PRISMACODER: PUSHING THE BOUNDARIES OF OPEN MULTIMODAL CODE INTELLIGENCE THROUGH DECOMPOSED RUBRIC SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal code intelligence connects the symbolic world of programs with the visual world their execution creates. Interest in this capability is burgeoning, as models are increasingly asked to produce charts, webpages, animations, and semi-structured documents. Recently, proprietary models have pushed far ahead, producing interactive artifacts even from a text description or a screenshot alone. Yet their data and training remain closed, and open progress is held back by the quantity and quality of available data and by the pipelines and methods for producing it. Quality is the harder of the two, since a sample must be correct as a program, well built, and above all visually right once rendered. No existing signal covers all three. Execution often returns just a pass or fail, code similarity does not map to the visual output, and task-specific verifiers do not generalize. To push open progress further, we start from the data side with PRISMAFORGE, a hierarchical data pipeline centered on a synthesized rubric that decomposes these factors into checkable criteria. The same rubric serves as both gatekeeper and reward: it curates the supervised corpus, then drives reinforcement learning on visual-coding tasks. Leveraging PRISMAFORGE for data synthesis and filtering, we build PRISMACODER-2M, the most diverse open dataset for multimodal code intelligence and the largest to date. On the model side, we train PRISMACODER-9B and PRISMACODER-35B on this data and release both. They lead the open state of the art by a wide margin and rival leading commercial systems. Extensive analyses across models further verify the effectiveness of our pipeline, rubrics, and data. We release our code and data at this link.

1 INTRODUCTION

From symbols to pixels, code intelligence now extends beyond textual source code into the visual world that programs produce (Zan et al., 2023; Sun et al., 2024; Zhao et al., 2026). A growing range of applications specifies intent through images and judges a result by what its code renders, from charts and web pages to diagrams, typeset documents, and animations. Leading proprietary systems now produce interactive artifacts from a text description or a screenshot alone.

Prior open efforts have mostly addressed one corner of this space at a time, from plotting (Ni et al., 2025b; Galimzyanov et al., 2025; Yang et al., 2024) and chart-to-code (Yang et al., 2025a; Zhao et al., 2025c) to screenshot-to-page generation (Gui et al., 2025; Si et al., 2025). Each comes with its own data and recipe, making it difficult for the resulting models to generalize across task domains and output formats. Recently, open work such as JanusCoder (Sun et al., 2026b) has brought many of these scenarios under a shared visual-programmatic interface, with follow-up refinements building on the same view (Zhao et al., 2025a; Jiang et al., 2026; Ni et al., 2026). Despite these advances, publicly available training data remains limited in scale and uneven in quality, with only a handful of open releases to date (Sun et al., 2026b; Jiang et al., 2026). The pipelines used to build these datasets are rarely released, while proprietary systems keep their data and training closed. Moreover, existing public data are typically curated with either a single per-sample score, which provides limited diagnostic information about errors, or a task-specific verifier, which does not generalize across tasks.

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.

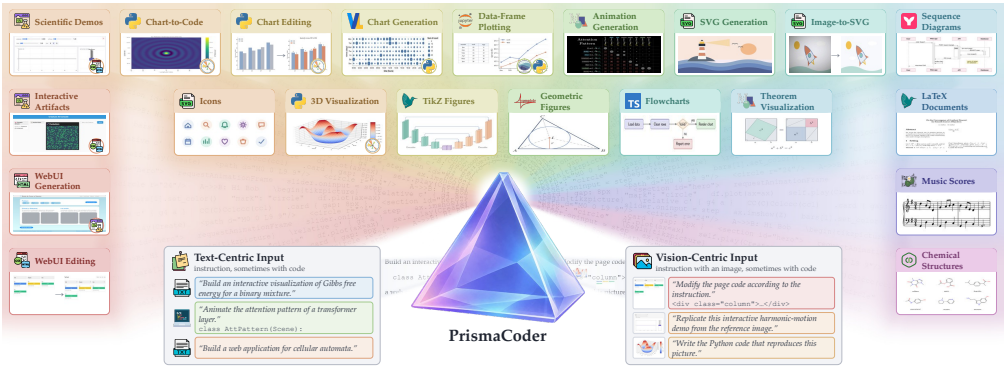


Figure 1: The span of multimodal code intelligence in PRISMACODER-2M and PRISMACODER, across scenarios and formats.

To address this gap, we introduce PRISMAFORGE, a rubric-based framework for multimodal code data curation and policy optimization. Rather than relying solely on a holistic judgment or a fixed checklist, PRISMAFORGE applies a common evaluation procedure across data sources and task domains while tailoring its criteria to each instruction. Each rubric combines reusable domain-level checks with instruction-specific requirements, with the number of criteria adjusted to the estimated task difficulty. Code-level and visual-level criteria are evaluated separately on the generated program and its rendered artifact, respectively. The resulting scores guide data curation and serve as rewards for tasks using rubric-based reinforcement learning.

Using PRISMAFORGE, we curate PRISMACODER-2M, the broadest open dataset to date for advancing multimodal code intelligence, spanning more than 12 categories and 10 programming languages with both text-centric and vision-centric data. To build this coverage, we draw seeds from complementary sources: public visual-coding corpora cover common plotting and graphics tasks; research figures and open-source repositories contribute long-tail scientific examples; and web, interactive-demo, and chemical-structure sources broaden the range of artifacts. We then convert the seeds into self-contained tasks, check candidate programs through format-specific execution and rendering, and use instruction-specific rubric scores to filter or refine the resulting examples. This gives the dataset both breadth of coverage and a consistent quality-control process.

To assess whether the breadth of PRISMACODER-2M translates into broad model capability, we develop PRISMACODER-9B and PRISMACODER-35B. Evaluated on a wide range of representative benchmarks, PRISMACODER leads the existing open state of the art by a wide margin and matches the proprietary systems to a considerable degree (§5). Through extensive analyses (§6), we show that the data and pipeline generalize across base model beyond our release. We further show that rubric-guided curation improves model performance relative to direct-judge selection and that rubric-based scores align closely with human judgments.

Our main contributions are summarized as follows:

- **Decomposed Rubric Synthesis for Visual-coding Data.** We introduce PRISMAFORGE, a principled pipeline whose hierarchically synthesized rubrics serve as both filter and reward. Based on it, we curate PRISMACODER-2M, the broadest and largest open dataset to date.
- **Open Models that Rival Proprietary Systems.** We build PRISMACODER-9B/35B on PRISMACODER-2M, which lead the open state of the art by a wide margin and match or surpass leading proprietary systems on charts, front-end pages, and semi-structured documents.
- **Empirical Insights across Data, Modeling, and Training.** Beyond data and model contributions, we further share insights through extensive analysis. It shows that our pipeline generalizes across scenarios and backbones and yields artifacts better aligned with human preference.

2 RELATED WORK

Multimodal Code Intelligence. Code intelligence now extends beyond text-only programming to visual-programmatic tasks, where intent may be visual (Li et al., 2024; Zhang et al., 2024a; Wang

et al., 2025) and outputs can be judged only after execution or rendering (Zhao et al., 2026). On the output side, syntactic validity is insufficient; correctness also depends on layout, semantics, interaction, and editability. On the input side, reading values and layout off renders remains a perception bottleneck (Masry et al., 2022; Wang et al., 2024; Xia et al., 2025) across vision-language models (Zhang et al., 2024b; Liu et al., 2025). The tasks run from screenshot-to-code (Gui et al., 2025; Si et al., 2025; Yun et al., 2024) to front-end engineering (Jiang et al., 2025; Zhu et al., 2025; Sun et al., 2025a) and generative UI (Zheng et al., 2026); chart generation (Yang et al., 2025a; Zhao et al., 2025c; Wu et al., 2025) and editing (Zhang et al., 2024c; Zhao et al., 2025b); SVG and symbolic graphics (Yang et al., 2025b; Nishina & Matsui, 2024; Xu et al., 2024) with their structural probes (Qiu et al., 2025; Chen et al., 2025b); and interactive artifacts and code-driven demonstrations (Zhang et al., 2025; Ku et al., 2025; Sun et al., 2025b). Recent generalists unify these domains via JanusCoder's visual-programmatic interface (Sun et al., 2026b), VisCodex's vision–code integration (Jiang et al., 2026), VisCoder2's agentic visualization coding (Ni et al., 2026), and VinciCoder's visual reinforcement learning (Zhao et al., 2025a). Yet a shared modeling interface does not by itself make heterogeneous notions of correctness consistently measurable or usable for training (Zhao et al., 2026). Complementing this progress, we contribute on the data side, with PRISMACODER-2M spanning the broadest scenarios and open data types among these works.

Learning Signals for Code Generation. Conventional code tasks (Zhang et al., 2023) draw training data from instruction synthesis and evolution (Wei et al., 2024; Luo et al., 2024; 2025b). For many such tasks, executable tests provide a cheap, reproducible signal for data filtering (Xu et al., 2025b; Sun et al., 2026a) and reinforcement learning (Shao et al., 2024; Xu et al., 2025a; Zhuo et al., 2025). Visual outputs break this assumption: successful execution confirms only that the code runs, not that its render is faithful, well composed, or semantically correct (Zhao et al., 2026; Liu et al., 2026; Guo et al., 2025). Visual benchmarks instead use task-specific render checks for charts (Yang et al., 2025a), visual and structural similarity for web pages (Si et al., 2025), or judge protocols for interactive artifacts (Zhang et al., 2025); effective individually, these measures do not compose across artifacts. Recent work moves toward finer supervision, from JanusCoder's single-pass rubric over fixed dimensions (Sun et al., 2026b) to VinciCoder's coarse-to-fine rewards (Zhao et al., 2025a). Neither signal is yet both fine enough to localize failures and general enough to cover every task, at curation and at training alike. In this work, one synthesized rubric per instruction serves both roles, filtering the training data and rewarding reinforcement learning across every format we cover.

3 PRISMAFORGE

In this section, we present PRISMAFORGE, a rubric-based scoring framework that lies in the center of our data curation and training pipeline.

3.1 ELASTIC RUBRIC SCORER

For a large-scale coding dataset prepared for visualization or demonstration, the key challenge is to determine which candidates are sufficiently correct and useful to retain. Unlike problems with easily verifiable outputs, visualization quality is often specified only implicitly (Wu et al., 2025; Yang et al., 2025a). A single holistic score is therefore hard to verify and offers little insight into why a candidate succeeds or fails (Ye et al., 2024; Sun et al., 2026b).

A natural alternative is to evaluate candidates using explicit criteria, but a fixed rubric is too coarse for diverse visualization tasks. Different instructions may prioritize numerical accuracy and axis scales, visual similarity, layout, or artifact editability. A universal rubric must stay broad and may miss the properties most important for a specific example. At the other extreme, manually designing separate rubrics for every visualization instruction would require substantial human effort and is hard to scale.

To balance these two extremes, we introduce a sample-specific rubric system that combines reusable domain-level criteria with criteria synthesized specifically for each instruction.

Formally, let $i \in \{1, \ldots, N\}$ index dataset instances, and let $d_i \in \mathcal{D}$ denote the visualization domain associated with instance i. We represent candidate sample i as a tuple (x_i, c_i, v_i) where x_i denotes the instruction, which may contain text and optional visual references; c_i denotes the response code produced for that instruction; and $v_i = \mathrm{Exec}(c_i)$ denotes the visual artifact obtained by executing c_i.

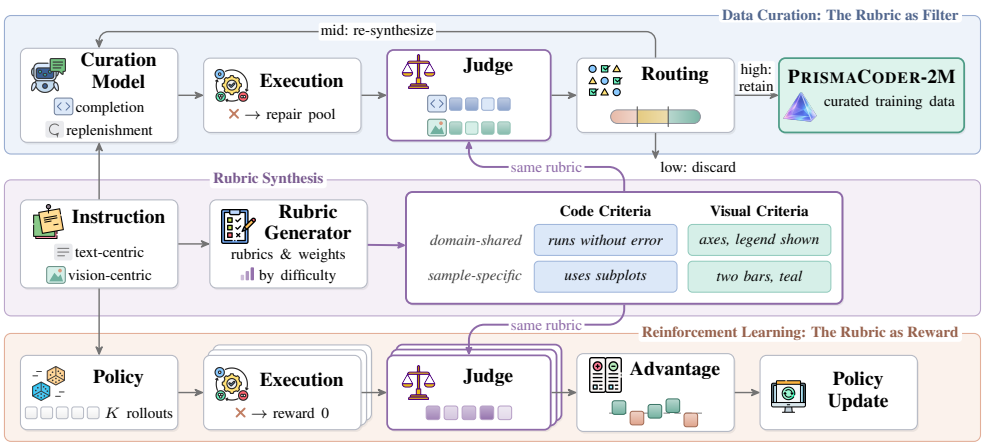


Figure 2: PRISMAFORGE writes one rubric per instruction and uses it twice, to filter candidate data into PRISMACODER-2M and to reward rollouts in reinforcement learning.

We evaluate each candidate using two complementary sets of rubric criteria: code rubrics $\mathcal{R}_i^{(c)}$, which assess properties that can be verified from c_i, and visual rubrics $\mathcal{R}_i^{(v)}$, which assess properties directly observable from v_i.

Within each modality, we further distinguish between domain-shared and sample-specific criteria. Domain-shared rubrics encode requirements that recur across samples within the same visualization domain, whereas sample-specific rubrics are derived from the instruction x_i and capture requirements unique to the current sample. Thus, for modality $m \in \{c, v\}$,

$$\mathcal{R}_i^{(m)} = \mathcal{R}_{\text{shared}}^{(m)}(d_i) \cup \mathcal{R}_{\text{spec}}^{(m)}(x_i). \tag{1}$$

Because individual criteria may differ in importance, each rubric item $r \in \mathcal{R}_i^{(m)}$ is assigned a nonnegative, sample-dependent weight $w_{i,r}^{(m)}$.

