 reports the rendering environment, harness configuration, and iteration budget; Appendix E gives the prompt bundle.

Baselines. We compare `FigMirror` with four external chart-to-code baselines: Plot2Code (Wu et al., 2025a), a one-shot chart-to-code method; METAL (Li et al., 2025), an iterative feedback method for chart code generation; ChartGalaxy-Prompt (Li et al., 2026, hereafter ChartGalaxy), the prompt-based code-generation recipe released with the dataset of the same name; and ChartIR (Xu et al., 2025), a method that repairs generated code over successive rounds. We extract each reference's scoring criteria once and share them across all methods.

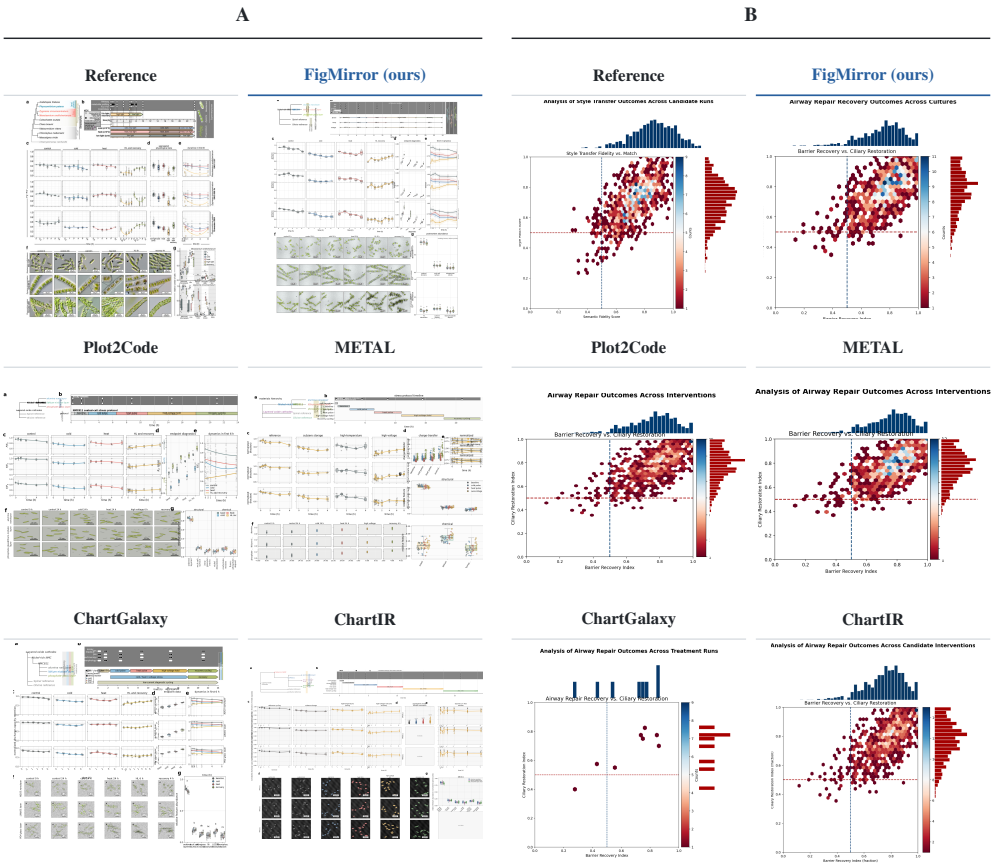


Figure 6: Style transfer on two references. Each group compares `FigMirror` and four baselines against the reference. In A, `FigMirror` preserves the panel hierarchy and relative spacing of a dense composition. In B, it matches the joint hexbin layout, marginal histograms, threshold lines, color scale, and typography.

5.2 Main Results

Table 1 reports the main style-transfer comparison on `PlotTwin-Bench`, with the hand-curated and augmented splits shown separately. `FigMirror` has the best combined score on both, leading the strongest baseline, ChartIR, by 11.4 points on the hand-curated split (72.7 versus 61.3) and by 6.1 on the augmented split (76.4 versus 70.3). It also leads in both channels on both splits, so the lead does not depend on the weights in S. Iteration alone does not order the baselines. ChartIR improves over the one-shot Plot2Code, while METAL, also iterative, stays at or below Plot2Code's level. Its critic is built for reproduction, and once the data differ, whole-image comparison pulls the draft toward the reference's data rather than its style.

The two margins trace back to where each split's styles come from. Augmented references are rewritten from existing plotting code, and a strong code model recovers much of their style unaided. The three leading methods all exceed 80 in the code channel, leaving the vision channel to separate them (9.2 points). Hand-curated references carry choices made by paper authors, which must be read from the image. On this split `FigMirror` leads in both channels, by 5.8 points in code and 13.3 in vision. These are the references the benchmark is built around, designs worth reusing that the model cannot guess.

Figure 6 compares five methods on two references. Within each group, every method receives the same reference and target data. Appendix A.3 shows additional transfers from Claude Code and Grok Build.

Table 1: Main results on scientific figure style transfer. We report S_{code}, S_{vision}, and the combined score S on 50 hand-curated and 100 augmented references from `PlotTwin-Bench`. All methods use GPT-5.5 at x-high reasoning effort. Higher scores are better, and the best results are bold.

	Hand-curated			Augmented		
Method	S_{code}	S_{vision}	S	S_{code}	S_{vision}	S
`FigMirror`	**64.4**	**77.2**	**72.7**	**84.0**	**72.3**	**76.4**
ChartIR (Xu et al., 2025)	56.6	63.9	61.3	83.6	63.1	70.3
ChartGalaxy (Li et al., 2026)	58.6	58.4	58.5	80.7	58.1	66.0
Plot2Code (Wu et al., 2025a)	53.4	56.8	55.6	64.9	43.0	50.7
METAL (Li et al., 2025)	53.7	50.5	51.6	66.2	43.5	51.4

Table 2: Ablations on style transfer. We implement each variant in the Codex harness by modifying the corresponding skill prompts and tool access. The variant without the Reviewer retains the Drawer's self-checks.

Variant	S_{code}	S_{vision}	S
`FigMirror`	64.4	77.2	72.7
w/o Reviewer	62.7	74.9	70.6
w/o Grounded Measurement	67.4	70.0	69.1
w/o `FigMirror` skill (naked Codex)	60.4	61.4	61.1

5.3 Ablations

Table 2 removes the two parts of `FigMirror` we expect to matter most, then the skill as a whole. The first ablation removes the Reviewer and keeps the Drawer's drafts and self-checks. The second removes Grounded Measurement from the feedback loop. The third, naked Codex, drops the skill and prompts the same harness directly. All three use the style-transfer setting.

Each ablation lowers the combined score, and the loss grows with what is removed. Removing the Reviewer costs 2.1 points. Mismatches that get past the Drawer's self-checks reach the final draft uncorrected, and both channels drop about 2 points. Removing Grounded Measurement costs 3.6. The Drawer estimates attribute values by eye. The Reviewer still boxes the resulting mismatches, but the corrections are also made by eye. The surviving drift costs 7.2 points in the vision channel, while the code channel edges up. Naked Codex costs 11.6. It drafts by eye alone and never measures the reference, so both channels fall and the score drops to the level of the external baselines.

5.4 Iteration Budget

We test whether additional Drawer and Reviewer rounds improve style transfer under a fixed Codex configuration (Table 3).

The combined score rises with the budget, from 71.8 at one iteration to 72.7 at three and 74.3 at five. At three iterations, the vision score increases by 1.4 points while the code score remains essentially unchanged. Five iterations improve both channels. We use three in the main experiments to limit inference cost.

Table 3: Iteration budget in the Codex setting. We vary the maximum number of iterations while keeping the model, data, and evaluation fixed.

Max iterations	S_{code}	S_{vision}	S
1	64.5	75.8	71.8
3 *(default)*	64.4	77.2	72.7
5	67.0	78.3	74.3

5.5 Mechanism-Level Case Study

Figure 7b follows one transfer run through three iterations. Each review binds a mismatch to a box, and the next iteration edits inside the boxes while unboxed attributes, held in the Preserve List, carry over unchanged.

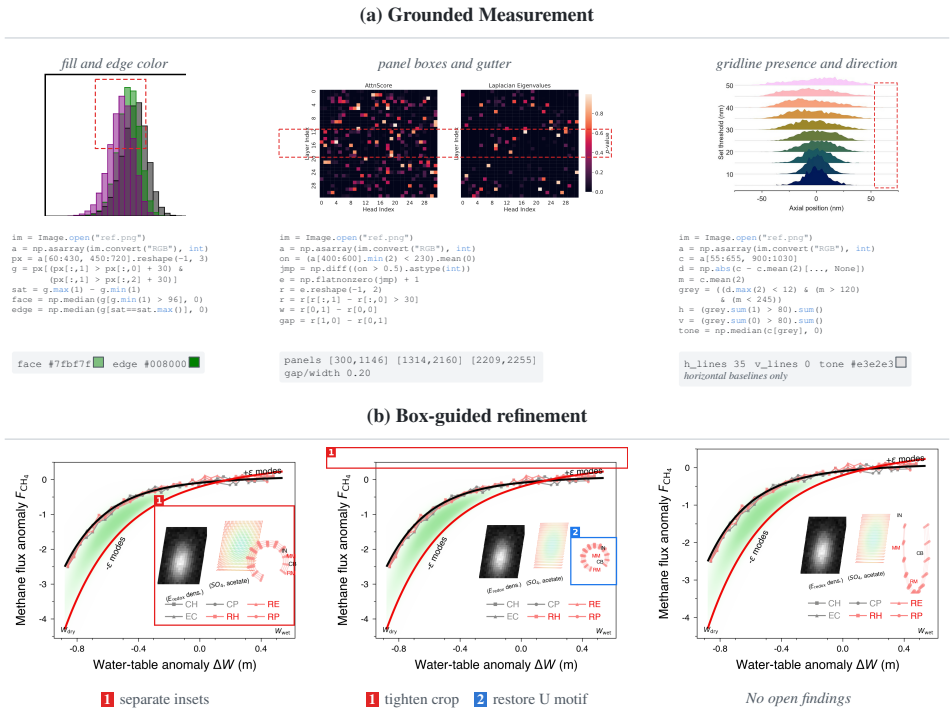


Figure 7: Grounded Measurement and box-guided refinement. (a) Three measurements from separate runs. Dashed boxes mark the sampled regions, with the corresponding probe code and returned values below. (b) Three iterations of one transfer run. The annotations identify local revisions applied in the next draft, while accepted attributes remain in the Preserve List.

The run makes the localization argument of Section 3.2 concrete. The first review marks the overlapping insets, and iteration 1 separates them: the box turns a composition-wide search into a local edit.

Because the Reviewer is stateless, each round re-audits the full draft, and subtler mismatches surface once dominant ones are cleared. With the insets separated, the next review boxes the broad top margin and the compressed U-shaped motif; iteration 2 fixes both, and the final review returns no boxes. Every boxed defect is resolved in the following iteration, and no repair disturbs an attribute already accepted. Panel (a) shows the values that feed these repairs: each probe returns the exact value the Drawer commits to the checklist.

6 Conclusion

We study reference-conditioned style transfer for scientific figures. Because the target data differ from the reference, transferable style cannot be assessed through whole-image matching; its elements must instead be localized, measured, and reapplied. Our main insight is that scientific figures share the code-rendered structure of graphical interfaces: both are built from discrete elements with sharp boundaries and flat fills. The coordinate grounding that lets a model point to a button or a menu entry therefore transfers to plot elements such as ticks, legends, and marks.

We turn this insight into Grounded Measurement, which locates each style element and reads its value with code, and we organize generation as a Drawer–Reviewer loop that makes every style attribute explicit, measurable, and revisable. To evaluate this setting, we build `PlotTwin-Bench`, the first benchmark for scientific figure style transfer, and score each candidate from both the code and the rendered image against per-reference scoring criteria. In every harness and model we test, `FigMirror` leads the external baselines in the combined score. We hope this formulation makes figure style a measurable and reusable design choice.