To score a candidate, let $f^{(m)}(r, y) \in [0, 1]$ denote the judge score indicating how well artifact y satisfies rubric criterion r under modality m. The normalized score for modality m is then

$$S_i^{(m)} = \frac{1}{\sum_{r \in \mathcal{R}_i^{(m)}} w_{i,r}^{(m)}} \sum_{r \in \mathcal{R}_i^{(m)}} w_{i,r}^{(m)} f^{(m)}(r, y_i^{(m)}) \tag{2}$$

The two modality scores are then combined into an overall sample score with appropriate weighting

$$S_i = \lambda_c S_i^{(c)} + \lambda_v S_i^{(v)}, \tag{3}$$

where $\lambda_c, \lambda_v \geq 0$ and $\lambda_c + \lambda_v = 1$.

In practice, candidate samples vary substantially in difficulty and complexity, and therefore should not be evaluated with the same number of rubric items. We first assign each instruction x_i a difficulty level $q_i \in \mathcal{Q}$ and use this level to guide rubric generation. Each difficulty band is associated with a bounded range for the number of generated criteria. This allows more complex samples to receive finer-grained evaluation while keeping the rubric compact enough to control judging cost.

3.2 RUBRIC-BASED POLICY OPTIMIZATION

The same rubric framework can also be used to construct rewards for reinforcement learning. Because rubric scores are normalized within each modality and aggregated using a common scoring rule, they provide a consistent reward scale across samples with different numbers and types of criteria. We therefore use the rubric-based evaluator as a structured reward function during policy optimization.

For each prompt x_i, we sample K rollout responses from the behavior policy

$$c_i^{(1)}, \cdots, c_i^{(K)} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x_i). \tag{4}$$

Each rollout $c_i^{(k)}$ is executed to obtain the visual artifacts $v_i^{(k)} = \text{Exec}\left(c_i^{(k)}\right)$. Rollouts that failed to execute or render recieve reward 0.

The judge model then scores these rollouts. For each successful rollout trajectory $(x_i, c_i^{(k)}, v_i^{(k)})$, we apply its accompany rubrics set $(\mathcal{R}_i^{(c)}, \mathcal{R}_i^{(v)})$ and obtain the aggregated score $s_i^{(k)}$ via Eq. 2–Eq.3. These scores are then leveraged to compute the within-group normalized advantage

$$\hat{A}_i^{(k)} = \frac{s_i^{(k)} - \mu_i}{\sigma_i + \epsilon} \tag{5}$$

where $\mu_i = \frac{1}{K}\sum_{k=1}^{K} s_i^{(k)}$ and σ_i denotes the standard deviation of $s_i^{(k)}$.

With this reward formulation, the policy can then be optimized with the standard GRPO clipped objective (Shao et al., 2024). In practice, we also used GSPO objective (Zheng et al., 2025) with Rollout Routing Replay techniques (Ma et al., 2025) for the MoE models to stabilize experts training.

4 PRISMACODER-2M

We build PRISMACODER-2M with PRISMAFORGE by collecting seeds from complementary sources, gating them by rubric score into a supervised corpus, and holding out a subset for reinforcement learning, where the same rubrics supply the reward.

4.1 DATA SOURCING

Multimodal code generation needs several capabilities at once, so we collect four kinds of seeds that each cover one of them and train on them together (Table 1 and Fig. 3).

Visualization Fundamentals. Visual coding starts from a core of plotting skills, such as mapping data to axes and marks and composing legends and layered plots. Public plotting datasets build this foundation with instruction–code and visualization–code pairs in Python (Hunter, 2007) and in SVG, Mermaid, Vega-Lite, and other visualization languages (Ni et al., 2025b; 2026; Jiang et al., 2026).

Diverse Scientific Artifacts. Scientific work produces more varied visuals than standard charts, from multi-panel figures to domain-specific encodings. Plotting code from open-source GitHub repositories and figures from arXiv papers bring this variety into the data, and an AST-based pass splits long or monolithic scripts into self-contained units.

Front-End Layout and Interaction. Front-end development asks a model to arrange many elements on a page and to make them respond to users. Web pages from WebCode2M (Gui et al., 2025) and the MCD dataset (Jiang et al., 2026) supply the layout, while interactive demos supply the behavior. The demos are adapted from prior releases (Sun et al., 2026b) or rewritten from static pages (Laurençon et al., 2024) and complex generation prompts (Luo et al., 2025a; Ni et al., 2025a) into tasks of varying visual, functional, and reasoning complexity.

Symbolic Notation. Science, mathematics, and music communicate through symbolic notation, where a single wrong symbol breaks the result. Chemical-structure code seeded from PubChem (Kim et al., 2025) must describe a valid molecule, and LaTeX, Asymptote, LilyPond, and Manim programs must compile into well-formed documents, figures, scores, and scenes. The first three formats draw on the same sources as the plots, and the Manim seeds come from the 3Blue1Brown video code. Per-source details are presented in Appendix §B.

4.2 ITERATIVE DATA GATING

Every seed enters PRISMAFORGE as a candidate (x_i, c_i, v_i). Where a seed lacks working code, a curation model $\mathcal{L}^C$ generates it, $c_i \sim \mathcal{L}^C(\cdot \mid x_i)$, through a code-data synthesis pipeline. PRISMAFORGE works with any such pipeline, and we use a typical open-source one (Sun et al., 2026b). Each candidate must then compile or render through a per-format gatekeeper, $v_i = \text{Exec}(c_i)$, and candidates that still fail after regeneration are set aside in a repair pool.

Table 1: Statistics of PRISMACODER-2M.

Task family	#Samples	Share
Vision-centric		
WebUI Generation	357.1K	17.8%
Chart-to-Code	196.5K	9.8%
Image-to-SVG	181.0K	9.0%
WebUI Editing	175.0K	8.7%
Demo / Artifacts	163.3K	8.1%
Chemical Structures	108.0K	5.4%
Interactive Artifacts	9.1K	0.5%
Text-centric		
Chart Generation	181.1K	9.0%
Demo / Artifacts	163.0K	8.1%
Text-to-WebUI	135.2K	6.7%
LaTeX	133.1K	6.6%
SVG Generation	53.3K	2.7%
Animation (Manim)	35.9K	1.8%
JavaScript	28.5K	1.4%
Mermaid	22.7K	1.1%
LilyPond	21.9K	1.1%
Vega-Lite	20.0K	1.0%
Asymptote	19.4K	1.0%
Total	2M	100%

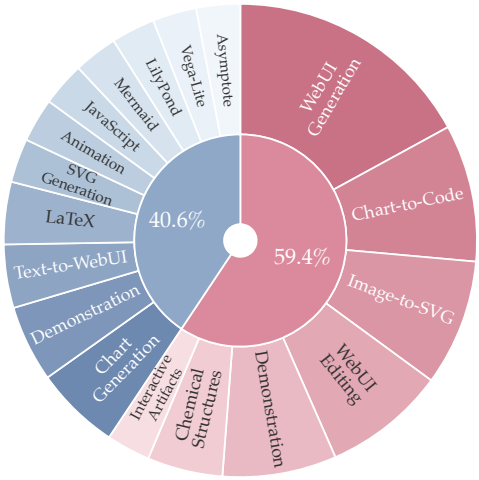


Figure 3: Composition of PRISMACODER-2M by input modality and task family.

A rubric generator $\mathcal{L}^R$ assigns each candidate its difficulty level and its code and visual rubrics (§ 3.1), and a separate judge $\mathcal{L}^J$ scores the code against $\mathcal{R}_i^{(c)}$ and the render against $\mathcal{R}_i^{(v)}$. The modality scores and the overall score S_i follow Eqs. 2 and 3. We use distinct models for rubric generation and judging so that biases in rubric construction are not reproduced during scoring.

We then gate candidates by their overall scores. Those with $S_i < \tau_1$ are discarded, those with $\tau_1 \le S_i < \tau_2$ are sent back for re-synthesis, and those with $S_i \ge \tau_2$ are retained in the curated pool.

The synthesis pipeline serves this gate in two ways. It re-rolls the code of a mid-band candidate with the failed rubric items as feedback, and it synthesizes new samples seeded by retained candidates. Both kinds of output re-enter the pipeline at the execution gate and are scored and routed like any other candidate. For the final supervised corpus, we retain a sample only if its score meets the threshold for its domain, $S_i \ge \tau_{d_i}$.

4.3 RL DATA CURATION

Reinforcement learning draws on a compact set of instructions held out from the supervised pool, which SFT never sees. Among them we keep the medium or hard instructions on which the SFT policy sometimes succeeds and sometimes fails. Such instructions sit near the policy's capability boundary, and their rollouts differ in quality enough to give group-relative optimization a contrast to learn from, whereas instructions that every rollout passes or fails provide none.

Formally, let π_{SFT} denote the SFT-tuned policy. For each candidate instance i, we draw K independent responses $c_i^{(k)} \sim \pi_{\mathrm{SFT}}(\cdot \mid x_i)$ and evaluate each rollout using the rubric-based scoring function defined in §3.1. We then measure the informativeness of instance i by the empirical dispersion of its rubric scores

$$U_i = \frac{1}{K}\sum_{k=1}^{K}\left(S_i^{(k)} - \bar{S}_i\right)^2, \quad \bar{S}_i = \frac{1}{K}\sum_{k=1}^{K} S_i^{(k)}. \tag{6}$$

A large U_i means that the policy sometimes succeeds and sometimes fails on the same instruction.

In practice, we run RL on all Python, HTML, and SVG tasks, and for each domain we hold out 20K *medium* or *hard* candidates before SFT. We then draw $K = 5$ rollouts per instance from π_{SFT}, score each rollout against its code and visual rubrics, and retain the 5K instances with the largest U_i. RL training thus concentrates on the policy's capability frontier at a fraction of the rollout cost of a large new candidate pool.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models. We release PRISMACODER-9B and PRISMACODER-35B-A3B (abbreviated as PRISMACODER-35B), post-trained from Qwen3.5-9B and Qwen3.5-35B-A3B (Qwen Team, 2026). We also train Gemma-4-31B-it (Gemma Team, 2026) and Qwen3-VL-30B-A3B-Instruct (Bai et al., 2025). More training details are in Appendix §F.

Data Pipeline Details. We reprocess data from prior open releases (Sun et al., 2026b; Jiang et al., 2026; Ni et al., 2026). PRISMAFORGE treats its models as pluggable, and in this paper we use `Claude-Opus-4.6` (Anthropic, 2026a) as the curation model and `Gemini-3-Pro` (Gemini Team, 2025) as the rubric generator. Both data gating and reinforcement learning use Gemini 3 Flash as the judge. The sources are listed per category in Appendix §B.

Baselines. Besides Qwen3.5, we compare against open multimodal code models, namely JanusCoderV (Sun et al., 2026b), VinciCoder (Zhao et al., 2025a), and VisCoder2-7B (Ni et al., 2026).

5.2 BENCHMARKING

Vision-Centric Settings. The vision-centric suite comprises ChartMimic (Yang et al., 2025a), WebCode2M (Gui et al., 2025), DesignBench (Xiao et al., 2025), Design2Code (Si et al., 2025), and ChemDraw (Zhao et al., 2025a), together with the image-conditioned lenses of InteractScience (Chen et al., 2025a) (VQT) and UniSVG (Li et al., 2025) (ISVGen).

Text-Centric Settings. The text-centric suite comprises ArtifactsBench (Zhang et al., 2025), PandasPlotBench (Galimzyanov et al., 2025), and VisPlotBench (Ni et al., 2026) in eight visualization languages, together with the programmatic lens of InteractScience (PFT) and the text-conditioned lens of UniSVG (TSVGen).

Evaluation Protocol. Evaluation follows each benchmark's official protocol, with deterministic metrics reported alongside judge scores. Since some designated judges share a model family with our pipeline, we check that the gains hold across judges (§6.3). Details are presented in §E.

5.3 VISION-CENTRIC RESULTS

Web Interfaces. PRISMACODER-9B and PRISMACODER-35B lead open baseline on front-end generation, achieving judge scores more than 20 points higher than those of task-specific open baselines (Table 2). PRISMACODER-9B also scores above proprietary model on DesignBench generation and WebCode2M CLIP score. DesignBench editing is the only web-interface metric on which both models score below all Claude 4.6 and Gemini 3 variants.

Table 2: Vision-centric results. **Best** / second among open models.

Model	ChartMimic		WebCode2M			DesignBench		Design2Code		InteractSci.	UniSVG	ChemDraw	
	Low-Level	High-Level	LLM Score	Visual	CLIP	Gen.	Edit.	LLM Score	Visual	VQT	ISVGen	Validity	Tani. Sim.
Open models													
Qwen3-VL-8B-Instruct	65.92	62.81	53.67	73.55	79.16	55.12	8.38	57.47	82.81	34.06	66.27	78.91%	54.79
Qwen3-VL-30B-A3B-Instruct	71.04	67.30	54.80	77.40	79.42	51.58	8.07	57.94	85.11	37.67	74.30	82.03%	54.55
Qwen3.5-9B	58.35	55.55	51.74	81.60	79.93	50.73	8.19	53.12	85.60	27.28	63.09	88.28%	66.79
Qwen3.5-35B-A3B	70.22	63.90	65.32	88.64	85.94	64.70	8.66	67.49	90.98	29.33	62.91	92.97%	79.86
JanusCoderV-7B	75.12	68.97	40.92	72.90	76.00	40.37	7.44	36.49	70.81	27.67	61.60	84.75%	72.81
JanusCoderV-8B	76.54	69.23	35.17	65.95	73.04	34.30	7.26	34.53	68.61	33.32	62.47	86.10%	76.91
VinciCoder-7B	78.30	66.09	43.11	72.42	78.59	43.24	1.40	42.30	76.64	0.00	85.72	93.18%	92.15
VinciCoder-8B	77.59	66.44	46.16	78.26	80.79	48.09	1.71	45.72	81.89	2.33	**87.18**	94.21%	91.02
PRISMACODER-9B	81.57	73.82	**69.72**	**90.41**	**88.46**	**69.95**	**8.93**	**70.51**	**91.54**	51.87	69.64	96.88%	95.94
PRISMACODER-35B-A3B	**83.42**	**80.69**	67.78	89.66	87.78	68.98	8.89	67.55	91.22	**53.54**	84.14	**100.00%**	**97.56**
Proprietary													
`GPT-4o`	76.38	72.02	54.94	83.45	80.22	52.64	8.89	58.48	88.23	46.01	70.98	92.19%	57.80
Gemini 3 Flash	85.12	83.04	69.63	90.41	87.34	69.28	9.45	70.12	92.18	65.25	82.24	63.28%	81.39
Gemini 3.1 Pro	77.96	81.73	70.97	89.96	87.64	69.47	9.57	70.90	90.70	71.05	76.85	93.12%	92.45
Claude Sonnet 4.6	80.23	80.73	68.19	87.76	86.80	66.53	9.24	67.99	90.83	77.36	80.38	88.28%	85.43
Claude Opus 4.6	81.11	79.68	66.38	88.25	87.11	65.69	9.55	66.75	90.19	76.25	78.70	96.88%	77.71

Data Visualization. On ChartMimic, both of our models lead all open baselines. Specifically, PRISMACODER-35B outperforms most proprietary models under both settings. Notably, our 9B model demonstrates the most substantial improvement over its base (the lowest among all evaluated models) and even surpasses most proprietary models. This clearly demonstrates the quality and utility of our data. In addition, the low-level score evaluates rendered text, layout, chart type, and color fidelity, which means our model is more capable of generating those details.

Table 3: Text-centric results. **Best** / second among open models.

Model	Artifacts Bench	PandasPlotBench			Inter-actSci.	UniSVG	VisPlotBench		
		Incorr. Code↓	Visual	Task	PFT	TSVGen	ExecPass	Mean	Good ≥75
Open models									
Qwen3-VL-8B-Instruct	34.71	16.00%	60.01	73.67	15.47%	68.13	56.52%	40.25	33.62
Qwen3-VL-30B-A3B-Instruct	41.70	11.43%	60.05	**80.58**	24.40%	69.33	64.38%	49.28	42.88
Qwen3.5-9B	41.24	24.57%	52.98	65.47	22.40%	69.28	52.22%	41.18	36.71
Qwen3.5-35B-A3B	44.98	15.43%	60.16	75.17	26.67%	**72.15**	63.01%	51.45	48.51
JanusCoderV-7B	31.16	23.43%	53.05	65.77	17.73%	66.43	58.42%	48.10	32.15
JanusCoderV-8B	32.00	26.30%	56.30	70.91	17.60%	62.17	56.21%	47.22	30.87
VinciCoder-7B	11.35	**9.14%**	25.37	51.57	0.40%	64.06	64.10%	23.95	10.11
VinciCoder-8B	11.97	12.00%	25.47	60.79	0.54%	63.45	63.92%	12.55	2.80
VisCoder2-7B	17.51	20.00%	59.48	67.59	1.00%	65.44	66.73%	49.84	42.50
PRISMACODER-9B	45.28	10.29%	60.54	77.94	24.80%	68.49	65.30%	53.99	52.62
PRISMACODER-35B-A3B	**46.01**	9.29%	**62.38**	79.85	**28.00%**	70.72	**71.14%**	**62.47**	**61.49**
Proprietary									
GPT-4o	33.54	8.00%	67.63	82.46	31.07%	72.80	67.99%	56.01	52.75
Gemini 3 Flash	51.42	3.43%	69.95	91.21	28.13%	74.64	76.25%	60.92	58.54
Gemini 3.1 Pro	53.04	1.71%	78.41	94.70	44.13%	74.52	79.57%	69.47	69.15
Claude Sonnet 4.6	64.65	3.43%	74.59	91.93	37.33%	74.93	73.00%	62.20	61.06
Claude Opus 4.6	62.47	1.71%	76.03	94.97	35.47%	75.47	73.09%	65.94	66.03

Scientific and Vector Graphics. On chemical-structure code, PRISMACODER-35B parses every output and reaches the highest Tanimoto similarity in the table (97.56). On InteractScience's visual lens both models lead the open baselines, while image-conditioned SVG is the one vision-centric column led by an open baseline, VinciCoder-8B. Full per-benchmark metrics are presented in Appendix §G, and a chart-to-code case across training stages in Appendix §I.

5.4 TEXT-CENTRIC RESULTS

Interactive Applications and Plots. On ArtifactsBench, both models exceed all open baselines and GPT-4o on the overall score (Table 3). On InteractScience's programmatic lens, PRISMACODER-35B scores 28.00, above the best open baseline and comparable to Gemini 3 Flash. Scaling from 9B to 35B raises the InteractScience PFT score by 3.20 points.

Long-Tail Visual Languages. In VisPlotBench, PRISMACODER-35B leads the reported open baselines on all three metrics (Table 3); its Mean score exceeds the strongest by 11.02 points and is comparable to Claude Sonnet 4.6.

6 ANALYSIS

6.1 RUBRIC EFFICACY

To intuitively demonstrate the advantages of the rubric-based method over direct judging, we fine-tune Qwen3.5-9B on two equal-sized subsets drawn from the same data pool.

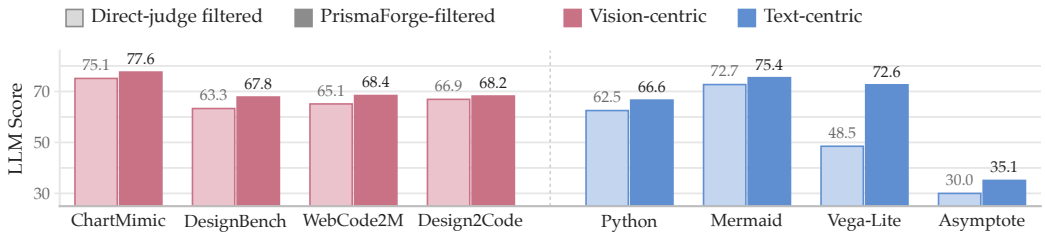


Figure 4: PRISMAFORGE filtering outperforms direct-judge filtering on every benchmark.

One is filtered by direct scoring, where a judge assigns each sample a single holistic score as in a typical pipeline (Sun et al., 2026b), and the other by the PRISMAFORGE rubric. The model trained on the rubric-filtered subset scores higher on every benchmark, by as much as 24 points on Vega-Lite (Fig. 4). The gains are largest on the text-centric formats and small on the vision-centric benchmarks, where Design2Code moves by a fraction of a point. Since both subsets have the same size, the gap reflects which samples each judge keeps.

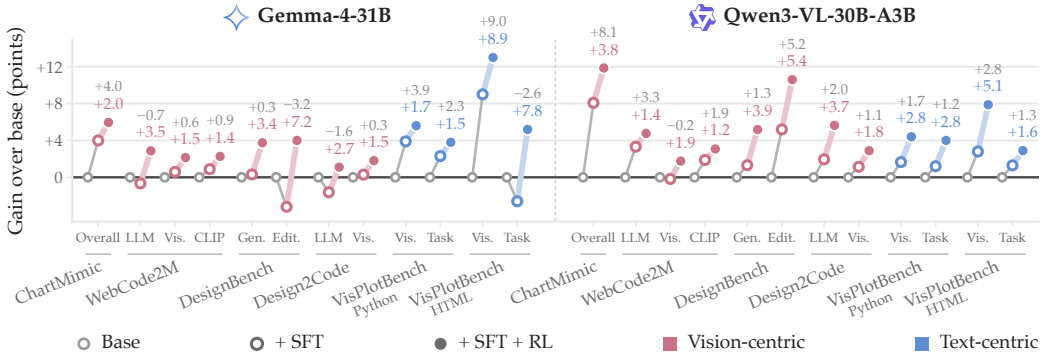


Figure 5: Per-stage gains of our SFT and RL pipeline over two base models on every metric.

6.2 Backbone Generalization

Beyond the released Qwen3.5 backbones, we apply the same data and method to Gemma-4-31B-it (Gemma Team, 2026) and Qwen3-VL-30B-A3B-Instruct (Bai et al., 2025) on Python plotting and HTML, the two most represented domains. On both backbones all twelve metrics end above the base model, and RL adds to SFT on every one (Fig. 5). SFT carries most of the ChartMimic gain, whereas RL drives most of the HTML gains and recovers every metric that dips after SFT alone. More results on Gemma are presented in Appendix §H.1.

6.3 Rubric Agreement with Humans

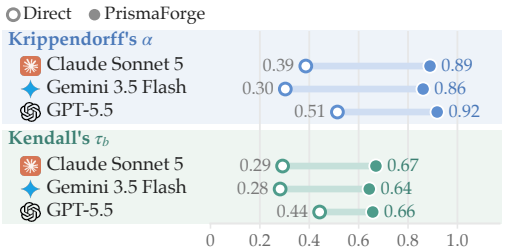


Figure 6: PrismaForge scoring aligns judges with humans better than direct scoring.

Because multimodal code artifacts are ultimately created for humans, consistency with human judgment provides a compelling test to verify that the quality of data yielded by our rubric-based PrismaForge aligns more closely with human preferences. At the same time, preserving the relative ordering induced by scores is particularly important for reward-based policy optimization, where inconsistent rankings can provide conflicting training signals. To evaluate our method along both dimensions, we conducted a human study. We sampled 100 examples from our training data and asked human annotators, as well as three proprietary model judges, to evaluate them using direct scoring and the PrismaForge rubrics, respectively. We then measured the agreement between each model judge and the human ratings in terms of both absolute score levels and pairwise ordering (Fig. 6). The rubric-based judging method achieves substantially higher agreement with human judgments and greater consistency in the induced partial ordering across all three model judges. The full study protocol and detailed statistics are provided in Appendix §H.2. We further show that the gains of our pipeline hold across judge models, with the experiments presented in Appendix §H.1.

7 Conclusion

In this work, we push open multimodal code intelligence forward by jointly contributing and fully releasing data, pipeline, and models. PrismaForge treats multimodal code data under one lens with rubrics synthesized from coarse to fine and decomposed into code-level and visual-level criteria, a signal that both generalizes and localizes what went wrong. We then curate and release PrismaCoder-2M, the most broadly covering open dataset for multimodal code intelligence, which enables principled selection and stable reinforcement learning on visual-coding tasks. On this data we build PrismaCoder-9B/35B, which lead the open state of the art by a wide margin and rival proprietary systems. Extensive analyses across base models and task recipes show that the data and the pipeline generalize beyond our release, and we share these insights together with the resources as a foundation for the community to democratize visual-coding capabilities.

Ethics Statement

All seed data we build on come from open-source releases. The released artifacts are program code and its rendered outputs, charts, web interfaces, documents, and related media, expanded with frontier models under their providers' terms of service. The corpus is not built from personal data, and every sample passes an execution or rendering gate. The small-scale human evaluation was conducted with the consent of all participants, and no personally identifiable information was retained. Rubric judging and reward modeling operate on code and renders only. The models we release generate visual code and inherit the general dual-use risks of code generation, such as producing deceptive interfaces; we release the dataset, models, and pipeline for research, with provenance documented so that downstream filtering remains possible. The authors declare no conflicts of interest.

AI Use Statement

In this work, we used generative AI tools for refining the writing, including grammar and spelling corrections. Additionally, we used generative AI tools for coding assistance and making case studies. The authors reviewed and take responsibility for the research design, experiments, analysis, claims, and final presentation.

Reproducibility Statement

Our code and data are released at this anonymous repository. The paper specifies how PrismaForge synthesizes rubrics, curates data, and computes the reinforcement-learning reward. Training, evaluation, and computation details are provided in the appendix.

References

Anthropic. Introducing Claude Opus 4.6. https://www.anthropic.com/news/claude-opus-4-6, February 2026a. Accessed: 2026-06-29.

Anthropic. Introducing Claude Sonnet 5. https://www.anthropic.com/news/claude-sonnet-5, June 2026b. Accessed: 2026-09-23.

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.

Qiaosheng Chen, Yang Liu, Lei Li, Kai Chen, Qipeng Guo, Gong Cheng, and Fei Yuan. Interactscience: Programmatic and visually-grounded evaluation of interactive scientific demonstration code generation, 2025a. URL https://arxiv.org/abs/2510.09724.

Yamei Chen, Haoquan Zhang, Yangyi Huang, Zeju Qiu, Kaipeng Zhang, Yandong Wen, and Weiyang Liu. Symbolic graphics programming with large language models, 2025b. URL https://arxiv.org/abs/2509.05208.

Timur Galimzyanov, Sergey Titov, Yaroslav Golubev, and Egor Bogomolov. Drawing pandas: A benchmark for llms in generating plotting code. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pp. 503–507, 2025. doi: 10.1109/MSR66628.2025.00083.

Gemini Team. Gemini 3 pro model card, November 2025. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/.

Gemini Team. Gemini 3.5 flash model card, May 2026. URL https://deepmind.google/models/model-cards/gemini-3-5-flash/.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Bohua Chen, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. Webcode2m: A real-world dataset for code generation from webpage designs. In *THE WEB CONFERENCE 2025*, 2025. URL https://openreview.net/forum?id=aeP5nmlw5B.

Jiawei Guo, Tianyu Zheng, Yizhi Li, Yuelin Bai, Bo Li, Yubo Wang, King Zhu, Graham Neubig, Wenhu Chen, and Xiang Yue. MAmmoTH-VL: Eliciting multimodal reasoning with instruction tuning at scale. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13869–13920, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.680. URL https://aclanthology.org/2025.acl-long.680/.

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

Lingjie Jiang, Shaohan Huang, Xun Wu, Yixia Li, Guanhua Chen, Dongdong Zhang, and Furu Wei. Viscodex: Unified multimodal code generation via merging vision and coding models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hUXzPauNEM.

Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R. Lyu, and Xiangyu Yue. Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. *CoRR*, abs/2507.22827, 2025. doi: 10.48550/ARXIV.2507.22827. URL https://doi.org/10.48550/arXiv.2507.22827.

S. Kim, J. Chen, T. Cheng, A. Gindulyte, J. He, S. He, Q. Li, B. A. Shoemaker, P. A. Thiessen, B. Yu, L. Zaslavsky, J. Zhang, and E. E. Bolton. Pubchem 2025 update. *Nucleic Acids Research*, 53(D1): D1516–D1525, Jan 2025. doi: 10.1093/nar/gkae1059.