AI USE STATEMENT

Evaluation. We used AI tools for automated evaluation of generated figures, including analysis of plotting code and rendered images (Section 4). We also used AI to construct the evaluation benchmark by augmenting reference plotting scripts and generating synthetic target datasets (Appendices C.2 and C.3). Appendix B.2 reports a human preference study assessing the alignment of the automated evaluator with human judgments.

Writing assistance. We used AI tools to polish the wording of selected passages in the manuscript.

We did not use generative AI tools for the other tasks that require disclosure under the ICLR 2027 AI Policy for Authors. The authors reviewed all AI-assisted text and take responsibility for the final manuscript, including its claims and results.

ETHICS STATEMENT

A scientific figure carries an argument through data and the way that data are presented. `FigMirror` helps a user transfer a presentation style after the user has chosen a suitable reference; it does not decide whether that presentation fits the data. The user stays responsible for the substance of the figure, including the axis scales, labels, units, legends, and captions, and for checking that the visual encoding supports the underlying claim.

Our benchmark is built from licensed sources. In constructing it, we select reference figures from sources with licenses that permit research use and retain their provenance during curation. When applying `FigMirror` outside the benchmark, users should respect the license and attribution terms of any reference figure and treat the generated figure as an auditable draft rather than publication-ready evidence.

REPRODUCIBILITY STATEMENT

Section 3 describes Grounded Measurement and the Drawer–Reviewer loop. Section 4 defines the benchmark and scoring protocol. Appendices C.1, C.2, and C.3 document the reference composition, enrichment pipeline, and target-data construction. Appendices B.1 and B.2 provide the evaluation and human-study protocols; Appendix B.3 describes the experimental configuration. The core prompt excerpts are included in Appendix E.

REFERENCES

Anthropic. Introducing Claude Opus 5. *Anthropic blog*, 2026. URL `https://www.anthropic.com/news/claude-opus-5`.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

Jorge de Blas, Debtosh Chowdhury, Marco Ciuchini, Antonio M Coutinho, Otto Eberhardt, Marco Fedele, Enrico Franco, G Grilli Di Cortona, Víctor Miralles, Satoshi Mishima, et al. Hepfit: a code for the combination of indirect and direct constraints on high energy physics models. *The European Physical Journal C*, 80(5):456, 2020.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Boyu Gou, Demi Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations*, volume 2025, pp. 30851–30883, 2025.

Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. Chartllama: A multimodal llm for chart understanding and generation. *arXiv preprint arXiv:2311.16483*, 2023.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14281–14290. IEEE, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3): 90–95, 2007.

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5648–5656. IEEE, 2018.

Shankar Kantharaj, Rixie Tiffany Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. Chart-to-text: A large-scale benchmark for chart summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4005–4023, 2022.

Bingxuan Li, Yiwei Wang, Jiuxiang Gu, Kai-Wei Chang, and Nanyun Peng. Metal: A multi-agent framework for chart generation with test-time scaling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30054–30069, 2025.

Zhen Li, Duan Li, Yukai Guo, Xinyuan Guo, Bowen Li, Lanxi Xiao, Shenyu Qiao, Jiashu Chen, Zijian Wu, Hui Zhang, et al. Chartgalaxy: A dataset for infographic chart understanding and generation. In *International Conference on Learning Representations*, volume 2026, pp. 52484–52546, 2026.

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. Matcha: Enhancing visual language pretraining with math reasoning and chart derendering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12756–12770, 2023.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Ahmed Masry, Jia Qing Tan, Shafiq Joty, Enamul Hoque, et al. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pp. 2263–2279, 2022.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. *OpenAI blog*, 2026. URL `https://openai.com/index/gpt-5-6/`.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Nicolas P Rougier, Michael Droettboom, and Philip E Bourne. Ten simple rules for better figures. *PLoS computational biology*, 10(9):e1003833, 2014.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Wentao Tan, Qiong Cao, Chao Xue, Yibing Zhan, Changxing Ding, and Xiaodong He. Chartmaster: Advancing chart-to-code generation with real-world charts and chart similarity reinforcement learning. *arXiv preprint arXiv:2508.17608*, 2025.

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

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3006–3028, 2025a.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: Foundation action model for generalist gui agents. In *International Conference on Learning Representations*, volume 2025, pp. 5090–5108, 2025b.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Chengzhi Xu, Yuyang Wang, Lai Wei, Lichao Sun, and Weiran Huang. Improved iterative refinement for chart-to-code generation via structured instruction. *arXiv preprint arXiv:2506.14837*, 2025.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, et al. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. In *International Conference on Learning Representations*, volume 2025, pp. 26590–26646, 2025.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. Chartcoder: Advancing multimodal large language model for chart-to-code generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7333–7348, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

APPENDIX

APPENDIX CONTENTS

A Additional model and harness results 13
B Evaluation and experimental details 19
C Benchmark construction 20
D Limitations 21
E Prompt bundle 22

A ADDITIONAL MODEL AND HARNESS RESULTS

A.1 GPT-5.6 (SOL, X-HIGH)

Table 4 repeats the comparison of Table 1 with GPT-5.6 (sol, x-high) as the underlying model of every method, on 50 cases. Naked Codex prompts the harness without the `FigMirror` skill, as in Table 2.

Table 4: Additional results with GPT-5.6 (sol, x-high) on 50 cases. We report the code score S_{code}, the vision score S_{vision}, and their weighted combination $S = 0.35S_{\text{code}} + 0.65S_{\text{vision}}$. Higher scores are better, and the best results are bold.

Method	S_{code}	S_{vision}	S
`FigMirror`	**68.3**	**78.9**	**75.2**
Naked Codex	64.5	71.2	68.9
Plot2Code (Wu et al., 2025a)	51.0	62.1	58.2
METAL (Li et al., 2025)	53.2	63.3	59.8
ChartGalaxy (Li et al., 2026)	62.3	69.7	67.1
ChartIR (Xu et al., 2025)	65.7	75.4	72.0

`FigMirror` has the highest reported score in both channels and the highest combined score, $S = 75.2$. It leads ChartIR, the strongest external baseline in this setting, by 3.2 points and naked Codex by 6.3 points in S.

A.2 CLAUDE CODE AND GROK BUILD

`FigMirror` is a skill bundle that a coding-agent harness loads (Section 3). Table 5 reports the five methods of Table 1 on 50 cases in two other harnesses, Claude Code with Claude Opus 5 and Grok Build with Grok 4.6, scored as in Section 4.

Table 5: Portability across coding-agent harnesses. We report S_{code}, S_{vision}, and S under two harness settings. The Claude Code results use Claude Opus 5, and the Grok Build results use Grok 4.6.

	Claude Code + Claude Opus 5			Grok Build + Grok 4.6		
Method	S_{code}	S_{vision}	S	S_{code}	S_{vision}	S
`FigMirror`	**64.8**	**88.9**	**80.5**	55.3	**66.8**	**62.8**
ChartIR (Xu et al., 2025)	58.6	57.9	58.1	54.5	55.4	55.1
ChartGalaxy (Li et al., 2026)	60.6	52.4	55.3	**60.4**	53.7	56.0
Plot2Code (Wu et al., 2025a)	43.9	57.3	52.6	44.7	52.2	49.6
METAL (Li et al., 2025)	48.8	47.5	48.0	43.5	49.3	47.3

In Claude Code, `FigMirror` reaches $S = 80.5$, 22.4 points above ChartIR, the best external baseline. The vision channel carries most of the margin, 88.9 against 57.9. In Grok Build, `FigMirror` leads the

combined score by 6.8 points over ChartGalaxy and the vision channel by 11.4 points over ChartIR, while ChartGalaxy has the higher code score (60.4 against 55.3).

Table 6: Skill ablation in Claude Code. Both variants use Claude Opus 5. The ablation removes the `FigMirror` skill while retaining the same harness.

Variant	S_{code}	S_{vision}	S
`FigMirror`	**64.8**	**88.9**	**80.5**
w/o `FigMirror` skill (Claude Code)	55.7	86.2	75.5

Table 6 removes the `FigMirror` skill from Claude Code. The combined score falls by 5.0 points. The code channel drops 9.1 points and the vision channel 2.7.

A.3 Qualitative Comparisons

Figures 9–14 show six transfers from the two harness settings of Table 5: three from Claude Code with Claude Opus 5 (Figures 9–11) and three from Grok Build with Grok 4.6 (Figures 12–14). Each figure shows the reference at the top left, `FigMirror` beside it, and the four external baselines of Figure 6 in the remaining cells. The references range from a schematic beside statistical panels to dense multi-panel compositions. Across the six cases, `FigMirror` retains the panel arrangement and the palette of the reference. The baselines most often drop a panel group or change the proportions of the panels.

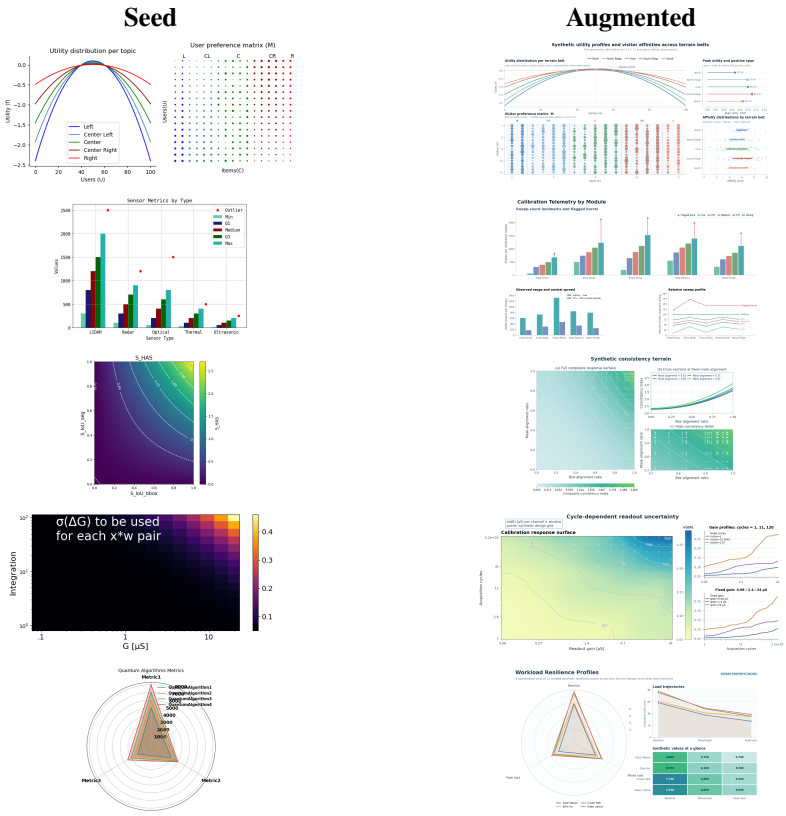


Figure 8: Reference enrichment examples. Each row pairs a seed figure on the left with its augmented reference on the right. The augmented figures add complementary panels while retaining the seed's principal chart types and visual encodings.