Max Ku, Cheuk Hei Chong, Jonathan Leung, Krish Shah, Alvin Yu, and Wenhu Chen. TheoremExplainAgent: Towards video-based multimodal explanations for LLM theorem understanding. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6663–6684, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.332. URL https://aclanthology.org/2025.acl-long.332/.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.

Jinke Li, Jiarui Yu, Chenxing Wei, Hande Dong, Qiang Lin, Liangjing Yang, Zhicai Wang, and Yanbin Hao. UniSVG: A unified dataset for vector graphic understanding and generation with multimodal large language models. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 13156–13163, 2025.

Kaixin Li, Yuchen Tian, Qisheng Hu, Ziyang Luo, Zhiyong Huang, and Jing Ma. MMCode: Benchmarking multimodal large language models for code generation with visually rich programming problems. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 736–783, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.42. URL https://aclanthology.org/2024.findings-emnlp.42/.

Junteng Liu, Weihao Zeng, Xiwen Zhang, Yijun Wang, Zifei Shan, and Junxian He. On the perception bottleneck of vlms for chart understanding, 2025. URL https://arxiv.org/abs/2503.18435.

Ziyu Liu, Shengyuan Ding, Xinyu Fang, Xuanlang Dai, Penghui Yang, Jianze Liang, Jiaqi Wang, Kai Chen, Dahua Lin, and Yuhang Zang. Visual-erm: Reward modeling for visual equivalence. *arXiv preprint arXiv:2603.13224*, 2026.

Yuxuan Luo, Ryan Yuan, Junwen Chen, Haonan Cai, Ziyi Yue, Yuwei Yang, Fatima Zohra Daha, Ji Li, and Zhouhui Lian. Mmmg: A massive, multidisciplinary, multi-tier generation benchmark for text-to-image reasoning. *Advances in Neural Information Processing Systems*, 38, 2025a.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 27168–27188, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/72eba29737f9c3a5a4ce8cdb7b667145-Paper-Conference.pdf.

Ziyang Luo, Kaixin Li, Hongzhan Lin, Yuchen Tian, Mohan Kankanhalli, and Jing Ma. Tree-of-evolution: Tree-structured instruction evolution for code generation in large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 297–316, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.14. URL https://aclanthology.org/2025.acl-long.14/.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing moe reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 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, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Minheng Ni, Zhengyuan Yang, Yaowen Zhang, Linjie Li, Chung-Ching Lin, Kevin Lin, Zhendong Wang, Xiaofei Wang, Shujie Liu, Lei Zhang, et al. Proimage-bench: Rubric-based evaluation for professional image generation. *arXiv preprint arXiv:2512.12220*, 2025a.

Yuansheng Ni, Ping Nie, Kai Zou, Xiang Yue, and Wenhu Chen. Viscoder: Fine-tuning llms for executable python visualization code generation, 2025b. URL https://arxiv.org/abs/2506.03930.

Yuansheng Ni, Songcheng Cai, Xiangchao Chen, Jiarong Liang, Zhiheng Lyu, Jiaqi Deng, Kai Zou, Ping Nie, Fei Yuan, Xiang Yue, and Wenhu Chen. Viscoder2: Building multi-language visualization coding agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=4zoMnmZzh4.

Kunato Nishina and Yusuke Matsui. Svgeditbench: A benchmark dataset for quantitative assessment of llm's svg editing capabilities, 2024. URL https://arxiv.org/abs/2404.13710.

OpenAI. Introducing GPT-4.1 in the API. https://openai.com/index/gpt-4-1/, April 2025. Accessed: 2026-09-22.

OpenAI. GPT-5.5 system card. https://openai.com/index/gpt-5-5-system-card/, April 2026. Accessed: 2026-09-22.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Z. Xiao, Katherine M. Collins, Joshua B. Tenenbaum, Adrian Weller, Michael J. Black, and Bernhard Schölkopf. Can large language models understand symbolic graphics programs? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Yk87CwhBDx.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3956–3974, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.199. URL https://aclanthology.org/2025.naacl-long.199/.

Haoyu Sun, Huichen Will Wang, Jiawei Gu, Linjie Li, and Yu Cheng. Fullfront: Benchmarking mllms across the full front-end engineering workflow, 2025a. URL https://arxiv.org/abs/2505.17399.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*, 2024.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. OS-genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5555–5579, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.277. URL https://aclanthology.org/2025.acl-long.277/.

Qiushi Sun, Jingyang Gong, Lei Li, Qipeng Guo, and Fei Yuan. CodeEvo: Interaction-driven synthesis of code-centric data through hybrid and iterative feedback. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9668–9687, San Diego, California, United States, July 2026a. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.438/.

Qiushi Sun, Jingyang Gong, Yang Liu, Qiaosheng Chen, Lei Li, Kai Chen, Qipeng Guo, Ben Kao, and Fei Yuan. Januscoder: Towards a foundational visual-programmatic interface for code intelligence. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=N4BB09TXad.

Hanbin Wang, Xiaoxuan Zhou, Zhipeng Xu, Keyuan Cheng, Yuxin Zuo, Kai Tian, Jingwei Song, Junting Lu, Wenhui Hu, and Xueyang Liu. Code-vision: Evaluating multimodal llms logic understanding and code generation capabilities, 2025. URL https://arxiv.org/abs/2502.11829.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. Charxiv: Charting gaps in realistic chart understanding in multimodal LLMs. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=cy8mq7QYae.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: empowering code generation with oss-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 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, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.164. URL https://aclanthology.org/2025.findings-naacl.164/.

Renqiu Xia, Bo Zhang, Hancheng Ye, Xiangchao Yan, Qi Liu, Hongbin Zhou, Zijun Chen, Peng Ye, Min Dou, Botian Shi, Junchi Yan, and Yu Qiao. Chartx & chartvlm: A versatile benchmark and foundation model for complicated chart reasoning, 2025. URL https://arxiv.org/abs/2402.12185.

Jingyu Xiao, Ming Wang, Man Ho Lam, Yuxuan Wan, Junliang Liu, Yintong Huo, and Michael R. Lyu. Designbench: A comprehensive benchmark for mllm-based front-end code generation, 2025. URL https://arxiv.org/abs/2506.06251.

Fangzhi Xu, Zhiyong Wu, Qiushi Sun, Siyu Ren, Fei Yuan, Shuai Yuan, Qika Lin, Yu Qiao, and Jun Liu. Symbol-LLM: Towards foundational symbol-centric interface for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13091–13116, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.707. URL https://aclanthology.org/2024.acl-long.707/.

Fangzhi Xu, Qiushi Sun, Kanzhi Cheng, Jun Liu, Yu Qiao, and Zhiyong Wu. Interactive evolution: A neural-symbolic self-training framework for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12975–12993, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.635. URL https://aclanthology.org/2025.acl-long.635/.

Zhangchen Xu, Yang Liu, Yueqin Yin, Mingyuan Zhou, and Radha Poovendran. KodCode: A diverse, challenging, and verifiable synthetic dataset for coding. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6980–7008, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.365. URL https://aclanthology.org/2025.findings-acl.365/.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran XU, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. Chartmimic: Evaluating LMM's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=sGpCzsfd1K.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Jiaxu Zhang, Liao Wang, Gang Yu, Xinjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. *arXiv preprint arxiv:2504.06263*, 2025b.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, Zhiyuan Liu, Xiaodong Shi, and Maosong Sun. MatPlotAgent: Method and evaluation for LLM-based agentic scientific data visualization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11789–11804, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.701. URL https://aclanthology.org/2024.findings-acl.701/.

Seonghyeon Ye, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, Seungone Kim, Yongrae Jo, James Thorne, Juho Kim, and Minjoon Seo. Flask: Fine-grained language model evaluation based on alignment skill sets. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 55361–55414, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/f41b4a6b202adcd8e150a9d4f124d8f6-Paper-Conference.pdf.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal LLMs. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=hFVpqkRRH1.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7443–7464, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.411. URL https://aclanthology.org/2023.acl-long.411/.

Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Shihui Hu, Dengpeng Wu, Guanhua Huang, Kejiao Li, Qi Yi, Ruibin Xiong, Haotian Zhu, Yuanxing Zhang, Yuhao Jiang, Yue Zhang, Zenan Xu, Bohui Zhai, Guoxiang He, Hebin Li, Jie Zhao, Le Zhang, Lingyun Tan, Pengyu Guo, Xianshu Pang, Yang Ruan, Zhifeng Zhang, Zhonghu Wang, Ziyan Xu, Zuopu Yin, Wiggin Zhou, Chayse Zhou, and Fengzong Lian. Artifactsbench: Bridging the visual-interactive gap in llm code generation evaluation, 2025. URL https://arxiv.org/abs/2507.04952.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2471–2484, Singapore, December 2023. Associa-

tion for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.151. URL https://aclanthology.org/2023.emnlp-main.151/.

Fengji Zhang, Linquan Wu, Huiyu Bai, Guancheng Lin, Xiao Li, Xiao Yu, Yue Wang, Bei Chen, and Jacky Keung. Humaneval-v: Benchmarking high-level visual reasoning with complex diagrams in coding tasks. *arXiv preprint arXiv:2410.12381*, 2024a.

Liang Zhang, Anwen Hu, Haiyang Xu, Ming Yan, Yichen Xu, Qin Jin, Ji Zhang, and Fei Huang. TinyChart: Efficient chart understanding with program-of-thoughts learning and visual token merging. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1882–1898, Miami, Florida, USA, November 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.112. URL https://aclanthology.org/2024.emnlp-main.112/.

Zhehao Zhang, Weicheng Ma, and Soroush Vosoughi. Is GPT-4V (ision) all you need for automating academic data visualization? exploring vision-language models' capability in reproducing academic charts. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8271–8288, Miami, Florida, USA, November 2024c. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.485. URL https://aclanthology.org/2024.findings-emnlp.485/.

Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. Vincicoder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning, 2025a. URL https://arxiv.org/abs/2511.00391.

Xuanle Zhao, Xuexin Liu, Yang Haoyue, Xianzhen Luo, Fanhu Zeng, Jianling Li, Qi Shi, and Chi Chen. ChartEdit: How far are MLLMs from automating chart analysis? evaluating MLLMs' capability via chart editing. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3616–3630, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.185. URL https://aclanthology.org/2025.findings-acl.185/.

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, Vienna, Austria, July 2025c. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.363. URL https://aclanthology.org/2025.acl-long.363/.

Xuanle Zhao, Qiushi Sun, Jingyu Xiao, Xuexin Liu, Haoyue Yang, Qiaosheng Chen, Xianzhen Luo, Jing Huang, Yufeng Zhong, Lei Chen, Shuai Fu, Zhenlin Wei, Jinhe Bi, Lei Jiang, Haibo Qiu, Siqi Yang, Peng Shi, Jian Hu, and Zhixiong Zeng. Beyond NL2code: A structured survey of multimodal code intelligence. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=pn2sdu3Vrf.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Yuhao Zheng, Li'an Zhong, Yi Wang, Rui Dai, Kaikui Liu, Xiangxiang Chu, Linyuan Lv, Philip Torr, and Kevin Qinghong Lin. Code2world: A gui world model via renderable code generation. *arXiv preprint arXiv:2602.09856*, 2026.

Hongda Zhu, Yiwen Zhang, Bing Zhao, Jingzhe Ding, Siyao Liu, Tong Liu, Dandan Wang, Yanan Liu, and Zhaojian Li. Frontendbench: A benchmark for evaluating llms on front-end development via automatic evaluation, 2025. URL https://arxiv.org/abs/2506.13832.

Terry Yue Zhuo, Xiaolong Jin, Hange Liu, Juyong Jiang, Tianyang Liu, Chen Gong, Bhupesh Bishnoi, Vaisakhi Mishra, Marek Suppa, Noah Ziems, Saiteja Utpala, Ming Xu, Guangyu Song, Kaixin Li, Yuhan Cao, Bo Liu, Zheng Liu, Sabina Abdurakhmanova, Wenhao Yu, Mengzhao Jia, Jihan Yao,

Kenneth Hamilton, Kumar Shridhar, Minh Chien Vu, Dingmin Wang, Jiawei Liu, Zijian Wang, Qian Liu, Binyuan Hui, Meg Risdal, Ahsen Khaliq, Atin Sood, Zhenchang Xing, Wasi Uddin Ahmad, John Grundy, David Lo, Banghua Zhu, Xiaoning Du, Torsten Scholak, and Leandro von Werra. Bigcodearena: Unveiling more reliable human preferences in code generation via execution, 2025. URL https://arxiv.org/abs/2510.08697.

A Data Toolkit Details

A.1 AST-Based Decomposition

Plotting scripts collected from GitHub and Manim scene files are often long, and one file may draw several figures. For such data, we parse each script into an abstract syntax tree without executing it and split it into self-contained units. A unit is the code that produces one figure in a plotting script, or one scene in a Manim file (a class inheriting from `Scene`). Each unit keeps the imports and statements it depends on, drops project-private helpers that exist only in the source repository, and carries its embedded text and a short excerpt as features. The units are stored as structured entries that the synthesis pipeline and the gatekeeper of Section §4.2 read. All of these processing pipelines will be made public.

A.2 Per-Format Gatekeepers

Every candidate runs in a sandbox matched to its language before it is judged. Python plotting code executes headlessly and must save a figure, and HTML pages render in a headless browser to a screenshot. SVG, LaTeX, Mermaid, Asymptote, LilyPond, and Vega-Lite sources go through their standard renderers or compilers, Manim scenes render to their frames, and chemical-structure code must parse into a valid molecule. A candidate passes when the run finishes within a fixed time limit and produces its artifact, and it fails on an error, a timeout, or an empty artifact. The same gatekeeper serves data gating (§4.2) and reinforcement learning (§3.2), where a failed rollout receives reward 0.

A.3 Decontamination

Part of PrismaCoder-2M is synthesized by frontier models that may have seen public benchmarks, so we decontaminate the released data against the test sets of every benchmark in §5.2. A training sample is removed when its instruction shares a long n-gram with a test instruction.

B Data Sources

Table 4: Data sources and routing statistics.

Data Source	# Seed Samples	# Regenerated Samples	# Synthesized Samples
MCD dataset	598,000	307,184	273,199
GitHub repositories	417,000	297,942	188,630
WebCode2M (Generate)	243,100	182,300	215,663
WebCode2M (Edit)	264,541	167,810	190,882
JanusCoder-800K	109,826	80,000	29,000
WebSight	1,922,671	40,000	12,000
ProImageBench	654	600	80
MMMG	4456	2400	975
UniSVG	528,591	65,100	51,100
OmniSVG	255,412	37,000	10,000
arXiv papers	67,000	60,155	21,400
PubChem	60,000	60,000	59,000
Manim video code	41,707	25,440	16,300

Table 4 provides overall statistics on the samples used from each data source, as well as the number of samples processed at each stage of the gating procedure. For datasets that are higher in quality and more closely aligned with our target tasks, more than half of the original responses can be retained. In contrast, broader and less task-specific data sources, such as GitHub and arXiv papers, require more extensive processing, with a larger fraction of samples needing to be regenerated.

C Rubric Synthesis Prompts and Examples

Web UI reconstruction must be evaluated from two complementary perspectives: whether the generated HTML/CSS implements the required content and structure, and whether its rendered output reproduces the intended visual appearance. A single holistic score often conflates these two

aspects and provides little evidence about the source of an error. We therefore synthesize two sets of sample-specific binary rubrics from the task instruction and reference screenshot. Code rubrics target properties that can be verified directly from the implementation, whereas vision rubrics target properties observable in the rendered page. Figures 7 and 8 show the corresponding prompt templates. Both prompts organize checks into shared quality criteria and sample-dependent requirements, and assign each check a weight according to its importance. The difficulty level assigned to the instruction enters both prompts as an item budget, so harder samples receive more criteria (§3.1). The templates are abridged to remove repetitive wording.

Code Rubric Prompt (Web UI Generation)

You are an expert evaluator of Web UI generation. Your task is to generate a sample-specific binary rubric for evaluating the generated HTML/CSS from code evidence.

SAMPLE INPUTS

1. Problem statement: `{prompt}`
2. Input screenshot: [the input screenshot is attached to this message]
3. Difficulty level: `{difficulty}`, with a budget of `{min_items}`–`{max_items}` rubric items in total

TASK

Generate code rubrics that can later be judged from the generated HTML/CSS code alone, while remaining grounded in the problem statement and screenshot. Stay within the item budget and spend it on the properties that matter most for this sample. Every question must be decidable from the code alone; do not ask about the rendered appearance. Each item contains (1) a yes/no `question`, where 1 is pass and 0 is fail, and (2) a `weight` in $[0, 1]$: 0.8–1.0 for critical, 0.4–0.7 for important, and 0.1–0.3 for minor requirements.

RUBRIC DIMENSIONS

Shared rubrics (recur across the domain)

1. **Screenshot faithfulness.** Check whether the code captures the major visible regions and their relationships.
2. **Render safety and structural integrity.** Check whether the HTML/CSS is complete and can render without structural or layout-breaking errors.
3. **Placeholder image handling.** Check whether placeholders are present, constrained, and placed only where the reference requires them.

Task-specific rubrics (unique to this instruction)

1. **Layout mapping and section structure.** Check the order, nesting, alignment, and spatial organization of page sections.
2. **Content element coverage.** Check whether important text, controls, repeated items, and media regions are represented.
3. **Self-contained implementation.** Check that the page uses inline CSS and no unpermitted external dependencies or assets.

OUTPUT FORMAT

Return only valid JSON following this structure:

```
{
  "shared_rubrics": {
    "screenshot_faithfulness": [ {"question": "...", "weight": ...}, ... ], ...
  },
  "task_specific_rubrics": { "layout_mapping_section_structure": [...], ... }
}
```

Figure 7: Prompt template for synthesizing code rubrics in Web UI generation.

The complementary roles of the two rubric views are illustrated by the representative reconstruction in Fig. 9. Code rubrics verify that the expected elements and implementation constraints are present, whereas vision rubrics reveal rendering failures that may remain hidden in otherwise valid HTML/CSS. In this example, the synthesized page preserves the reference's overall content, colors, and organization, yet contains a salient local layout failure. It receives a perfect code-rubric score of 1.0, indicating that the implementation contains the expected structure and content. It also obtains a holistic LLM judgment score of 78, outperforming approximately 78.6% of all samples. Nevertheless, our sample-specific vision rubrics assign an aggregate score of only 0.182. They identify that the footer is incorrectly positioned in the middle of the page, where it overlaps and clips the third service button and disrupts the intended vertical hierarchy. They also capture that the synthesized buttons have square rather than rounded corners. We next examine the individual judgments that produce this discrepancy.

Vision Rubric Prompt (Web UI Generation)

You are an expert evaluator of Web UI generation. Your task is to generate a sample-specific binary rubric for evaluating the generated webpage from visual evidence in its rendered screenshot.

SAMPLE INPUTS

1. Problem statement: `{prompt}`
2. Input screenshot: [the input screenshot is attached to this message]
3. Difficulty level: `{difficulty}`, with a budget of `{min_items}`–`{max_items}` rubric items in total

TASK

Generate vision rubrics that can later be judged from the rendered screenshot, while remaining grounded in the problem statement and input screenshot. Stay within the item budget and spend it on the properties that matter most for this sample. Every question must be decidable from the rendered screenshot alone; do not ask about the source code. Each item contains (1) a yes/no `question`, where 1 is pass and 0 is fail, and (2) a `weight` in [0, 1]: 0.8–1.0 for critical, 0.4–0.7 for important, and 0.1–0.3 for minor requirements.

RUBRIC DIMENSIONS

Shared rubrics (recur across the domain)

1. **Screenshot faithfulness.** Check whether the output preserves the reference's major regions, hierarchy, and visual relationships.
2. **Visual integrity.** Check for overlap, clipping, collapse, distortion, or other visible rendering failures.
3. **Text legibility.** Check whether visible text remains readable and unobscured.

Task-specific rubrics (unique to this instruction)

1. **Layout structure and alignment.** Check the placement, spacing, and alignment of the major sections and elements.
2. **Content element coverage.** Check whether important visible text, controls, repeated items, and media regions appear.
3. **Placeholder image normality.** Check whether placeholders have the expected count, size, and placement without causing visible artifacts.

OUTPUT FORMAT

Return only valid JSON following this structure:

```
{
  "shared_rubrics": {
    "screenshot_faithfulness": [ {"question": "...", "weight": ...}, ... ], ...
  },
  "task_specific_rubrics": { "layout_structure_alignment": [...], ... }
}
```

Figure 8: Prompt template for synthesizing vision rubrics in Web UI generation.



Reference **Synthesized**

Figure 9: A representative reconstruction example. Although the synthesized page preserves the reference's overall appearance, its footer is misplaced across the Services section.

Rubric-level breakdown. Table 5 reports the individual checks behind these aggregate scores. The code evaluator passes all five checks because the source contains the expected colors, stacking mechanism, centering rule, footer styling, and copyright text. In contrast, the rendered output passes only the horizontal-centering check; the remaining vision rubrics expose the misplaced footer, clipped service button, and incorrect button corners.

Rubric	Weight	Pass	Judgment evidence
Code rubrics (all pass; weighted score 1.0)			
The three Product blocks use a teal-like background.	0.8	✓	The buttons explicitly use `background-color: #008080`.
The three Product items are stacked vertically.	0.8	✓	The shared button class uses `display: block`.
The "Technology Company" heading is horizontally centered.	0.8	✓	The header uses `text-align: center`.
The footer containing the copyright text has a dark background.	0.7	✓	The footer explicitly uses `background-color: #333`.
The footer contains the required copyright statement.	0.9	✓	The exact statement is present, with `©` encoding the copyright symbol.
Vision rubrics (one pass; weighted score $0.8/(1.0+0.8+0.9+0.9+0.8) = 0.182$)			
The page preserves the centered vertical hierarchy from the main heading through the footer.	1.0	✗	The footer overlaps the Services section instead of following it at the bottom.
All primary content is horizontally centered without unintended horizontal scrolling.	0.8	✓	The content is centered and no horizontal scrollbar is visible.
The teal and purple buttons form vertically stacked blocks with rounded corners.	0.9	✗	The buttons are stacked, but their corners are sharp rather than rounded.
The footer is a dark, full-width bar at the bottom with centered white text.	0.9	✗	Although its color and width are correct, the footer is placed over the last service item rather than at the bottom.
Exactly three teal Product buttons and three purple Service buttons are visible.	0.8	✗	The misplaced footer obscures and clips the third Service button.

Table 5: Code and vision rubrics with their judgments for the example in Fig. 9.

Holistic judgment. For comparison, the holistic evaluator assigns a score of 78 and provides the following rationale:

> The rendered image matches the reference closely in fonts, colors, and general structure. However, there is a layout error where the footer is positioned too high, overlapping the "Service 3" button and leaving empty space at the bottom.

Thus, the holistic evaluator notices the dominant layout error but compresses it into a relatively high scalar score. The rubric-based evaluation both amplifies its impact according to sample-specific requirements and identifies the additional corner-shape and content-visibility failures. These localized but perceptually important errors are easily diluted in a single holistic similarity score, whereas our fine-grained rubrics explicitly expose and penalize each failure. The example therefore demonstrates that the two rubric views provide more diagnostic and structurally sensitive feedback than either code inspection or holistic visual judgment alone.

In our experiments, we use Gemini-3-Flash for rubrics. For common visualization tasks, such as Python plotting, the average cost of generating rubrics for a single sample is approximately $0.006 based on the official API pricing, while evaluating the generated code and visual artifacts costs approximately $0.01 per sample. For HTML-related tasks, the input cost is substantially higher because rich, complex websites typically involve much longer HTML code. In these cases, rubric generation costs approximately $0.01 per sample, while evaluation costs approximately $0.04.

D RL Reward Specification

This appendix specifies the reward computation procedure behind §3.2.

For each rollout response $c_i^{(k)}$, the procedure first extracts code from the response text. For Python, we extract the longest fenced code block. For HTML, we extract a single complete document spanning from the doctype declaration to the closing tag. For SVG, we extract either the code contained in a fenced block or the first complete `<svg>` span. A completion that does not contain valid code of the required type is assigned a reward of 0 without further evaluation.

A dispatcher then routes each rollout to exactly one task family, and the scorer registered for that family returns a reward in $[0, 1]$, where 1 denotes a fully satisfactory solution. Any condition that makes a completion unscoreable (including a missing or malformed answer, unavailable rubric or reference material, execution or rendering failure, or a judge failure that persists after retries) results in a reward of 0.

The basic reward computation for each task largely follows Eq. 2 and Eq. 3. Specifically, the response code and the rendered visual artifact are scored separately and then aggregated using a weighted average. We use different combinations of code-score and visual-score weights across task families. This is because for tasks such as Python plotting, correct data representation and placement are as important as visual integrity and presentation quality. In contrast, for educational demonstration artifacts, visual readiness and interactivity are often more important than the particular implementation used to produce them.

We found, however, that predefined rubrics alone may fail to capture certain forms of misalignment in a few tasks such as interface reproduction. Instructions for these tasks inherently allow some flexibility in content generation, particularly with respect to styling and layout organization. As a result, some errors or mismatches can only be assessed reliably after the generated page is viewed alongside the reference page. To address this limitation, we introduce a complementary holistic score for task families of this type. This score is combined with the rubric-based scores using a weight denoted by λ_h. In this way, the evaluation retains the consistency and structure of rubric-based scoring while also capturing qualitative discrepancies that become apparent only when the generated artifact is evaluated as a whole.

The reward logic and the weight combinations used for each task domain are summarized in Table 6.

For each sample, the rubric-based method requires one API call to score each modality. Because code and visual artifacts are evaluated separately, its API cost is approximately $2\times$ that of directly scoring the visual outputs alone. In our experiments, a full round of GRPO training involves approximately 808K judgment requests. Given the resources constraint, we were unable to conduct a head-to-head comparison between PRISMAFORGE and direct scoring over a full GRPO training run. Instead, in a more focused Python plotting experiment, rubric-based scoring yielded a 6.91% improvement in the overall ChartMimic score compared with direct scoring.

Table 6: RL reward specification by task family. Every reward lies in $[0, 1]$; any unscoreable completion earns 0.

Task family	Spec. → output	Reward Type	Weights Assignment
Chart generation	text → Python	Rubrics	$\lambda_c = \lambda_v = 0.5$
Chart reproduction	image → Python	Rubrics	$\lambda_c = \lambda_v = 0.5$
Webpage generation	text → HTML	Rubrics	$\lambda_c = 0.4, \lambda_v = 0.6$
HTML demonstration	plan → HTML	Rubrics	$\lambda_c = 0.4, \lambda_v = 0.6$
+ reference snapshots	plan + images → HTML	Rubrics	$\lambda_c = 0.4, \lambda_v = 0.6$
SVG generation	text → SVG	Rubrics	$\lambda_c = 0.4, \lambda_v = 0.6$
Interface reproduction	image → HTML	Rubrics+Holistic	$\lambda_c = 0.2, \lambda_v = 0.5, \lambda_h = 0.3$
SVG reproduction	image → SVG	Rubrics+Holistic	$\lambda_c = 0.2, \lambda_v = 0.5, \lambda_h = 0.3$
Interface editing	page + instr. → HTML	Rubrics+Holistic	$\lambda_c = 0.2, \lambda_v = 0.4, \lambda_h = 0.4$

E BENCHMARK DETAILS AND EVALUATION SETTINGS

Code-driven video and animation, and theorem and symbolic visualization, are released as training data only and are not part of the evaluation.