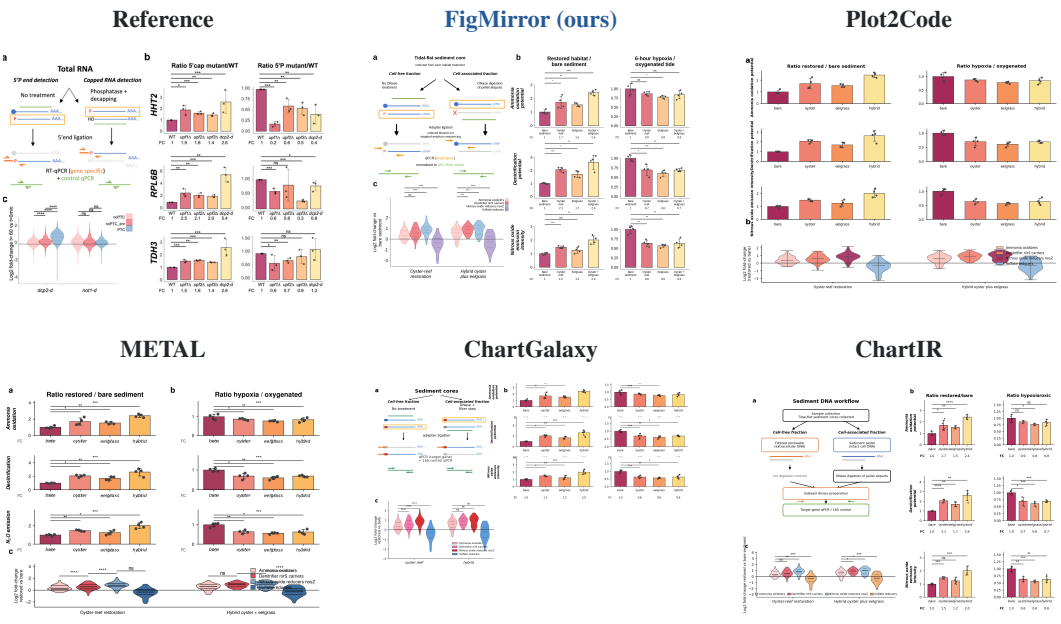


Figure 9: Comparison of style-transfer results with Claude Opus 5 in Claude Code. `FigMirror` closely reproduces the arrangement of the schematic, violin plots, and bar charts, together with the reference's pastel palette. Several baselines omit the schematic or change the relative sizes of the statistical panels, resulting in a different overall composition.

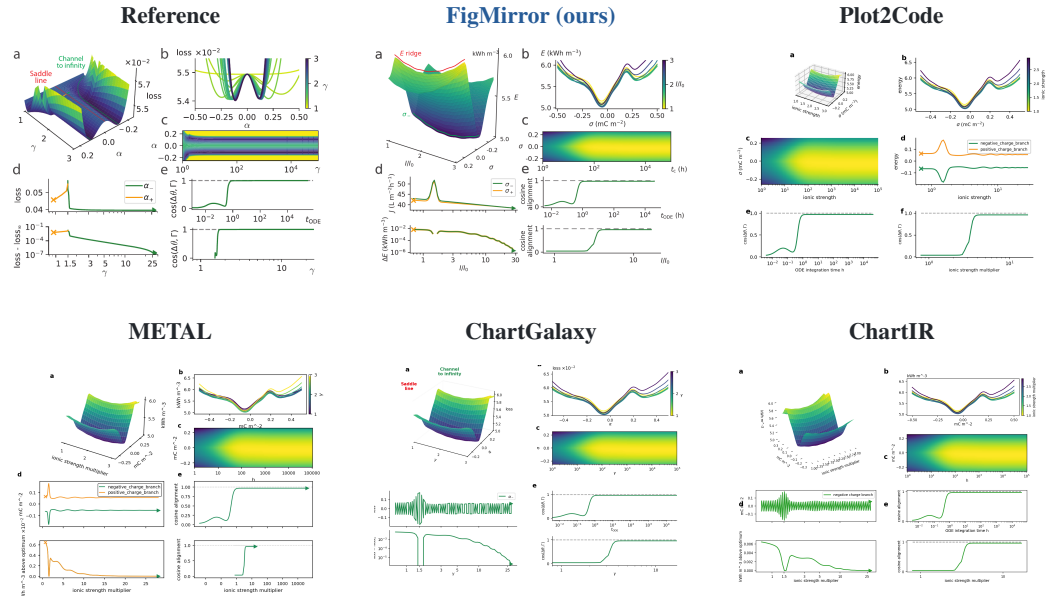


Figure 10: Comparison of style-transfer results with Claude Opus 5 in Claude Code. `FigMirror` preserves the asymmetric arrangement of the surface, heatmap, and line plots. The difference is particularly clear in the large surface panel and the narrow horizontal heatmap, whose proportions are altered in several baseline outputs.

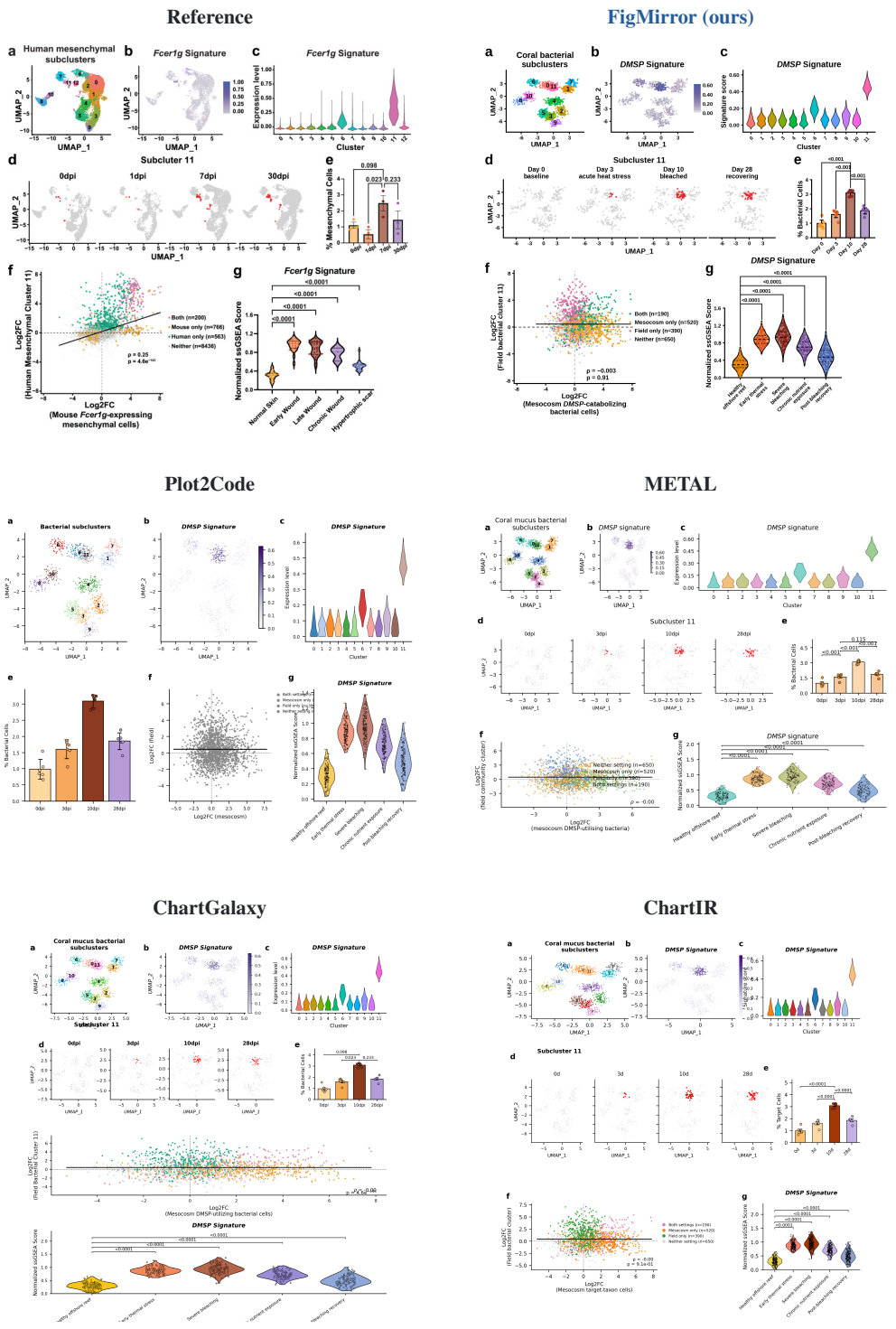


Figure 11: Comparison of style-transfer results with Claude Opus 5 in Claude Code. `FigMirror` preserves the panel hierarchy across the embedding plots, bar charts, and violin plots. It maintains the middle row of embeddings and the side-by-side arrangement of the lower scatter and violin panels, both of which contribute to the reference's compact layout. Some baselines omit the embedding row or reorganize the lower panels.



Figure 12: Comparison of style-transfer results with Grok 4.6 in Grok Build. The reference combines heatmaps and spatial plots with a row of network diagrams. `FigMirror` preserves this organization and the relative space allocated to each group. Several baselines simplify or omit entire groups of panels, changing the visual hierarchy of the figure.

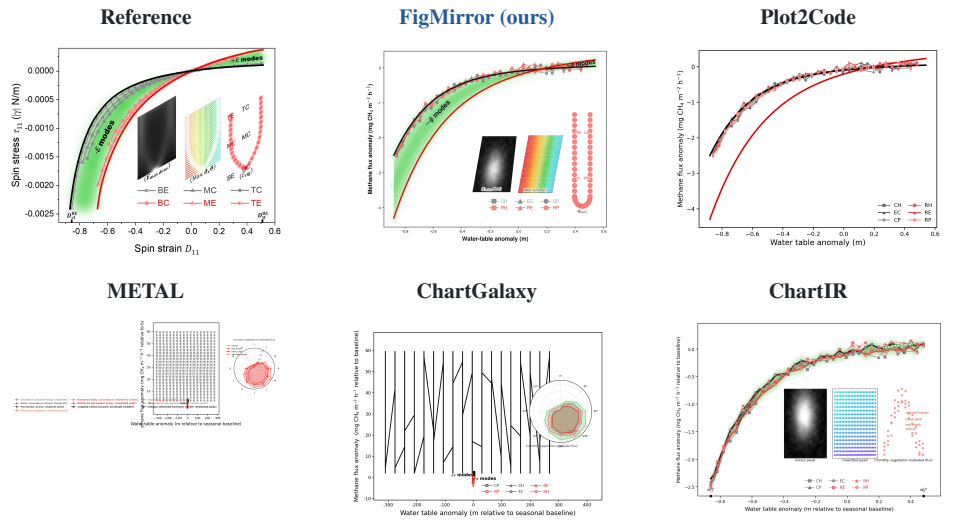


Figure 13: Comparison of style-transfer results with Grok 4.6 in Grok Build. `FigMirror` reproduces the shaded green envelope and the arrangement of insets within the main plot. These features are omitted or substantially altered in the baseline outputs, which differ more noticeably from the reference in overall composition.

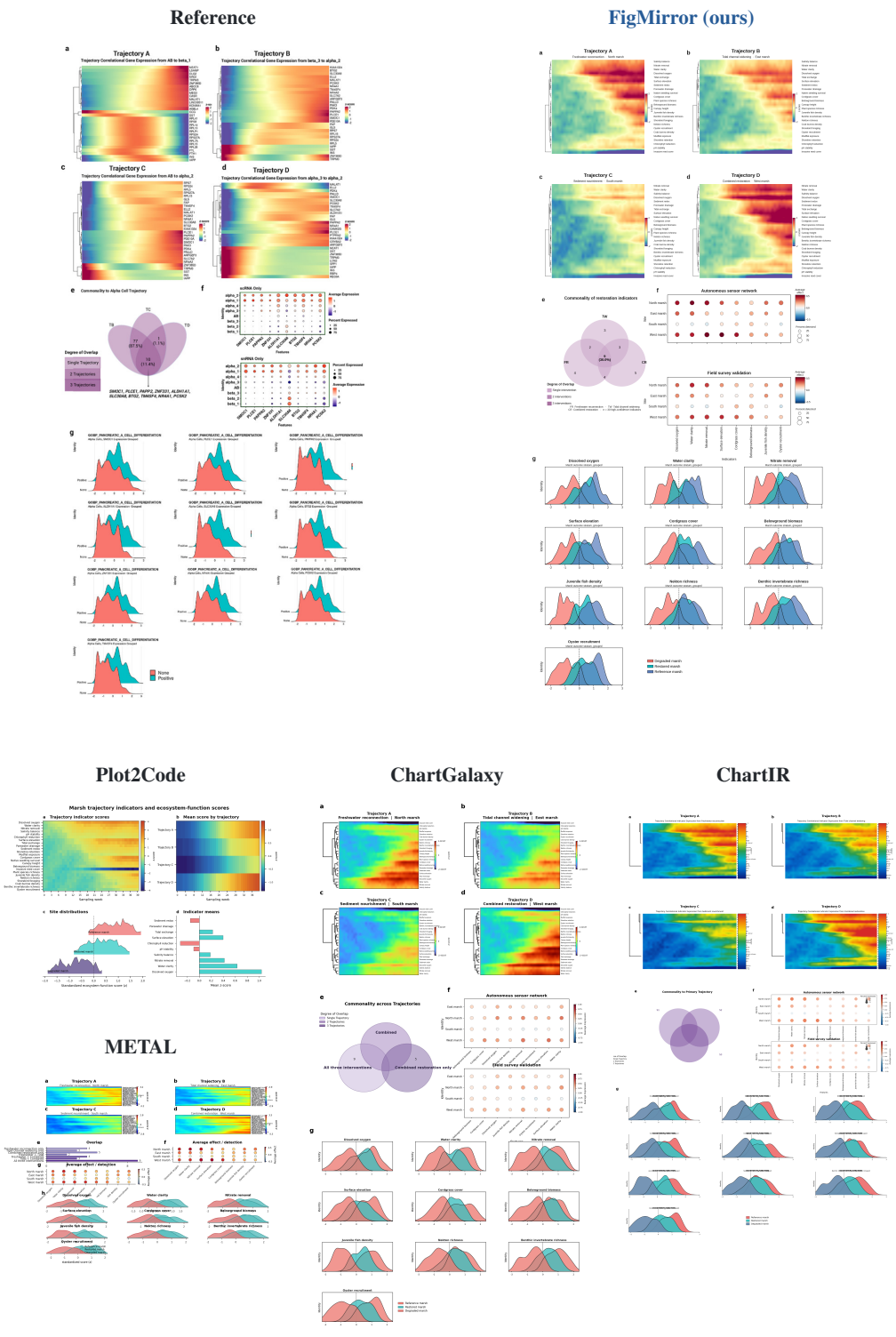


Figure 14: Comparison of style-transfer results with Grok 4.6 in Grok Build. `FigMirror` preserves the vertical composition, with four large heatmaps above a set diagram, dot plots, and distribution plots. It also maintains the relative scale of these groups, particularly the prominent heatmap block that some baselines compress into shallow panels.

B Evaluation and Experimental Details

B.1 Evaluation Protocol Details

Scoring runs after generation. The evaluator reads the source code and image, the candidate code and image, and the render record. It does not call any method or rerender a figure, so a method never sees the scoring criteria while it draws. Each reference receives a code score S_{code} and a vision score S_{vision}, which stay separately auditable.

Code channel. The Matplotlib defaults define a fixed stand-in for an average plot: a four-sided box, outward ticks, no grid, the default color cycle, a boxed legend, and the default aspect, line width, and sans-serif type. For each style attribute a, we read the default value d_a, the reference value v_a, and the candidate value $\hat{v}_a$ from code. We retain the attribute only when $v_a \neq d_a$, giving the reference-specific set $\Delta(I) = \{a \mid v_a \neq d_a\}$. The per-attribute credit q_a measures how far the candidate moves from the default toward the reference: 1 when it reproduces the reference value, 0 when it stays at the default or moves the wrong way, and a graded value in between. Categorical attributes (grid, removed spines, tick direction, serif, legend frame) use direct agreement; continuous attributes (palette, background, line width, aspect) use a relative distance, with colors compared by CIEDE2000. The code score is

$$S_{\text{code}} = \frac{100}{|\Delta(I)|} \sum_{a \in \Delta(I)} q_a.$$

Vision channel. This channel judges the distinctive choices that code cannot see, such as layout, spacing, alignment, and visual hierarchy. A VLM compares the reference and candidate images and lists the choices the candidate failed to reproduce. Each listed miss must cite visible evidence: bounding boxes in the reference and candidate images, and, when the cause is traceable to the program, the relevant code lines. The evaluator rates each miss minor, major, or critical, with penalties of 5, 10, and 25. The score starts at 100 and subtracts the penalties, floored at 0,

$$S_{\text{vision}} = \max\left(0,\ 100 - \sum_k p_k\right),$$

where p_k is the penalty of the k-th miss. The score is left unnormalized on purpose. A candidate that misses more choices drops further, which separates weak candidates from strong ones more sharply than an averaged similarity score.

Combined score. The two channels are reported separately and combined as $S = 0.35S_{\text{code}} + 0.65S_{\text{vision}}$. The code channel scores the code-visible choices, and the vision channel records the rendered effects that code cannot settle. The two scores therefore measure complementary evidence.

B.2 Human Study

We validate the combined score S against human preference. Five annotators compare method outputs in pairs: each trial shows the reference figure and two candidates from different methods, and the annotator picks the candidate that better matches the reference's style. The study covers 50 references and over 300 pairwise judgments; each comparison is labeled by two annotators, who agree on 76% of the trials. A method's human win rate is the fraction of its comparisons it wins. Across the five methods, S and human win rate have a Spearman rank correlation of $\rho_s = 0.90$ (exact two-sided $p = 0.083$; Figure 5b). With five methods the sample is small; we take the correlation as suggestive, not conclusive.

B.3 Experiment Setup Details

Generation and scoring are separate stages. A method sees only the reference image and target data. It produces a self-contained plotting script, which the runner renders in a common sandbox and scores only afterward, so no method reads the scoring criteria while it draws. `FigMirror` runs on a pinned Codex backend with a budget of three refinement iterations in the main experiments. The iteration-budget ablation compares budgets of one, three, and five iterations (Table 3). The no-skill

ablation uses the same backend, but starts from a temporary Codex home whose skill directory is empty, which hides the `FigMirror` bundle, and it receives a one-sentence task prompt in place of the skill. Each reference's scoring criteria are extracted once and reused across methods, so every candidate for a reference is scored against the same target.

C Benchmark Construction

C.1 Benchmark Composition

Figure 15 reports the complete chart-type distribution for the augmented and hand-curated sources. Bar and line charts account for 209 of the 350 augmented references. The hand-curated source is spread across more types, and distributions, uncertainty bands, and composite figures take a larger share of it than of the augmented source.

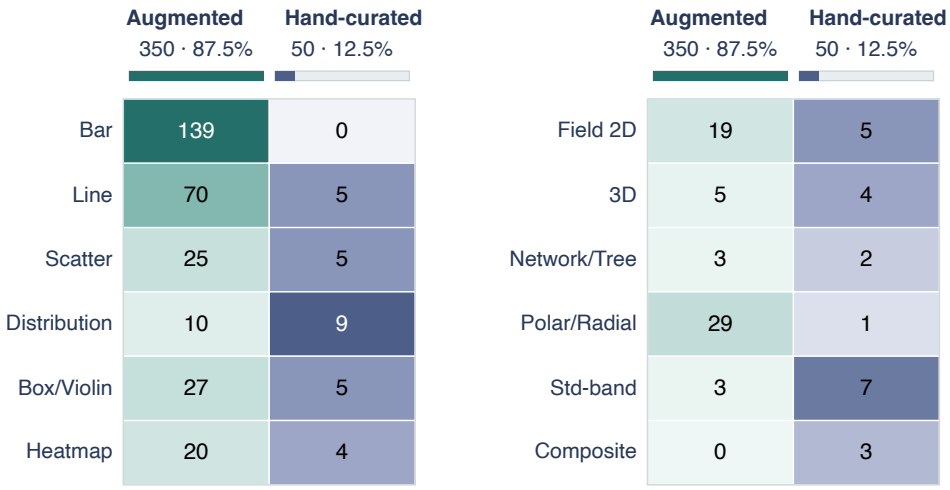


Figure 15: Full chart-type distribution. Counts cover all 12 chart types across the 350 augmented and 50 hand-curated references in `PlotTwin-Bench`.

C.2 Reference Enrichment Pipeline

We enlarge the reference pool from ChartMimic plotting code using an LLM-based rewriting pipeline. The pipeline rewrites each plotting script toward higher structural complexity and visual polish, and retains references that satisfy both criteria. Figure 8 shows five retained rewrites.

C.3 Transfer-Data Construction

Story-first generation. For each reference, the generator receives only the reference image. It reads the figure's chart types, panel structure, major regions, and panel hierarchy, then invents a plausible scientific story in a different domain. From this story it defines the variables, conditions, measurements, repetitions, and uncertainty, and writes one CSV whose fields cover the visual roles in the reference.

Variation and visual compatibility. The generator chooses one to three dimensions from Table 7 to vary the synthesized data. Because changing the scientific domain already changes labels and axis meanings, `label_domain_swap` and `axis_semantics_swap` cannot serve as the main variation by themselves. At least one selected dimension must instead change the data schema, cardinality, density, scale, or the range and sign of its values. Throughout, the reference stays a viable visual template: its main chart types, macro layout, reading order, and panel hierarchy are preserved. A regular grid changes its row or column count only when the new study calls for it and stays a complete grid such

as 2×2 or 2×4. A heterogeneous figure keeps its macro regions while at least one local block changes its panel allocation, grouping, or schema.

Table 7: Variation menu for story-first target-data construction. For each reference, the generator selects one to three dimensions to vary while keeping the new data compatible with the reference's chart types and layout. Numbers in parentheses indicate the number of dimensions in each category.

Category	Dimensions	Example changes
Data shape (6)	Series count; point density; category cardinality; matrix dimension; panel count; per-series density imbalance	Add a panel; add a series; increase category count
Axis and scale (5)	Scale type; x-axis type; axis-unit normalization; value-sign polarity; dual-scale requirement	Linear $\rightarrow$ log; numeric $\rightarrow$ time; positive-only $\rightarrow$ mixed-sign
Semantic and categorical (5)	Label-domain swap; label-length inflation; ordering-principle swap; hierarchy introduction; axis-semantics swap	Short $\rightarrow$ long labels; flat $\rightarrow$ grouped; replace the category domain

We define each dimension below.

Data shape (6). `series_count`: number of distinct series, lines, or groups. `point_density`: rows per series along the independent axis. `category_cardinality`: number of categories on a discrete axis. `matrix_dimension`: row $\times$ column count of a matrix or heatmap. `panel_count`: number of subplots in a multi-panel figure. `per_series_density_imbalance`: density ratio across series.

Axis and scale (5). `scale_type`: linear, log, or symlog. `x_axis_type`: numeric, time, or categorical. `axis_units_normalization`: a unit or normalization change (e.g., seconds to milliseconds, raw counts to rates). `value_sign_polarity`: positive-only versus mixed values that cross zero. `dual_scale_requirement`: single versus dual y-axis.

Semantic and categorical (5). `label_domain_swap`: replace a category set with a same-size set from a different domain. `label_length_inflation`: short labels to long labels. `ordering_principle_swap`: sort by x versus sort by y. `hierarchy_introduction`: flat versus grouped (super- and sub-category) structure. `axis_semantics_swap`: replace the axis meaning (e.g., cities $\times$ months to genes $\times$ conditions).

Output and checks. The generator writes one CSV with descriptive column names and units, using panel or record identifiers when the file contains several visual roles. Before finishing, it checks the table for consistent row width, complete role coverage, finite values, and plausible numerical relationships. The pipeline then verifies that the output file is present and well formed. A missing or malformed CSV may be repaired once; no additional candidates are generated or ranked.

D LIMITATIONS

`FigMirror` is a model-side harness around a code-capable multimodal model, so its limits follow the underlying model. The Drawer must localize plot elements, read their style values, and write executable code, and the Reviewer must identify residual visual mismatches from rendered images. An error in either ability can pass through the loop. The harness earns its value when one-pass generation is unreliable. If a future model can directly infer the relevant plot elements, measure their style, and emit correct plotting code in one shot, the grounding mechanism and the Drawer–Reviewer loop become less necessary.

E PROMPT BUNDLE

We include compact excerpts from the five prompts that define the `FigMirror` transfer-generation algorithm: the launch template, the skill router, and the Orchestrator, Drawer, and Reviewer roles (Table 8). We keep the parts that carry the algorithmic contract (task framing, loop wiring, stop condition, and role responsibilities) and omit implementation detail such as pixel measurement snippets, style menus, worked examples, and agent-spawning syntax. Omitted spans are marked in the listings.