Vision-Centric Benchmarks. ChartMimic reproduces a chart from its image; its Direct split regenerates the chart and its Customized split edits an existing one, scored by low-level element extraction (text, layout, type, color) and a high-level judge score. WebCode2M, DesignBench, and Design2Code cover front-end generation from a screenshot: WebCode2M buckets pages by length and reports a judge score, visual similarity, CLIP, and TreeBLEU; DesignBench scores generation and editing with a judge score plus CLIP, MAE, SSIM, and CMS; Design2Code reports element-level block, text, position, and color matches, CLIP, and a judge score. InteractScience's visual lens (VQT) is a visually grounded qualitative test judged by a vision–language model. UniSVG's image-conditioned lens (ISVGen) reproduces an image as SVG. ChemDraw scores chemical-structure code by execution rate, molecule validity, exact match, and Tanimoto similarity.

Text-Centric Benchmarks. ArtifactsBench reports an overall score over ten artifact domains. PandasPlotBench reports the incorrect-code rate and mean and Good($\geq$75) scores under a visual and a task lens. InteractScience's programmatic lens (PFT) is a functional test over the generated demonstration. UniSVG's text-conditioned lens (TSVGen) generates SVG from text. VisPlotBench covers eight languages with the same metric structure as PandasPlotBench, judged by GPT-4.1 (OpenAI, 2025) under its designated protocol.

Evaluation Settings. Evaluation follows the benchmark's official protocol, including its designated judge model where one is specified, so that our numbers stay comparable with published results. Due to the availability of legacy models, ChartMimic's high-level score is judged by GPT-5.5 (OpenAI, 2026) instead of the original `GPT-4o`.

F Training Details

All experiments were conducted on four interconnected compute nodes, each equipped with 8 NVIDIA H200 GPUs, for a total of 32 GPUs. We used the Megatron backend in the ms-swift framework for distributed SFT and RL. Both stages performed full-parameter fine-tuning with FlashAttention and sequence parallelism enabled. Activation recomputation was used throughout training to reduce memory consumption.

For SFT, we initialized from Qwen3.5-9B/Qwen3.5-35B-A3B and trained the language model for 5 epochs. We used a maximum sequence length of 32,768 tokens and enabled sequence packing for improved token utilization. The global batch size was set to be 128. The learning rate peaked at 1×10^{-5}, with a warm-up fraction of 5%, and decayed to a minimum learning rate of 5×10^{-7}.

For RL, we initialized from the SFT checkpoint and performed one epoch of GRPO/GSPO. We implemented a rubric-based reward plugin following the methodology described in Section 3.2 and Appendix D. For the judge model L^J used to evaluate online rollouts, we selected Gemini-3-Flash to balance evaluation quality and cost. The global training batch size was 512 in terms of rollouts (equivalent to 64 samples per batch) and 8 candidate responses were generated per prompt at each rollout step. Rollout generation was performed with colocated vLLM inference using tensor parallelism of size 8 and a maximum model length of 36,000 tokens. Sampling used a temperature of 1.0 and top-p of 0.95.

The GRPO/GSPO learning rate was set to 3×10^{-6}. We used a KL regularization coefficient of $\beta = 0.04$, and asymmetric clipping thresholds of $\epsilon = 0.20$ and $\epsilon_{\text{high}} = 0.28$. Dynamic sampling and overlong-response filtering were enabled during rollout collection. Training was conducted in BF16 precision, with model and optimizer offloading enabled to reduce GPU memory usage.

G Full Per-Benchmark Results

Each benchmark reports several metrics; the main text shows a compact headline set per benchmark, and the complete metric sets are here, one table per benchmark, mirroring our evaluation workbook.

G.1 Visualization

ChartMimic. The complete results of the direct-1800 split of the ChartMimic benchmark is shown in Table 7. We can see that PrismaCoder shows strong performance against both open and proprietary models. PrismaCoder-9B achieves an overall score of 77.70, substantially outperforming other similarly sized open models (around 72–73) and even surpassing GPT-4o (74.20). Scaling to PrismaCoder-35B further improves the overall score to 82.05, with particularly strong execution rate (96.16), text fidelity (90.41), layout (94.00), and GPT-based similarity (80.69). Its overall performance is competitive with much larger proprietary models, exceeding Claude Sonnet 4.6 and Claude Opus 4.6 in this evaluation, while approaching Gemini 3 Flash. These results suggest that PrismaCoder is especially effective at faithfully reconstructing chart structure and visual details while maintaining reliable executable code.

PandasPlotBench. Table 8 illustrates a more detailed results of PandasPlotBench. On this benchmark, PrismaCoder demonstrates competitive visualization-code generation performance across both model scales. PrismaCoder-9B achieves a mean visual/task score of 60.54/77.94 with 14.29% incorrect code, comparing favorably with other models in a similar size range, while PrismaCoder-35B

Table 7: ChartMimic, full metrics for the Direct-1800 splits.

Model	Direct							
	Exec. Rate	Text	Layout	Type	Color	Avg. (L)	GPT (H)	Overall
Open models								
JanusCoder-7B	89.16	77.61	85.49	70.06	67.34	75.12	68.97	72.05
JanusCoder-8B	80.91	73.10	81.19	65.00	69.03	76.54	69.23	72.89
VinciCoder-7B	90.88	79.91	87.99	73.72	71.59	78.30	66.09	72.20
VinciCoder-8B	90.72	79.78	86.62	73.03	70.95	77.59	66.44	72.01
PRISMACODER-9B	95.50	88.78	92.77	74.87	69.88	81.57	73.82	77.70
PRISMACODER-35B	**96.16**	**90.41**	**94.00**	**76.28**	**73.01**	**83.42**	**80.69**	**82.05**
Proprietary								
GPT-4o	92.38	81.13	86.71	72.83	64.87	76.38	72.02	74.20
Gemini 3 Flash	95.61	90.49	92.46	78.66	78.86	85.12	83.04	84.08
Gemini 3.1 Pro	95.77	81.68	82.84	73.33	73.98	77.96	81.73	79.85
Claude Sonnet 4.6	97.11	91.89	94.17	68.53	66.35	80.23	80.73	80.48
Claude Opus 4.6	95.05	90.20	92.01	73.10	69.13	81.11	79.68	80.40

further reduces the incorrect-code rate to 9.29% and reaches 63.38/79.85, with 60.6% of examples attaining a visual score above 75. These gains are relatively modest compared with some strong baselines, which is unsurprising given the nature of the benchmark. Introduced in 2024, the benchmark contains 175 human-curated, largely Matplotlib-gallery-derived tasks and evaluates plotting from controlled natural-language descriptions, including synthetically shortened variants. In contrast, much of PrismaCoder's training data is curated toward more realistic and diverse visualization workflows. However, it is still a classic Python visualization benchmark and we included it for a more realistic understanding of our model capability.

Table 8: PandasPlotBench, full metrics under the Visual and Task lenses.

Model	Incorr. Code↓	Mean		Good ≥75	
		Vis.	Task	Vis.	Task
Qwen3-VL-8B-Instruct	16.00%	60.01	73.67	0.506	0.701
Qwen3-VL-30B-A3B-Instruct	11.43%	60.05	80.58	0.555	0.806
Qwen3.5-9B	24.57%	52.98	65.47	0.417	0.640
Qwen3.5-35B-A3B	15.43%	60.36	75.17	0.509	0.743
JanusCoderV-7B	23.43%	53.05	65.77	0.440	0.640
JanusCoderV-8B	26.30%	56.30	70.91	0.450	0.650
VinciCoder-7B	9.14%	25.37	51.57	0.137	0.314
VisCoder2-7B	20.00%	59.48	67.59	0.511	0.623
PrismaCoder-9B	10.29%	60.54	77.94	0.497	0.766
PrismaCoder-35B	9.29%	62.38	79.85	0.606	0.772
GPT-4o	8.00%	67.63	82.46	0.571	0.822
Gemini 3 Flash	3.43%	69.95	91.21	0.566	0.949
Gemini 3.1 Pro	1.71%	78.41	94.70	0.703	0.960
Claude Sonnet 4.6	3.43%	74.59	91.93	0.686	0.914
Claude Opus 4.6	1.71%	76.03	94.97	0.674	0.966

VisPlotBench. Table 9 averages the three metrics over the eight languages, and Table 11 lists the incorrect-code rate per language. PRISMACODER-35B has the lowest incorrect-code rate of every model in the table (28.86%) and the highest mean and good-plot scores under both lenses. The mean task score of PRISMACODER-35B is 62.47, against 23.95 for the best VinciCoder model, and PRISMACODER-9B leads both VinciCoder models on every column. Per language, the remaining failures concentrate on LilyPond, LaTeX, and Asymptote, where both models still break more programs than the specialized open baselines. On the web and diagram formats the picture reverses: PRISMACODER-35B has the lowest incorrect-code rate of every model on Vega-Lite and HTML and produces no broken program on SVG, and PRISMACODER-9B is the strongest open model on HTML.

Table 9: VisPlotBench, full metrics averaged over its eight languages.

Model	Incorr. Code↓	Mean		Good ≥75	
		Vis.	Task	Vis.	Task
VinciCoder-7B	35.90%	15.93	23.95	4.16	10.11
VinciCoder-8B	36.08%	8.84	12.55	1.50	2.80
PRISMACODER-9B	34.70%	36.40	53.99	23.12	52.62
PRISMACODER-35B	28.86%	45.16	62.47	32.55	61.49
GPT-4o	32.01%	38.19	56.01	24.75	52.75

Table 10: UniSVG (Li et al., 2025), full metrics for both lenses.

Model	ISVGen	TSVGen	Final
JanusCoderV-7B	61.60	66.43	61.23
JanusCoderV-8B	62.47	67.28	63.47
VinciCoder-7B	85.72	64.06	67.22
VinciCoder-8B	87.18	63.45	67.58
PRISMACODER-9B	69.64	68.49	65.31
PRISMACODER-35B	84.14	70.72	72.96
GPT-4o	70.98	72.80	69.60

Table 11: VisPlotBench, per-language incorrect-code rate.

Model	Avg. Incorr.↓	Python	Vega-Lite	LilyPond	Mermaid	SVG	LaTeX	Asymptote	HTML
Open models									
VinciCoder-7B	35.90%	10.20%	66.67%	100.00%	7.63%	1.54%	56.25%	21.74%	23.15%
VinciCoder-8B	36.08%	2.55%	57.36%	100.00%	4.58%	7.69%	58.93%	19.57%	37.96%
VisCoder2	33.26%	43.37%	16.28%	41.82%	25.95%	12.31%	61.61%	43.48%	21.30%
PRISMACODER-9B	34.70%	29.59%	13.18%	85.45%	12.21%	1.54%	66.96%	65.22%	19.44%
PRISMACODER-35B	28.86%	11.22%	3.88%	89.09%	9.92%	0.00%	70.54%	51.09%	11.11%
Proprietary									
GPT-4o	32.01%	22.45%	8.53%	40.00%	24.43%	0.00%	61.61%	75.00%	24.07%

UniSVG. Table 10 reports the image-conditioned lens (ISVGen), the text-conditioned lens (TSVGen), and the benchmark's final score. PRISMACODER-35B has the highest final score in the table (72.96), and both of our models lead every open baseline on TSVGen. VinciCoder's strength sits in ISVGen alone, the lens its training targets, and its text-conditioned scores are the lowest among the open models. PRISMACODER-35B comes within about three points of it on ISVGen and leads it by seven on TSVGen, so the final score favors the balanced profile.

ChemDraw. Table 12 adds exact match and two Tanimoto thresholds to the two columns of the main table. PRISMACODER-35B leads every column. All 128 of its outputs parse, 92.19% match the reference molecule exactly, and 95.31% reach a Tanimoto similarity of 1. PRISMACODER-9B is second on every column and above every proprietary model on exact match and Tanimoto similarity. Both are large gains over the Qwen3.5 bases, whose exact-match rates are 33.59% and 53.12%. The mean Tanimoto similarity is averaged over RDKit-valid outputs only, so the 91.39 of Gemini 3 Flash covers the 63% of its outputs that parse, whereas ours cover every sample.

Table 12: ChemDraw, full metrics on the 128-sample test set. Tani. (mean) averages over RDKit-valid outputs; the other columns count all samples.

Model	Valid.	Exact Match	Tani. (mean)	Tani. $= 1$	Tani. $\geq .85$
Open models					
Qwen3-VL-8B-Instruct	78.91%	25.00%	54.79	25.00%	25.00%
Qwen3-VL-30B-A3B-Instruct	82.03%	26.56%	54.55	27.34%	28.12%
Qwen3.5-9B	88.28%	33.59%	66.79	37.50%	39.84%
Qwen3.5-35B-A3B	92.97%	53.12%	79.86	57.81%	60.16%
PRISMACODER-9B	96.88%	83.59%	95.94	87.50%	89.06%
PRISMACODER-35B	100.00%	92.19%	97.56	95.31%	95.31%
Proprietary					
GPT-4o	92.19%	32.03%	57.80	33.59%	33.59%
Gemini 3 Flash	63.28%	52.34%	91.39	53.12%	53.91%
Claude Sonnet 4.6	88.28%	53.91%	85.43	61.72%	64.06%
Claude Opus 4.6	96.88%	52.34%	77.71	56.25%	57.81%
Kimi K2.5	91.41%	65.62%	91.20	75.00%	76.56%

G.2 WEBDEV

WebCode2M. Table 13 shows that PRISMACODER's advantage is consistent across webpages of different code lengths. PRISMACODER-9B obtains Visual scores of 90.06, 90.02, and 91.15 on the short, mid, and long subsets, respectively, while its LLM score remains similarly stable across the three groups. CLIP similarity decreases modestly with page length, from 90.29 to 87.32, but remains well above the prior open models throughout. The main exception is TreeBLEU: both PRISMACODER models score below JanusCoder and VinciCoder despite producing substantially more faithful renderings according to the visual metrics. This difference arises because TreeBLEU measures similarity to the reference HTML tree rather than the rendered webpage itself. JanusCoder and VinciCoder are trained on the original WebCode2M code, which naturally favors reproducing structures similar to the benchmark references. PRISMACODER instead uses regenerated code paired with webpages rendered from that code, rather than directly using the original WebCode2M source code. This regeneration is intentional. The original WebCode2M code comes from diverse real-world webpages and can differ substantially from the code distribution naturally produced by LLMs. Using regenerated code therefore better aligns the training targets with inference-time generations, while re-rendering the regenerated programs ensures exact consistency between the code and its visual target. Its generated webpages can consequently closely match the target appearance without following the particular HTML or DOM structure of the reference implementation. The similarly low TreeBLEU scores of the proprietary models further illustrate that reference-code similarity does not necessarily track rendering fidelity.

Table 13: WebCode2M, full metrics by code length.

	Short				Mid				Long				Average			
Model	LLM	Visual	CLIP	Tree-BLEU	LLM	Visual	CLIP	Tree-BLEU	LLM	Visual	CLIP	Tree-BLEU	LLM	Visual	CLIP	Tree-BLEU
Open models																
JanusCoderV-7B	40.10	72.39	76.11	29.64	43.06	74.08	76.77	37.43	39.61	72.24	75.11	**39.37**	40.92	72.90	76.00	**35.48**
JanusCoderV-8B	34.60	65.74	73.14	30.37	36.17	65.41	73.46	29.88	34.74	66.71	72.53	27.66	35.17	65.95	73.04	29.30
VinciCoder-7B	46.01	76.76	80.43	30.00	44.34	73.57	79.38	**38.11**	38.99	66.92	75.97	29.74	43.11	72.42	78.59	32.62
VinciCoder-8B	48.35	80.43	82.03	**31.39**	48.08	79.53	81.70	**39.48**	42.05	74.83	78.63	32.57	46.16	78.26	80.79	34.48
PRISMACODER-9B	**70.65**	**90.06**	**90.29**	17.23	**69.08**	**90.02**	**87.76**	18.65	**69.42**	**91.15**	**87.32**	17.23	**69.72**	**90.41**	**88.46**	17.70
PRISMACODER-35B	68.52	89.58	89.93	15.78	68.12	89.54	87.21	17.77	66.69	89.87	86.19	17.18	67.78	89.66	87.78	16.91
Proprietary																
GPT-4o	57.62	83.96	82.10	11.73	53.27	83.35	79.62	10.00	53.93	83.03	78.94	8.13	54.94	83.45	80.22	9.95
Gemini 3.1 Pro	71.67	89.23	88.52	16.77	70.83	89.78	87.58	16.56	70.40	90.87	86.82	14.44	70.97	89.96	87.64	15.92

DesignBench. The detailed results in Table 14 show that the generation gains are consistent across multiple visual metrics. PRISMACODER-9B reaches a Judge score of 69.95 and a CLIP score of 84.77, while reducing MAE to 42.07 and increasing SSIM to 0.6956. PRISMACODER-35B exhibits a nearly identical profile and achieves the lowest MAE and highest SSIM among the evaluated models. The agreement across Judge, CLIP, MAE, and SSIM indicates that the improvement is not tied to a particular evaluator, but is consistently reflected across multiple complementary measures of visual similarity. Editing shows a more nuanced pattern: both PRISMACODER models achieve an Editing-Judge score of about 8.9, substantially exceeding the prior open baselines and matching GPT-4o, while remaining slightly behind Gemini 3.1 Pro. In contrast, their CMS and image-based similarity metrics are less competitive with the strongest proprietary models. This difference can be partly attributed to the reference-dependent nature of these metrics. CMS measures the overlap between the code lines modified by the model and those modified in the reference implementation, so a correct edit may still receive a lower score when it is realized through different code locations. Similarly, MAE and SSIM compare the resulting rendering against a specific reference output and can penalize visually different but still valid implementations. The strong Editing-Judge scores therefore suggest that PRISMACODER reliably completes the requested modifications, even when its implementation and final rendering do not exactly match the particular reference solution.

Design2Code. Table 15 provides a finer-grained view of the visual gains on Design2Code. PRISMACODER-9B reaches 90.09 in Block, 95.95 in Text, 89.30 in Position, 90.00 in Color, and 92.37 in CLIP, yielding an average Visual score of 91.54. Relative to VinciCoder-8B, the largest gains appear in Position, Color, and Block, which improve by 15.40, 13.00, and 10.68 points, respectively, whereas the improvements in Text and CLIP are smaller. Under the Design2Code evaluation, these dimensions respectively capture spatial alignment of matched elements, perceptual color fidelity, and the coverage of visual blocks while penalizing missing or hallucinated elements. The results therefore

Table 14: DesignBench, full metrics for generation and editing.

	Generation				Editing			
Model	Judge	CLIP	MAE↓	SSIM	Judge	CMS	MAE↓	SSIM
Open models								
JanusCoderV-7B	40.37	71.34	77.05	0.5910	7.44	0.2234	30.11	0.8399
JanusCoderV-8B	34.30	68.38	83.64	0.5822	7.26	0.2346	28.57	0.8446
VinciCoder-7B	43.24	72.81	68.05	0.6137	1.40	0.1686	45.33	0.8119
VinciCoder-8B	48.09	76.79	61.86	0.6330	1.71	0.1514	41.83	0.8199
PRISMACODER-9B	**69.95**	**84.77**	42.07	0.6956	**8.93**	**0.2791**	**30.56**	0.8373
PRISMACODER-35B	68.98	84.56	**41.97**	**0.6972**	8.89	0.2601	30.67	**0.8398**
Proprietary								
GPT-4o	52.64	76.45	64.11	0.6259	8.89	0.3567	27.97	0.8439
Gemini 3.1 Pro	69.47	85.23	44.11	0.6914	9.57	0.3751	26.13	0.8548

indicate that PRISMACODER's advantage comes primarily from recovering a more complete set of webpage elements and placing them more accurately, together with improved color matching, rather than simply reproducing the visible text. PRISMACODER-35B exhibits a very similar metric profile, with only minor differences between the two model sizes across the evaluated dimensions.

Table 15: Design2Code, full metrics.

Model	Block	Text	Pos.	Color	CLIP	Avg	LLM Score
Open models							
JanusCoderV-7B	63.88	85.40	62.74	59.11	82.94	70.81	36.49
JanusCoderV-8B	59.11	81.72	61.45	57.63	83.13	68.61	34.53
VinciCoder-7B	71.01	87.30	68.04	71.28	85.58	76.64	42.30
VinciCoder-8B	79.41	91.79	73.90	77.00	87.37	81.89	45.72
PRISMACODER-9B	**90.09**	**95.95**	**89.30**	90.00	**92.37**	**91.54**	**70.51**
PRISMACODER-35B	89.45	95.47	89.14	**90.05**	91.98	91.22	67.55
Proprietary							
GPT-4o	89.50	96.91	83.67	82.27	88.80	88.23	58.48
Gemini 3.1 Pro	90.55	95.21	88.58	87.69	91.46	90.70	70.90

G.3 ARTIFACTS AND DEMOS

ArtifactsBench. Table 16 presents the overall and domain-level performance on ArtifactsBench. The results show that our models generalize well across the heterogeneous artifact types in ArtifactsBench. Rather than being concentrated in a single domain, PRISMACODER-9B maintains strong performance across all five categories and achieves the highest overall score among the open models, with particularly strong results on SVG generation and web applications. PRISMACODER-35B reaches a similar overall level but shows a different strength profile, with especially strong performance on game development and simulation. This contrast suggests that the two models are capable of handling different forms of front-end complexity, from visual and interface construction to artifacts that require richer interaction and application logic. More importantly, both models consistently outperform the other open baselines and the proprietary baseline at the overall level, despite ArtifactsBench spanning substantially different artifact structures. This cross-domain consistency indicates that their advantage is not tied to a specific artifact format, but reflects stronger general front-end code generation and the ability to turn complex specifications into executable artifacts.

InteractScience. Table 17 shows full metrics on InteractScience for both PFT and VQT lenses. The results highlight the complementary strengths of our models captured by the two lenses of InteractScience. PFT measures whether a model can correctly implement the interactive components specified in a long implementation plan, while VQT places greater emphasis on integrating visual understanding, executable code, and scientific reasoning. Our models perform particularly strongly under this setting. PRISMACODER-9B and PRISMACODER-35B both achieve substantially higher VQT scores than the other open models, while also maintaining competitive PFT performance. This gap is important because success on VQT cannot be obtained simply by producing syntactically valid front-end code. The generated science demos must also reflect the intended visual state and the scientific relationships encoded in the task. The results therefore point to a useful property of our models

Table 16: ArtifactsBench, overall and per-domain scores.

Model	Overall	Game Dev.	SVG Gen.	Web Apps	Simulation	Mgmt. Sys.
Open models						
JanusCoderV-7B	31.16	27.95	28.97	33.32	34.04	33.64
VinciCoder-7B	11.35	10.51	12.04	12.48	11.48	11.29
VinciCoder-8B	11.97	11.06	14.94	12.34	11.27	12.10
VisCoder2	17.51	16.49	20.43	18.46	18.13	15.98
PRISMACODER-9B	45.28	44.49	50.95	46.22	48.23	42.27
PRISMACODER-35B	46.01	54.33	45.00	44.51	58.53	37.09
Proprietary						
GPT-4o	33.54	36.96	39.54	39.27	35.73	35.83

whose capability is not limited to implementing isolated UI interactions, but extends to coordinating code, visual requirements, and scientific content when following a detailed implementation plan.

Table 17: InteractScience, full metrics for both lenses.

	PFT			VQT
Model	Overall	Avg.	Perfect	VLM
JanusCoderV-7B	17.73%	16.91%	4.20%	27.67
JanusCoderV-8B	17.60%	17.30%	4.20%	33.32
VinciCoder-7B	0.40%	0.00%	0.00%	0.00
VinciCoder-8B	0.40%	0.42%	0.00%	2.33
PRISMACODER-9B	24.80%	22.47%	7.69%	51.87
PRISMACODER-35B	28.00%	25.94%	6.99%	53.54
GPT-4o	31.07%	28.59%	10.49%	46.01

H EXTENDED ANALYSIS

H.1 BACKBONE GENERALIZATION

Figure 10 restates the Gemma arm of Fig. 5 as dumbbells, with the base and final score of every metric. The dumbbells express every metric on a percentage scale, so the change in the 0–10 DesignBench Edit score is plotted $\times 10$ while its score columns keep the native scale. Table 18 scores the VisPlotBench splits under two judges, the benchmark's designated GPT-4.1 (OpenAI, 2025) and the newer GPT-5.5 (OpenAI, 2026). The two agree on the direction of every training effect. The one disagreement is the HTML task score, where GPT-4.1 ranks the RL checkpoint first and GPT-5.5 the SFT one.

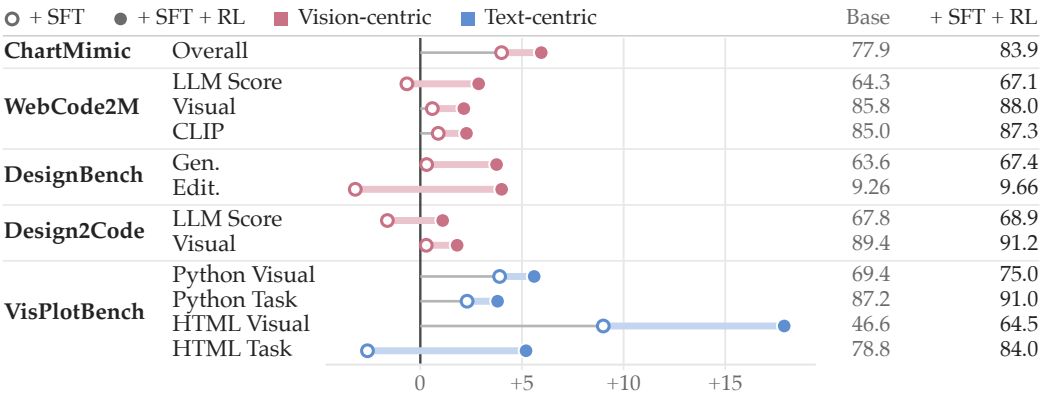


Figure 10: Per-metric gains of our SFT and RL pipeline on Gemma-4-31B-it, with the base and final scores.

Table 18: VisPlotBench scores of the Gemma arm under the designated judge and a newer judge of the same family. **Best** per column within a split.

		Judge: GPT-4.1			Judge: GPT-5.5		
Split	Model	Err%↓	Vis↑	Task↑	Err%↓	Vis↑	Task↑
HTML	Gemma-4 (base)	5.6	46.6	78.8	5.6	45.4	64.2
	+ SFT	**4.6**	55.6	76.2	**4.6**	53.0	**77.8**
	+ SFT + RL	**4.6**	**64.5**	**84.0**	**4.6**	**64.6**	75.4
Python	Gemma-4 (base)	**6.1**	69.4	87.2	**6.1**	66.8	85.0
	+ SFT	7.1	73.3	89.5	7.1	71.0	87.2
	+ SFT + RL	**6.1**	**75.0**	**91.0**	**6.1**	**72.9**	**87.6**

H.2 Cross-Family Adjudication and Human Evaluation

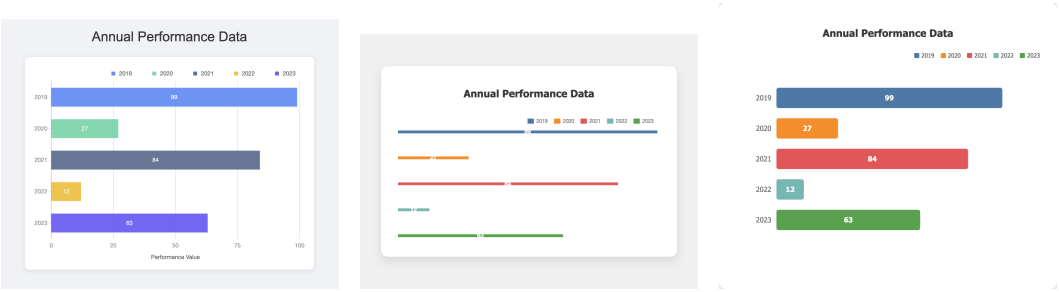
Table 19 lists the agreement statistics of the human study of §6.3. The 100 samples come from the SFT split of PrismaCoder-2M, sampled by difficulty level with Python and HTML artifacts in equal numbers across tasks. Twenty annotators each rated ten samples directly and ten others through the rubric, with disjoint sample sets per annotator, so each sample carries two human ratings under each mode. Each of the three judges, Claude Sonnet 5 (Anthropic, 2026b), Gemini 3.5 Flash (Gemini Team, 2026), and GPT-5.5 (OpenAI, 2026), scored every sample twice under each mode. The direct mode asks for a single holistic score, as in a typical pipeline (Sun et al., 2026b); the rubric mode scores through the PrismaForge rubric of §3.1. Krippendorff's α and Kendall's τ_b are computed between each judge's scores and the human ratings under the same mode. Across the three judges, α rises from 0.30–0.51 under direct scoring to 0.86–0.92 under the rubric, and τ_b from 0.28–0.44 to 0.64–0.67.