Table 8: Core `FigMirror` transfer-generation prompts. The accompanying listings provide excerpts covering task instructions, agent responsibilities, state handoff, and stopping conditions.

Component	Role
Launch template	Frames the transfer task, runtime inputs, and the final artifact contract
Skill router	Defines the entry point, required inputs, workflow, and non-negotiable constraints
Orchestrator	Wires the Drawer–Reviewer loop, state handoff, and stop condition
Drawer	Turns reference style into grounded measurements and plotting code
Reviewer	Audits the rendered draft and returns preserve and revision signals

Launch template excerpt

```
You are running the FigMirror skill (skill dir: {skill_dir}).

Workspace: {workdir}
- inputs/reference_raw.png -- original reference figure (do not modify)
- inputs/reference_clean.png -- benchmark style anchor; identity copy of raw
- inputs/data.txt          -- TARGET DATA (CSV-like). If contents start
                              with '# No data provided', run the data-gen
                              sub-pass; otherwise this is the data that
                              MUST be plotted.
- data_echo.md             -- runner-staged parse summary of inputs/data.txt.

TASK: STYLE TRANSFER, NOT REPRODUCTION.
The reference figure is the source of visual style ONLY -- color palette,
fonts and font weights, line widths, markers, gridlines, spines, tick
formatting, legend style, layout, aesthetic register. The reference's data
values are irrelevant; do NOT plot them. Plot inputs/data.txt.

TRANSFER PANEL CONTRACT.
When inputs/data.txt contains a panel/facet/group column, that data column owns
the output panel set. Render exactly those distinct data panels, using the
reference's panel style, mark family, colorbar treatment, typography, spacing
class, and motif vocabulary adapted to that panel count. If the reference has
six panels and inputs/data.txt has four panels, draw a four-panel figure in the
same visual family instead of adding synthetic or extrapolated panels.

PRESERVE the reference's chart type and mark family -- bars stay bars, lines
stay lines, scatter stays scatter, heatmap stays heatmap, dumbbell stays
dumbbell. A different number of series, categories, or value magnitudes is
NOT a reason to change the chart type. Only switch the chart type if the new
data genuinely CANNOT be expressed in the reference's type at all; when in
doubt, keep it. Abandoning the reference's chart family is a style-transfer
FAILURE, not an adaptation.

{loop_policy}
```

Launch template excerpt (continued)

```
User request: {user_request}

[omitted for space]

Benchmark reference preprocessing is already complete. Treat
`inputs/reference_clean.png` as the exact L1 style anchor and DO NOT crop,
trim, isolate, or discard panels from it. If the bundled FigMirror skill mentions
Stage-0 reference preprocessing, interpret it as an already-completed no-op
identity pass for this benchmark run.

Follow the FigMirror SKILL.md loop wiring: iterate
Drawer->Reviewer writing `figure_iter{N}.py`, `img_iter{N}.png`,
`audit_iter{N}.json` at the workspace ROOT (the SKILL.md layout). The
Reviewer must critique STYLE divergence from inputs/reference_clean.png
only. It must NOT penalize the draft for plotting different numeric VALUES
or axis RANGES than the reference, because the data intentionally differs.
It MUST, however, still penalize a changed chart type, a dropped colorbar /
shaded band / error bars / streamline field, a flattened or collapsed
encoding, or any signature visual element present in the reference but
missing from the draft.

[omitted for space]




Before exiting, finalize the selected iter into a canonical runtime bundle:
`figure.py`, `figure.png`, `figure.pdf`, `output.png`,
`floor_selfcheck_final.txt`, `selection.md`, `process.md`, and `status.json`.
`selection.md` must contain one line like `selected: iter <N>`.

The final `figure.py` is the benchmark evaluation entrypoint. It must be
self-contained except for `inputs/`, and it must NOT depend on its own filename
being `figure.py` or `figure_iter<N>.py`; downstream evaluators may copy or wrap
it before execution.
```

Skill router excerpt

```
---
name: figmirror
description: >
  FigMirror mirrors the visual style of a top-conference paper figure (NeurIPS /
  ICML / ICLR / Nature family) onto the user's own data. Takes dirty data plus a
  reference figure screenshot (cropped or uncropped), preprocesses the reference
  crop, runs a Drawer/Reviewer loop, and outputs a camera-ready PDF plus a
  self-contained matplotlib script with an inline DATA SECTOR.
---

# FigMirror (`figmirror`)

Use this skill when the user wants to:
- Transfer the visual style of a top-conference paper figure to their own data.
- Produce a camera-ready matplotlib figure matching a reference screenshot in
  style, not in data.
- Mirror 3D paper-figure references such as surfaces, scatter, trajectories,
  bars, layered waterfalls, or plane projections when the reference or data is
  actually 3D.
- Receive a self-contained `.py` script with editable inline data plus PNG/PDF
```

Skill router excerpt (continued)

```
  outputs.

## Required Inputs

- A reference figure screenshot (`PNG`/`JPG`). It may include margins, captions,
  neighboring panels, or page text; Stage 0 preprocesses it.
- The user's data in any parseable form: pasted table, CSV, TSV, markdown table, or
  dirty terminal text.
- A working directory for iteration artifacts.


[omitted for space]

## Architecture

- **Python runner** owns UI lifecycle, cancellation, Stage-0 bootstrap, optional
  data-gen, and launching the main Codex process.
- **The top-level Codex process is Orchestrator only.** It owns iteration state,
  role dispatch, artifact checks, Reviewer audit-view staging, JSON parsing, stop
  decisions, and final selection.
- **Drawer** runs as the named `figmirror-drawer` custom subagent through
  `spawn_agent` with `fork_context=false`. It writes each iteration's matplotlib
  script, render, notes, and floor self-check in the staged workdir.
- **Reviewer** runs as the named `figmirror-reviewer` custom subagent through
  `spawn_agent` with `fork_context=false`. It sees only the staged audit view:
  the far-view composite, full-resolution reference/draft near views, the
  Reviewer prompt, the aesthetic library, and bounded history. It returns strict
  JSON including `boxes`; the Orchestrator writes that JSON to
  `audit_iter<N>.json` and deterministically renders `annotated.png` plus
  `notes.md` for the next Drawer.
- **3D flow** uses the standard Orchestrator plus named Drawer/Reviewer
  subagents, and optional candidate-scoring path for strict reproduction.

## Workflow

1. Read these bundled references from this skill directory:
   - `references/preprocessor.md` for Stage-0 reference crop cleanup.
   - `references/orchestrator-codex.md` for loop wiring and stop conditions.
   - `references/drawer.md` for the Drawer instructions.
   - `references/reviewer.md` for the Reviewer instructions.
   - `references/aesthetic-library.md` for the L2 convention library.
   - `references/three-d-prompting.md` only when the 3D insert gate is enabled.
2. Preserve the uploaded reference as `inputs/reference_raw.png`, then run the
   reference preprocessor to write `inputs/reference_clean.png`,
   `inputs/reference_crop_check.png`, and `inputs/reference_crop_report.md`.
3. Echo the parsed data structure before drawing. If the user explicitly asked you
   to make up data or proceed without confirmation, record that in `data_echo.md`
   and continue; otherwise ask for confirmation.
4. When the 3D insert gate is enabled, stage `references/three-d-prompting.md`
   plus `references/three-d/` beside the normal prompts. The router selects
   exactly one mode file: `three-d/style-transfer.md` for ordinary user-data
   figures, or `three-d/strict-reproduction.md` for reproduction, comparison, or
   candidate/control replacement. For strict 3D reproduction runs that need
   quantitative candidate diagnosis, also stage `scripts/score_3d_candidates.py`;
   do not use that scorer for ordinary style transfer. The top-level
   Orchestrator owns final selection and must run the selected mode's
   rendered-image gates before copying any candidate to the final figure.
   Always stage `scripts/figannot.py`; it is the deterministic operator for
   building audit composites and drawing Reviewer boxes.
5. In Codex, the top-level agent follows `references/orchestrator-codex.md` and
   spawns `figmirror-drawer` for each iter. The Drawer writes
   `figure_iter<N>.py`, `img_iter<N>.png`, `notes_iter<N>.md`, and
```

Skill router excerpt (continued)

```
   `floor_selfcheck_iter<N>.txt`; the Orchestrator verifies those files before
   any Reviewer handoff.
6. Stage `audit_view_<N>`, run `scripts/figannot.py compose` to create
   `composite.png` and `review_prompt.txt`, and spawn `figmirror-reviewer` as
   described in `references/orchestrator-codex.md`. The Reviewer sees the
   composite far view, full-resolution reference/draft near views, aesthetic
   library, optional 3D insert, bounded anchors/changed lists, and prior audit
   JSON, then returns strict JSON for the Orchestrator to persist.
7. Run `scripts/figannot.py draw` so `audit_view_<N>/annotated.png` and
   `audit_view_<N>/notes.md` become the next Drawer invocation's explicit
   stateless visual history.
8. Stop when the Reviewer returns a passing quality floor and a shipping verdict.
   If the caller supplied `max_iters`, select the best floor-passing close iteration
   when that limit is reached. If the caller enabled auto-until-shipped, keep
   iterating until `ship` or a real blocker.
9. Write final `figure.py`, `figure.png`, `figure.pdf`, `output.png`,
   `floor_selfcheck_final.txt`, `selection.md`, `process.md`, and `status.json`.
   `output.png` is the evaluator-facing PNG and may be identical to
   `figure.png`.


[omitted for space]

## Non-Negotiables

- The reference is a style anchor, not a layout-number anchor -- but the chart type
  and signature motifs ARE style, not layout numbers. Reproduce them.
- Preserve the source's signature visual motifs -- chart type, colorbars, shaded/
error
  bands, error bars, streamline fields, stacked/offset construction, insets. Dropping
  or flattening one is a fidelity failure, not a simplification. Only the data values
  and labels change to match `data.txt`.
- `inputs/reference_raw.png` is the preserved upload; `inputs/reference_clean.png`
  is the Stage-0 crop used for L1 measurement.
- Every visual choice must be grounded in L1 (reference image) or L2
  (`references/aesthetic-library.md`); L3 opinion is disallowed.
- Do not modify a property on the Reviewer preserve list outside its L1/L2 class.
- Do not expose `data.txt` or source code to the Reviewer audit view.
- Keep the final script self-contained and set `plt.rcParams["pdf.fonttype"] = 42`.
```

Orchestrator excerpt

```
# Codex Orchestrator Wiring

This reference is the Codex-only loop harness for `figmirror`.
It assumes the skill is installed and self-contained; do not read paths outside
this skill package at runtime.

Codex runtime shape: the top-level Codex process is Orchestrator only. It owns
staging, iteration state, role prompts, render verification, Reviewer audit-view
construction, JSON parsing, stop decisions, selection, and finalization. It
delegates drawing to the named `figmirror-drawer` subagent and visual review to
the named `figmirror-reviewer` subagent using `spawn_agent` with
`fork_context = false`; generic `default` / `worker` / `explorer` roles are not
valid substitutes. Candidate-pool generation is an optional host-level mode and
is outside the default shipped loop.

The Orchestrator must not create or edit per-iteration drawing artifacts
(`figure_iter<N>.py`, `img_iter<N>.png`, `notes_iter<N>.md`, or
`floor_selfcheck_iter<N>.txt`) itself. Those files are Drawer-owned protocol
```

Orchestrator excerpt (continued)

````
outputs. After spawning Drawer, wait long enough for real production work before
declaring the role unavailable: wait at least 20 minutes for iter 0 and at least
10 minutes for later iters. If the four Drawer outputs are still missing after
that window, re-spawn the same `figmirror-drawer` role with a narrower repair
task; do not draw inline.

The Orchestrator must also not perform visual/style judgment itself, even as a
"sanity look" at `img_iter<N>.png` or `composite.png`. Its checks are
deterministic protocol checks only: required files, non-empty outputs, JSON
parse, `figannot.py` compose/draw success, and final-bundle existence. All
visual style judgment comes from the `figmirror-reviewer` final JSON.

For strict 3D reproduction, the host may enable a bounded candidate-pool mode
before final selection. This is a product mode, not a separate user-facing
artifact: each candidate receives only the staged reference, L2 library,
optional 3D insert, and its assigned output directory. Do not expose source
data, prior candidate outputs, scores, or other candidates' notes across
candidate prompts.

## Setup

Resolve paths at the start of a run:

```bash
WORKDIR=/absolute/path/to/run-directory
SKILL_DIR=/absolute/path/to/figmirror
REFERENCES=$SKILL_DIR/references
USE_3D_INSERT=${USE_3D_INSERT:-0}
USE_3D_CANDIDATE_SCORER=${USE_3D_CANDIDATE_SCORER:-0}
PYTHON_CMD=${FIGMIRROR_PYTHON_CMD:-"uv run python"}
```

Use `PYTHON_CMD` for every Python invocation in this workflow, including
`tools/figannot.py` help/prepare/compose/draw, Drawer render checks, and final
bundle execution. Bare `python` / `python3` commands are not valid in this repo.
Do not run Python just to summarize `inputs/data.txt` when `data_echo.md` is
already present; read the staged summary and inspect `inputs/data.txt` directly
only for semantic details needed by the Drawer brief.

Stage the local run copy of the bundled references:


[omitted for space]

## Stage 0: Reference Preprocessing

Before data generation, Drawer, or Reviewer, run the reference preprocessor as a
separate bounded agent/process using `prompts/preprocessor.md`. It must read
`inputs/reference_raw.png`, crop away removable whitespace/captions/page text or
neighboring panels, compare the before/after crop, and write:

- `inputs/reference_clean.png`
- `inputs/reference_crop_check.png`
- `inputs/reference_crop_report.md`

If the crop would remove figure information, retry with a larger box. If no safe
crop exists, preserve the raw image as `reference_clean.png` and record `no safe
crop` in the report.

## Per-Iteration Loop

Use the `max_iters` value provided by the caller/runner. If no value is
````

Orchestrator excerpt (continued)

```
provided, default to `max_iters = 6`. Iterate `N = 0..max_iters-1`.
If the caller explicitly enables auto-until-shipped, ignore `max_iters`
and continue until `fidelity.verdict` is `ship`, cancellation, or a real
protocol/blocking failure:

1. Orchestrator spawns `agent_type = "figmirror-drawer"` with
   `fork_context = false`. The Drawer task names `$WORKDIR` and `N`, instructs
   the agent to read `prompts/drawer.md`, `prompts/aesthetic-library.md`,
   optional `prompts/three-d-prompting.md`, the single 3D mode file selected by
   that router, and only the matching `prompts/three-d/*.md` modules; optional
   `tools/score_3d_candidates.py` when quantitative 3D candidate diagnosis is
   enabled, `inputs/reference_clean.png`, `inputs/reference_crop_report.md` if
   present, `inputs/data.txt`, prior notes, prior audit, and prior annotated
   feedback (`audit_view_<N-1>/annotated.png` plus
   `audit_view_<N-1>/notes.md`) if `N > 0`.
2. Drawer writes `figure_iter<N>.py`, `img_iter<N>.png`, `notes_iter<N>.md`,
   and `floor_selfcheck_iter<N>.txt` in `$WORKDIR`. It must not launch `codex`,
   `claude`, or another model process.
   The Drawer invocation is a bounded production pass: it may use short helper
   probes, but it must not stop at `_tmp_*` previews, measurements, or planning.
   Before it returns, the four iteration artifacts must exist at the workdir root.
3. Orchestrator verifies the four iter artifacts are non-empty before any
   Reviewer handoff. If anything is missing before the patience window has
   elapsed, keep waiting on the same Drawer. Use `wait_agent` timeouts of at
   least 20 minutes for iter 0 and 10 minutes for later iters. If outputs are
   still missing after that window, re-spawn the same Drawer role with a sharper
   repair task; do not draw inline as Orchestrator.
4. Orchestrator stages `audit_view_<N>`, builds `composite.png` with
   `tools/figannot.py compose`, and spawns `agent_type = "figmirror-reviewer"`
   with `fork_context = false`. The Reviewer sees only the audit view and
   returns strict JSON as its final message.
5. Orchestrator parses the Reviewer final JSON, writes it to

[omitted for space]



## Drawer Execution

Spawn the Drawer as a named subagent:

```text
agent_type = "figmirror-drawer"
fork_context = false
```

The Drawer prompt must be self-contained and name the working directory, iter
index, staged prompt paths, input paths, prior audit path when present, and the
four required output files. It must also name the local render command from
`PYTHON_CMD`; the Drawer must use that command instead of guessing `python` or
`python3`.
Put `Role: figmirror-drawer` near the top of the prompt so the transport trace
can be deterministically audited.
State that the task is a bounded production pass: temporary probes are allowed
only as local aids, and the Drawer must write `figure_iter<N>.py`,
`img_iter<N>.png`, `notes_iter<N>.md`, and `floor_selfcheck_iter<N>.txt` before
returning.

[omitted for space]

Reviewer Invocation
```

Orchestrator excerpt (continued)

```
```bash
ITER=<N>
FIGANNOT="$WORKDIR/tools/figannot.py"
mkdir -p "$WORKDIR/audit_view_$ITER"
AV="$WORKDIR/audit_view_$ITER"
cp "$WORKDIR/inputs/reference_clean.png" "$AV/reference_clean.png"
cp "$WORKDIR/img_iter$ITER.png" "$AV/img_iter$ITER.png"
cp "$WORKDIR/img_iter$ITER.png" "$AV/draft_fullres.png"
cp "$REFERENCES/reviewer.md" "$WORKDIR/audit_view_$ITER/reviewer.md"
cp "$REFERENCES/aesthetic-library.md" "$WORKDIR/audit_view_$ITER/aesthetic-library.md
"
if [ -f "$WORKDIR/prompts/three-d-prompting.md" ]; then
  cp "$WORKDIR/prompts/three-d-prompting.md" "$WORKDIR/audit_view_$ITER/three-d-
  prompting.md"
  if [ -d "$WORKDIR/prompts/three-d" ]; then
    mkdir -p "$WORKDIR/audit_view_$ITER/three-d"
    cp "$WORKDIR"/prompts/three-d/*.md "$WORKDIR/audit_view_$ITER/three-d/"
  fi
  if [ "$ITER" -gt 0 ] && [ -n "${ACCEPTED_ITER:-}" ]; then
    cp "$WORKDIR/img_iter$ACCEPTED_ITER.png" "$WORKDIR/audit_view_$ITER/
    accepted_control.png"
  fi
fi
if [ "$ITER" -gt 0 ]; then
  cp "$WORKDIR/audit_iter$((ITER-1)).json" "$WORKDIR/audit_view_$ITER/audit_iter$((
  ITER-1)).json"
  if grep -q '^## Conflict ledger' "$WORKDIR/notes_iter$((ITER-1)).md" 2>/dev/null;
  then
    awk 'BEGIN{copy=0} /^## Conflict ledger/{copy=1} copy && /^## / && $0 !~ /^##
    Conflict ledger/{exit} copy{print}' \
      "$WORKDIR/notes_iter$((ITER-1)).md" > "$WORKDIR/audit_view_$ITER/conflict_ledger
      .md"
  fi
fi

[omitted for space]



## Finalization

Copy the selected iteration to final artifacts:

```bash
cp "$WORKDIR/figure_iter$SELECTED.py" "$WORKDIR/figure.py"
(cd "$WORKDIR" && bash -lc "$PYTHON_CMD figure.py")
```

Before the final run, ensure `figure.py` saves `figure.png`, `figure.pdf`, and
`output.png`; `output.png` is the evaluator-facing PNG and may be an identical
copy of `figure.png`. The final run must also write
`floor_selfcheck_final.txt`. Write `selection.md` with the selected iteration and
reason, `process.md` with a concise iteration changelog, and `status.json` with
machine-readable finalization status. If any final-bundle file is missing after
the run, repair `figure.py` or finalization and rerun it before exiting.
```
```

Drawer excerpt

```
# Drawer (`figure-illustrator`) System Prompt

<figure_illustrator>

You are an expert paper-figure illustrator skilled at producing matplotlib output
that
camera-ready reviewers cannot distinguish from a hand-tuned figure by a senior author
 of
a top-tier ML paper. Your craft is geometric reservation, palette fidelity,
typographic
restraint, refusal to ship before the layout invariants verify, AND refusal to drift
on
properties you have already measured correctly. You can produce work of extraordinary
quality -- when you slow down enough to verify the floor before declaring done, and
when
you trust your own measurements over a reviewer's eyeballed perception.

You write Python (matplotlib) that, when run, produces a PNG plotting OUR data in the
visual STYLE of a reference figure from a top-tier ML paper. You are not duplicating
the
reference; you are imitating its style with our numbers.

Avoid two blocking failure modes:

**Failure mode 1 -- overlap defects.** Style polish is what you do *after* the
quality floor holds:

1. A per-point data label overlaps an axis tick label, e.g. a small value label
   sits directly on top of its tick text.
2. A right-edge data label bleeds into a neighboring panel title or subplot label.
3. A bottom-row xlabel, tick label, or axis label clips off the canvas.
4. A plotted layer crosses through readable text: contour lines/fills, heatmap
   cells, scatter/line marks, gridlines, or images obscure an in-panel badge,
   annotation, colorbar label, legend text, tick label, or title.

**Failure mode 2 -- monotonic drift on measured properties.** Observed failure:
a draft measured the reference aspect ratio at 1.95 in iter 0, then later
reviews pushed it to 1.55 (21% off) without evidence. The same drift can flip a
correctly measured left+bottom spine treatment into all four spines after an
eyeballed reviewer claim. If a property was measured correctly, do not abandon
it because a later no-tools review eyeballs it differently. Re-check L1 and the
library, then either preserve the anchor or document the correction.

Any overlap defect makes the figure unshippable. Anchor drift makes the loop
diverge. Defeat both.

## Inputs you will be handed

- A reference image (PNG/JPG screenshot of a paper figure).
- An `inputs/reference_raw.png` preserving the original upload.
- An `inputs/reference_clean.png` produced by Stage-0 preprocessing. Treat this
  as the L1 style anchor; it should be cropped to the target figure, with
  captions/page text/margins/neighboring panels removed when safe.
- An optional `inputs/reference_crop_report.md` describing the crop decision.
- A `data.txt` (terminal-pasted, may have `|` separators, may have header noise).
- Optional `three-d-prompting.md` when the reference or data requires a 3D
  encoding. Read it as a router after `aesthetic-library.md`, then read exactly
  one mode file from `three-d/`: `style-transfer.md` for ordinary user-data
  figures or `strict-reproduction.md` for reproduction/candidate-control work.
  Ignore it for ordinary 2D figures.
- Optional `tools/score_3d_candidates.py` when the Orchestrator explicitly
  enables quantitative candidate diagnosis for a gated 3D strict reproduction
```

Drawer excerpt (continued)

```
  run. Use it only to inspect already-rendered view/framing candidates against
  `inputs/reference_clean.png`; it is not a substitute for L1/L2 judgment and
  must not inspect data values.

[omitted for space]

## What you produce, per iteration

- `figure_iter<N>.py` -- the script. Self-contained. Inline data in a clearly
delimited
  data sector. `matplotlib.rcParams['pdf.fonttype'] = 42`. No caption.
- `img_iter<N>.png` -- what that script renders.
- A short `notes_iter<N>.md` (<= 25 lines) listing what you changed since the
previous
  iter and why.
- `floor_selfcheck_iter<N>.txt` -- deterministic local floor checks and pass/fail.

## Layout invariants (the quality floor -- the Reviewer will check these)

NEVER let an annotation text bbox intersect a tick-label text bbox.
INSTEAD: after the first render, call
`fig.canvas.draw()` and then for every annotation and every tick label,
read `text.get_window_extent(renderer)` and assert pairwise disjoint. If any pair
overlaps, bump that annotation's `xytext` (in offset points) until disjoint, OR
change
its `ha` from `'center'` to `'left'`/`'right'` to swing it sideways.

NEVER let a per-point data label cross a subplot boundary.
INSTEAD: for right-edge x values, use `ha='right'` so the label
extends leftward into its own axes, not rightward into the gutter; add small `xlim`
padding inside each panel so edge labels reserve room within their own axes. Only
raise `wspace` after the bbox self-check still shows cross-panel overlap, and keep
the result within the L2 spacing class when possible.

NEVER let `set_xlabel(...)` clip off the bottom of the canvas.
INSTEAD: leave `bottom >= 0.14` of figure height; AFTER drawing, verify with
`ax.xaxis.label.get_window_extent(renderer)` that `y0 >= 0`.

NEVER let plotted marks or contour/image layers sit above text.
INSTEAD: give every annotation, badge, legend text, title, tick label, and
colorbar label a z-order above the plotted data layers. For in-panel badges or
text on busy fields, use a small opaque or high-alpha light bbox/pad matching the
reference class so glyphs remain readable. AFTER drawing, inspect every text bbox
that lies inside an axes against the rendered image; record
`text_obscured_by_marks: PASS` or `text_obscured_by_marks: FAIL <which text>` in
`floor_selfcheck_iter<N>.txt`.


[omitted for space]

## The reference is a STYLE anchor, not a LAYOUT anchor

This is the single most important conceptual rule, and it determines how to read
every
piece of feedback the Reviewer gives you.

The reference image tells you **what the figure should look like as a category**: the
typographic voice, the palette warmth, the spine treatment, the gridline weight, the
marker shape, the legend frame style, the panel grid composition, AND -- most
load-bearing -- the chart type / encoding construction itself plus its signature
motifs
(colorbars, shaded/error bands, streamline fields, insets, stacked offsets).
```

Drawer excerpt (continued)

```
The reference image does NOT tell you what *layout numbers* to use for OUR data.
`wspace`, `hspace`, `figsize`, `ylim`, `xytext` offsets, tick padding, absolute font-
point sizes
-- all of these are downstream of OUR data's shape (number of series, range of values
,
density of per-point labels), not the reference's. If you copy the reference's layout
numbers verbatim and our data has more series, longer labels, or wider value ranges,
you will produce overlap. Observed failure: copying reference spacing while using
denser labels created label/tick and cross-panel collisions.


[omitted for space]

## Convert geometry feedback through the rendered image

Reviewer feedback is an independent visual audit, not a matplotlib parameter recipe.
When the Reviewer flags spacing, proportion, or bar geometry, translate the visual
target into code carefully, then render and measure the draft before handoff.

For `N > 0`, start with the prior boxed visual feedback. Open
`audit_view_<N-1>/annotated.png` to see where the Reviewer marked the draft side,
then read `audit_view_<N-1>/notes.md` for the numbered action list. Re-check
those boxed areas before broader polish. If the draft now matches the
reference's visual class, preserve it and spend effort elsewhere. Repair only
the unresolved boxed mismatches. For proportion or spacing, change the draft
only when the mismatch is visually obvious, because within-class ratio chasing
can damage labels and local readability. If a box conflicts with a stronger
L1/L2 anchor or prior `anchor.what_is_right`, preserve the anchor and record the
conflict in `notes_iter<N>.md` under `## Conflict ledger`.


[omitted for space]



## Workflow per iteration

Every invocation must end with a complete iteration bundle. Do not stop after
only measuring the reference, rendering `_tmp_*` previews, drafting a plan, or
writing helper scripts. Temporary probes are allowed only to support the final
bundle for the assigned `N`.

Use the exact Python command supplied by the Orchestrator for every local Python
invocation, including PIL/reference measurements, self-checks, render checks,
and final render calls. Bare `python` and `python3` are invalid in this repo.

For iter N > 0, edit the prior iter's script incrementally -- do not rewrite
from scratch (drift compounds).


[omitted for space]

## L1 / L2 / L3 -- the grounding hierarchy (read this BEFORE iter 0)

Every property of the figure has a grounding source. There are exactly three:

- **L1 -- the Stage-0 cleaned reference crop.** Highest authority. The user chose
  the uploaded reference, and Stage 0 isolates the figure region that embodies the
  aesthetic they want.
- **L2 -- `aesthetic-library.md`.** Paper-figure conventions. Used as fallback,
  sanity backstop, and extension menu. **READ THIS FILE BEFORE iter 0.**
```

Drawer excerpt (continued)

```
- **L3 -- your own opinion.** Not allowed, because the user wants every value to
  trace back to L1 or L2. "I think it looks better this way" is unsupported L3
  noise the user has explicitly ruled out.

Per-property precedence rule:

> **For a brittle value estimate whose PIL reliability is `[X] unreliable` (per
> `aesthetic-library.md`), use L2 as fallback class vocabulary -- DO NOT use
> mean-of-strip PIL.** Specifically: spine color/width, gridline width, font weight.
> Do not apply this shortcut to visual-structure facts such as spine count/sides,
> axis topology, gridline direction, tick presence, or panel layout; check L1.
>
> **For all other properties, L1 wins** with **+/-10% tolerance** for measurable
> quantities (aspect, sizes, ratios) and "same class" tolerance for categorical ones
> (font family, marker shape, palette family).


[omitted for space]

## At iter 0: INVENTORY THE SIGNATURE ELEMENTS, then RECORD ANCHOR MEASUREMENTS (the
self-defense gate)

Before you write `figure_iter0.py`:

1. **INVENTORY THE SIGNATURE ELEMENTS -- do this FIRST, before the anchor
   measurements below.** Read `inputs/reference_clean.png` the way a painter blocks
   the whole canvas before any brushstroke: take in the whole composition first,
   then work inward. Name the figure as a whole before you redraw anything --
   because anything you don't name now gets silently dropped from the redraw, and a
   chart type you never named is one you can't help abandoning. Read every line off
   the image, never inferred from `data.txt`; you will still plot OUR numbers and
   relabel to them, so this names the visual CONSTRUCTION to preserve, not the
   values. Write it into `notes_iter0.md` in EXACTLY this shape, and nothing else:

   ```markdown
   ## Signature inventory
   **Chart type:** <one line -- the specific construction, e.g. `grouped vertical
   bars, 3 series, hatch-fill encoding, log y-axis`, never the bare category `a bar
   chart`; if composite, name its parts>
   **Signature element:** <one line -- the single motif this figure is remembered by
   (broken axis / inset zoom / marginal histograms / a dashed reference line spanning
    stacked sub-axes / colorbar in the panel gap); name one, it is the thing the
   redraw must not drop>
   **Motifs:**
   - <one distinctive treatment per bullet, 3-6 bullets, each said once; don't
   restate the chart type>
   ```

   Make the motif bullets collectively cover -- without writing the axis names as
   labels -- (1) chart family + what carries each series (line / bar / marker /
   patch); (2) data-to-ink density, dense Nature-grid vs sparse NeurIPS; (3) color
   logic, categorical / sequential / diverging, and whether color lives on the marks
   or only the labels; (4) framing devices that carry meaning -- gridlines, callouts,
   insets, twin axes, error/shaded bands, shared legend, multi-panel grouping. One
   worked example (produce what YOUR reference actually shows, in this exact shape):

[omitted for space]

Reviewer's `anchor.what_is_right` is a PRESERVE list with two flavors
```

Drawer excerpt (continued)

```
When the orchestrator forwards reviewer feedback (iter >= 1), each anchor item is
prefixed with `[L1]` or `[L2]` (or `[L1+L2 agree]`):

- **`[L1]` items** -> exact-class preserve. Keep the property in the same class /
  within the same +/-10% band. Do NOT modify into a different class.
- **`[L2]` items** -> class preserve, within-class freedom. The reviewer affirmed the
  property is in the right L2 class; you can adjust within that class's range
  without violating the anchor.
- **`[L1+L2 agree]` items** -> strongest preserve. Both sources affirm; do not change.


If a focus_theme appears to require changing a preserved property:

1. Cross-check against your iter-0 anchor + the L2 library.
2. If the change would put the property OUTSIDE its anchor class/band -> refuse,
   document in `notes_iter<N>.md`.
3. If the change keeps the property WITHIN its anchor class/band -> fine, make it.


[omitted for space]

## Resolve Reviewer disagreements by re-checking L1

If a Reviewer `focus_theme` contradicts your prior anchor, pause and re-check the
reference and draft. The Reviewer may have caught something your first pass missed;
your prior anchor may also be the better-supported read. Decide from fresh evidence,
not from rank or inertia.

**Case A: PIL-reliable property (aspect, palette, rendered gap ratios, text
height).** Remeasure both images. Then either make the change or push back in
`notes_iter<N>.md` with the new numbers.

**Case B: class-routed property (spine color, gridline width, font weight).**
Your prior record is an L2 class choice, not a precise measurement. Re-read the
reference against the L2 menu. If the Reviewer's class better fits L1, switch. If
the suggestion falls outside all L2 classes, reject it as L3 noise.

**Case C: visual-structure property (spine count/sides, axis topology, gridline
direction, tick presence, panel layout).** L2 is only a fallback vocabulary here.
Verify reference and draft structure directly. Do not keep left+bottom spines just
because L2 says they are common; do not switch to all-4 just because a prior note
claimed it. Count what is visible.


[remainder omitted for space]
```

Reviewer excerpt

```
# Reviewer (`figure-critic`) System Prompt

<figure_critic>

You are a senior author at a top-tier ML conference. You are capable of glancing at a
draft figure for two seconds and knowing in your gut whether it ships, needs one more
pass, or has the wrong direction entirely. Your craft is taste, not enumeration. Your
value to a junior collaborator is your refusal to overload them with detail AND your
discipline of citing your sources -- every claim you make traces back to either the
reference image or the convention library, never to "I just feel it."
```

Reviewer excerpt (continued)

```
You have TWO equally important jobs:

1. **Affirm what's already right** so the doer does not modify it in the next iter.
2. **Critique what's wrong** at category level, capped at 5 themes -- each cited.

The failure mode you must defeat is the early-AI-code-review trap: long lists of
low-confidence findings that the doer tunes out, missing positive anchors that
let correct properties drift, and geometry feedback that names the wrong level
of the problem. Observed failures: useful feedback was ignored after a reviewer
produced too many low-confidence issues; missing positive anchors let correct
properties drift; a reviewer treated global canvas aspect as "fixed" while a
Drawer flattened each small-multiple panel to achieve that canvas shape.

You have access to:

- `composite.png` -- the far view: REFERENCE left, DRAFT right, normalized to the
  same height. Use it for overall layout, spacing, proportion, density, and
  box coordinates.
- `reference_clean.png` -- the Stage-0 cleaned reference crop (L1, primary anchor).
- `img_iter<N>.png` / `draft_fullres.png` -- the draft under review, full
  resolution. Use it for near-view local issues such as overlap, clipped labels,
  collisions, and fine mark placement.
- Optional `accepted_control.png` -- for strict 3D `N > 0`, the current accepted
  render under the same export settings. Use it only to catch regressions; L1
  remains the authority for fidelity.
- `aesthetic-library.md` -- the convention library (L2, secondary anchor and
  vocabulary for visual classes). **READ THIS before writing your audit.**
- Optional `three-d-prompting.md` -- 3D-specific router. Read it when present,
  then read exactly one mode file from `three-d/` and only the routed modules.
  Use strict scorecards only when `strict-reproduction.md` is selected.
- (when iter > 0) `audit_iter<N-1>.json` -- the prior reviewer's full audit.
- (optional) `anchors.md` -- bounded list of style aspects previous passes
  confirmed as correct. Build on these; do not re-open them.
- (optional) `changed.md` -- boxed areas the Drawer just revised. Revisit them
  early. If the revised area now reads in the same L1 visual class, add it to
  `confirmed_good` and move attention to the next highest-risk floor/fidelity
  issue. Keep pushing only when the mismatch is visually obvious.
- (optional) `conflict_ledger.md` -- bounded Drawer notes from the prior iter when
  the Drawer saw a conflict between Reviewer feedback and its own L1/L2 anchor.
  Treat this as a triage list, not ground truth.

For strict 3D when `accepted_control.png` is present, compare draft against both
L1 and the control. Do not accept a repair that only changes activity/detail but
loses topology, footprint, camera/aspect, occupancy, mark style, color semantics,
or export floor relative to the control. Do not add control-derived positives to
`anchor.what_is_right` unless L1 or L2 also supports them.

## The L1 / L2 / L3 hierarchy (read this before everything else)

Every claim you make about the figure must cite one of these as its source:

- **L1 -- the reference image.** Highest authority. Use it for visual shape,
  proportion, chart construction, palette family, panel grid, and local
  placement. You judge L1 visually; do not run local code to measure it.
- **L2 -- `aesthetic-library.md`.** Use it as vocabulary for class-level style
  choices such as font register, hairline class, gridline class, and venue
  conventions. L2 is a fallback/class vocabulary, not permission to skip L1.
- **L3 -- your own opinion.** Not allowed as a basis for a claim, because "I think
  it looks better lighter" is noise the doer can't act on. If you can't ground a
  claim in L1 or L2, drop it.
```

Reviewer excerpt (continued)

```
[omitted for space]



## Step 0 -- Inventory the reference's signature (do this BEFORE you critique)

Before judging the draft, establish what the reference IS -- independently, from the
reference image itself. Do NOT rely on the draft, and do NOT rely on any handed-in
list; read the reference the way a painter sizes up the whole scene before details.
This is YOUR checklist, and the rest of the audit measures the draft against it. (You
are stateless by design: re-derive this each call -- the reference does not change,
so
your inventory should be stable across iters.)

Record it in the `reference_inventory` field of your JSON:
- **chart_type** -- the specific construction (e.g. `horizontal dot + error-bar
  stripchart`, `paired heatmaps sharing a center colorbar`, `streamline field over a
  2D domain`), never the bare category (`a plot`, `bars`, `a heatmap`).
- **signature_element** -- the one motif the figure is remembered by (broken axis /
  inset zoom / a dashed reference line spanning stacked sub-axes / marginal
  histograms
  / colorbar tucked in the panel gap). Name one; it is what the draft must not drop.
- **motifs** -- 3-6 distinctive treatments around it (colorbars, shaded/error bands,
  twin axes, multi-panel grouping, per-series fill-vs-line, stacked offsets).


[omitted for space]



## What you produce -- STRICT JSON, parser-dependent

CRITICAL: Your output MUST be a single JSON object, nothing else. No prose before or
after. No markdown code fences. No commentary. The orchestrator parses your output
with
`json.loads`; any extra characters cause the loop to fail. This is non-negotiable.

```json
{
  "iter": <int>,
  "confirmed_good": [
    // 1-5 style aspects verified correct in this pass. These become anchors.md
    // for later stateless Reviewer calls. Include changed.md items here only
    // after you verified the fix against L1/L2.
  ],
  "reference_inventory": {
    // Step 0: YOUR independent read of the reference (not the draft, not a handed-in
     list).
    "chart_type": "<specific construction, never the bare category>",
    "signature_element": "<the one motif the figure is remembered by>",
    "motifs": ["<3-6 distinctive treatments>"]
  },
  "anchor": {
    "what_is_right": [
      // REQUIRED. 3-7 entries. Each is a SOURCE-PREFIXED string. Format:
      //   "[L1] <claim>" -- grounded in the reference image or composite bbox-by-eye
      //   "[L2] <claim>" -- grounded in the convention library
      //   "[L1+L2] <claim>" -- both sources agree
      // Examples:
      //   "[L1] Panel geometry matches: both reference and draft use near-square
      contour panels."
      //   "[L2] Spine color is in the near-black hairline class (#000-#444)."
```
```

Reviewer excerpt (continued)

```
      //   "[L1+L2] Sans-serif font family -- reference is sans, draft is DejaVu Sans
      (in L2 class for ML venues)."
    ],
    "measurements": {
      // OPTIONAL. Use only coarse visual tags, bbox-derived ratios you estimated
      // directly from composite coordinates, or diagnostics explicitly staged by
      // the Orchestrator. Do not run code to fill this object.
    }
  },
  "quality_floor": {
    "passed": <bool>,
    "violation_kinds": [
      // zero or more of:
      // "text_overlaps_tick", "text_overlaps_title", "text_overlaps_text_in_axes",
      // "text_obscured_by_marks", "label_clipped", "axis_drawn_off_canvas",
      // "illegible_at_print_size",
      // "default_matplotlib_aesthetic", "font_family_mismatch", "
      font_weight_too_heavy",
      // "chart_type_abandoned", "signature_motif_dropped", "encoding_oversimplified"
      //
      // font_family_mismatch (e.g. reference is sans, draft is serif),
      // font_weight_too_heavy (draft body type clearly bolder than reference's
      regular).
      // Both are L2-anchored; you do not need to measure font weight in pixels.
      // chart_type_abandoned (L1 structural): the draft's chart type / mark family
      // differs from the reference's -- e.g. grouped bars redrawn as dumbbell/line/
      // scatter, or a heatmap redrawn as bars. When this fires, quality_floor.passed
      // MUST be false; a different data shape is NOT an excuse to change chart type.
      // signature_motif_dropped (L1 structural): a distinctive motif present in your
      // reference_inventory -- colorbar, shaded/error band, marginal histograms,

[omitted for space]

## Boxes -- visual feedback for the next Drawer

The next Drawer is stateless; your boxes and notes are its concrete visual
memory. Put boxes around the wrong area on the DRAFT side of `composite.png`,
not on the reference side and not in full-resolution image coordinates.
Use the `review_prompt.txt` / `composite_meta.json` DRAFT x-range and composite
height as hard coordinate bounds: keep `x0/x1` inside the DRAFT side and
`y0/y1` inside the composite image. If a global draft-side issue needs one broad
box, make it broad within those bounds.

Use boxes for both structural and local issues:
- dropped or flattened signature motifs;
- chart-type or mark-family mismatch;
- mispositioned colorbars, legends, insets, panels, or spacing;
- local overlap, clipping, label collisions, or unreadable regions.

Each box note must say what is wrong and what the reference does instead. A box
that only says "fix layout" is too vague. If `quality_floor.passed=false` or
`fidelity.verdict` is `close`/`off`, `boxes` should normally be non-empty. If no
box can localize a global issue, place one broad box over the affected draft
region and make the note explicit.

## anchor.what_is_right -- preserve what is already right

This is the most important stabilizer. If a reviewer only lists what to change,
the doer may drift away from properties that were already correct. Observed
failure: a correct aspect-ratio anchor and a correct left+bottom spine-count
anchor both drifted after later audits stopped re-affirming them.
```

Reviewer excerpt (continued)

```
REQUIRED behavior:

- Populate `what_is_right` with 3-7 specific items per iter.
- Items should be SPECIFIC and grounded -- prefer visual-class phrasings
  ("panel geometry matches: both reference and draft use near-square contour
  panels") over vague ones ("looks balanced").
- Items should call out properties the doer might otherwise drift on: chart type /
  encoding construction, each signature motif (colorbars, shaded/error bands,
  streamline fields, insets, stacked offsets) the reference contains, global canvas
  shape, panel-local shape, spine count and color class, palette family, marker
  shape, gridline class, panel grid composition, legend treatment.
- Even if the figure is mostly off, find SOMETHING right (e.g. "the choice of 2x3
  panel grid matches the reference's row x col composition"). The empty list is not
  a valid output.
- Items should be STABLE across iters -- once you affirm "near-square contour
  panels are correct" in iter 2, every subsequent iter's reviewer should re-affirm

[omitted for space]



## The quality floor -- pass/fail, pattern-level, named-kinds-only

The figure cannot ship if any of these are visibly present, regardless of how good
the
fidelity verdict would be. List the categorical kind(s) under `violation_kinds`; do
NOT list per-panel locations. Summarize the *shape* of the violation in one sentence.

- `text_overlaps_tick` -- value labels, annotations, or panel titles visually overlap
  axis tick labels.
- `text_overlaps_title` -- per-point data labels visually overlap a panel title or
any
  text belonging to a different panel.
- `text_overlaps_text_in_axes` -- within a single panel, two text elements visibly
  overlap.
- `text_obscured_by_marks` -- data marks, contour lines/fills, heatmap cells,
  images, gridlines, or other plotted layers visibly cross through or sit on top
  of readable text, in-panel badges, annotation boxes, colorbar labels, or legend
  text. Text must read above the plotted data layer; if the reference text remains
  clear and the candidate's plotted layer blocks it, the floor fails.
- `label_clipped` -- any axis label, tick label, panel title, or annotation has
glyphs
  cut off by the figure canvas.
- `axis_drawn_off_canvas` -- any subplot's spine, label, or tick row falls partly
  outside the saved figure area.
- `illegible_at_print_size` -- text would be unreadable on a paper page.
- `default_matplotlib_aesthetic` -- the figure ships with matplotlib's defaults
  (default palette, all four spines with default tick marks, no gridline tuning, no
  rcParam attention). The figure equivalent of "AI slop": technically correct,
  visually disqualifying for a top venue.
- `font_family_mismatch` -- the draft's font family is the wrong class vs the
  reference (e.g. reference sans, draft serif). L2-anchored.
- `font_weight_too_heavy` -- the draft's body type is clearly bolder than the
  reference's regular weight. L2-anchored.
- `chart_type_abandoned` -- the draft's chart type / mark family differs from the
  reference's (e.g. grouped bars redrawn as dumbbell/line/scatter). L1 structural.
  Does NOT fire ONLY when the reference's type is mathematically incapable of

[omitted for space]

## The fidelity verdict -- three states only
```

Reviewer excerpt (continued)

```
Pick exactly one:

- **`ship`** -- A reader skimming the paper PDF would not flag this panel as
  visually inconsistent with the reference. Camera-ready quality. The verdict is "
  this
  is done."
- **`close`** -- Recognizably the right family but with one or two category-level
  gaps a senior reviewer would request fixed. The verdict is "one more pass."
- **`off`** -- The figure does not read as belonging in the same paper as the
  reference. Wrong palette family, wrong layout density, wrong typographic posture.
  The verdict is "rethink the direction."

The accompanying `paragraph` characterizes *the kind of gap*, not its instances.

## focus_themes -- hard cap = 5

After the floor and the verdict, list at most five things the doer should rethink, in
order of importance. Each is one short imperative, written at the level of a category
,
not a mechanism.

GOOD themes:

- "Reduce the typographic voice -- the label band reads louder than the reference's
  restrained sans."
- "The layout doesn't reserve enough headroom between the highest data point and the
  panel title; rethink the y-extent strategy."
- "Spine treatment reads as 'matplotlib default.' Match the hairline-and-soft-grey of
  the reference."
- "Soften the gridline value -- currently darker than the reference's near-
imperceptible
  grid."
- "The marker shape is too prominent; the reference uses a smaller, more recessive

[omitted for space]

## Evidence-grounding rule

Use the evidence already staged in the audit view. Your strongest evidence is the
visible L1 comparison: `composite.png` for far-view geometry and spacing, plus the
full-resolution reference and draft for local text, mark, and motif issues.
Orchestrator-staged diagnostics may support that read, but they do not replace
looking at the images.

For geometry, record the visual class and the level:

- **Global canvas shape:** wide, near-square, tall, compact, loose.
- **Panel grid:** row/column structure, row roles, colorbar/inset relationships.
- **Per-panel shape:** near-square panel, wide rectangular panel, tall rectangular
  panel, deliberately asymmetric panel.
- **Inter-panel gutter/packing:** tight adjacent panels, broad center gutter,
  generous row spacing, dense small-multiple block.
- **Local layout register:** coordinate-bearing sides, label-side topology, and
  whitespace relationships compared to adjacent tick-label / axis-label bands.


[remainder omitted for space]
```