Table 19: Agreement of judge scores with human ratings on 100 samples. **Best** per judge.

Judge	Scoring	Krippendorff's α	Kendall's τ_b
Claude Sonnet 5	Direct	0.386	0.292
	Rubric-based	**0.889**	**0.670**
Gemini 3.5 Flash	Direct	0.303	0.282
	Rubric-based	**0.861**	**0.643**
GPT-5.5	Direct	0.514	0.442
	Rubric-based	**0.918**	**0.657**

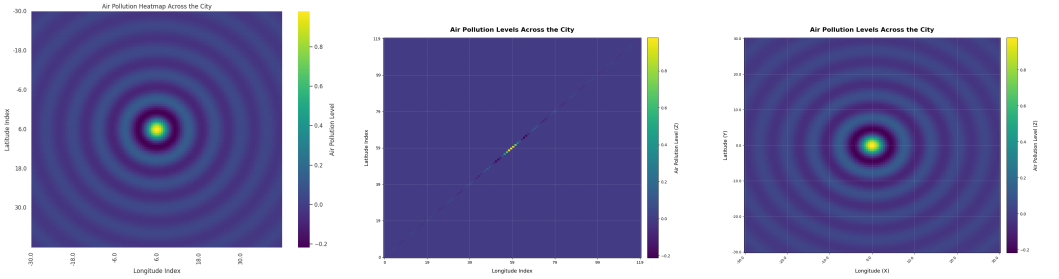
I Case Studies

Six qualitative cases drawn from VisPlotBench, one per format family. Three compare PrismaCoder-9B against its Qwen3.5-9B base (LaTeX, SVG, Mermaid); three compare PrismaCoder-35B against its SFT-only variant (Vega-Lite, HTML, Python), isolating what the rubric-based RL stage adds. In every case the middle panel executes and renders, and the failure is visual. A seventh case, chart-to-code on ChartMimic, follows PrismaCoder-9B across its training stages (Fig. 16) and sets the result beside an open and two proprietary references (Fig. 17). Four front-end cases close the appendix, two screenshot-to-page generations from Design2Code and two instruction-driven edits from DesignBench, each across the same training stages (Figs. 18 to 21).



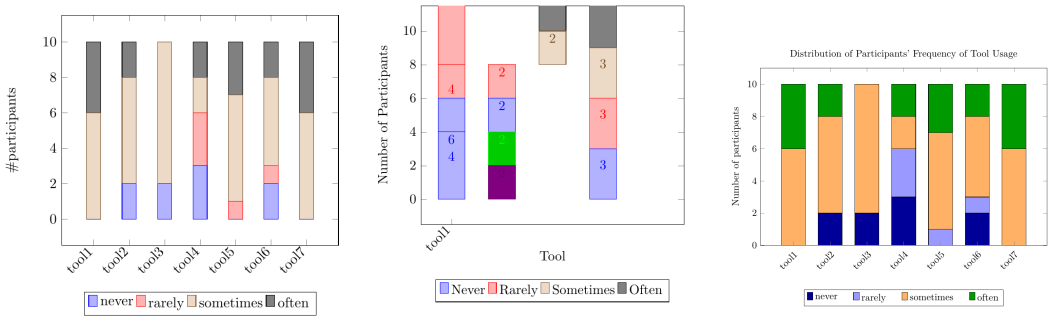
Gold PrismaCoder-35B w/o RL PRISMACODER-35B

Figure 11: HTML bar chart. The SFT-only model renders the bars as hairlines; the RL model recovers bar geometry and value labels.



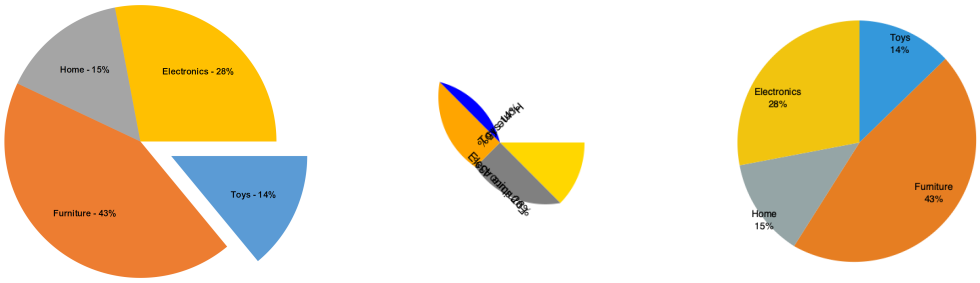
Gold PrismaCoder-35B w/o RL PRISMACODER-35B

Figure 12: Python heatmap. The SFT-only output runs and passes as code yet renders the radial field onto a diagonal; the RL model restores it.



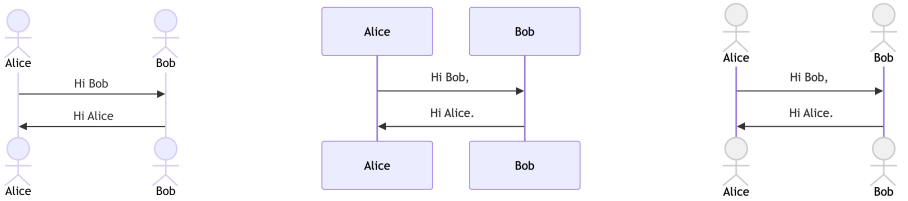
Gold Qwen3.5-9B PRISMACODER-9B

Figure 13: LaTeX stacked bars. The base model mis-stacks the categories and drops bars; PRISMACODER-9B reproduces the per-tool composition.



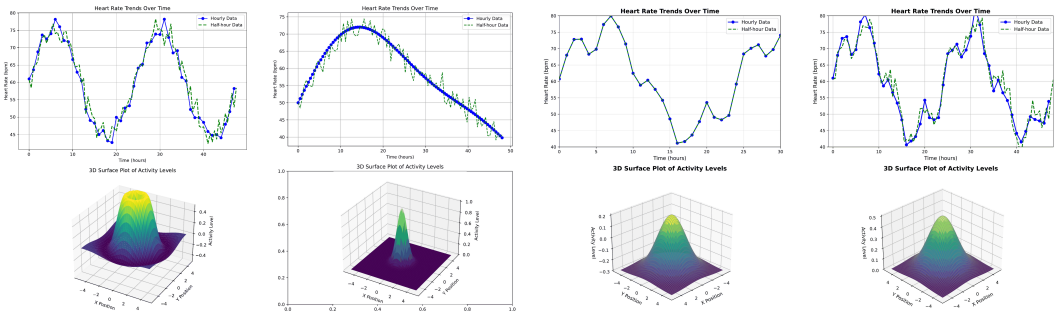
Gold Qwen3.5-9B PRISMACODER-9B

Figure 14: SVG pie chart. The base model's sectors collapse into overlapping wedges; PRISMACODER-9B closes the pie with correct shares and labels.



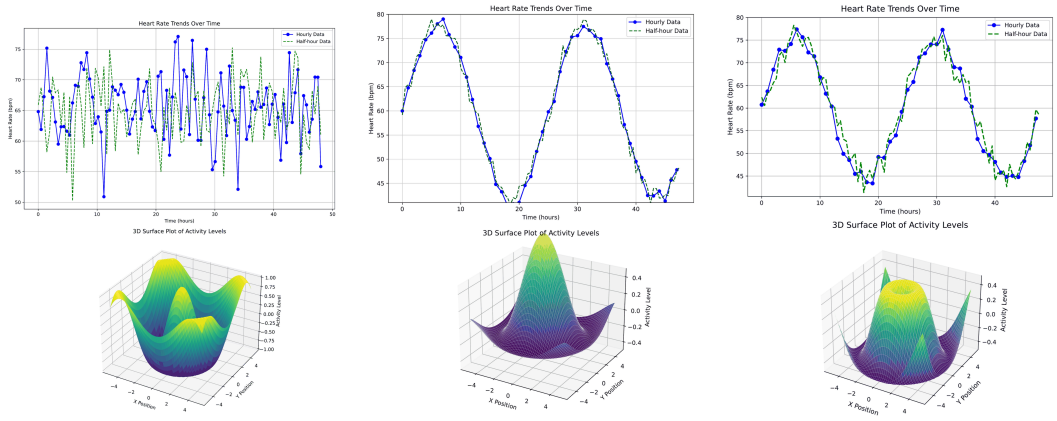
Gold Qwen3.5-9B PRISMACODER-9B

Figure 15: Mermaid sequence diagram. The base model draws participants as plain boxes; PRISMACODER-9B reproduces the actor figures and the message order.



Gold Qwen3.5-9B PRISMACODER-9B w/o RL PRISMACODER-9B

Figure 16: Chart-to-code on ChartMimic across training stages, where SFT recovers one of the line chart's two cycles and RL restores both.



VinciCoder-7B | Claude Opus 5 | Gemini 3.5 Flash

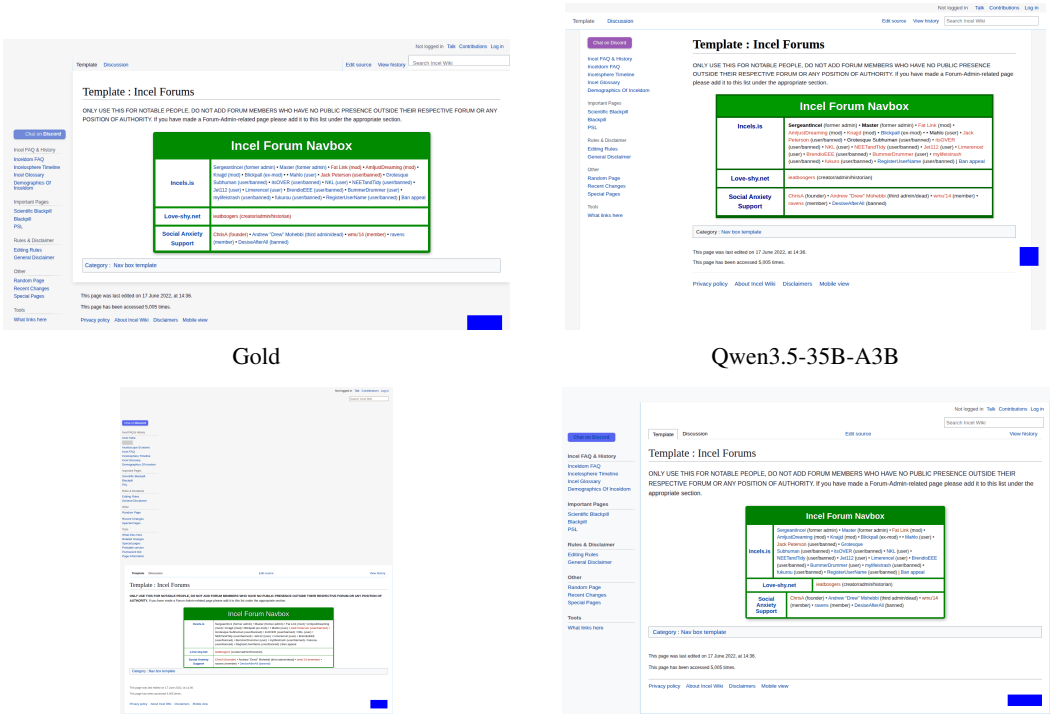
Figure 17: The same case for an open and two proprietary references, of which only the proprietary models reproduce the line chart.



Gold | Qwen3.5-9B

PRISMACODER-9B w/o RL | PRISMACODER-9B

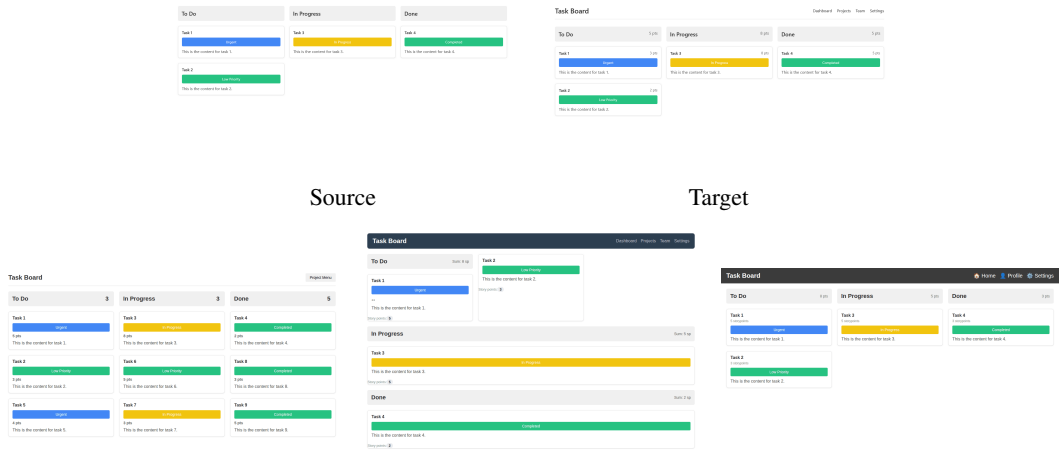
Figure 18: Front-end generation on Design2Code, where the base and SFT-only models break the title layout and PRISMACODER-9B reproduces the page.



Gold | Qwen3.5-35B-A3B

PRISMACODER-35B w/o RL | PRISMACODER-35B

Figure 19: Front-end generation on Design2Code, where only PRISMACODER-35B reproduces the layout.



Source | Target

Qwen3.5-9B | PRISMACODER-9B w/o RL | PRISMACODER-9B

Figure 20: Front-end editing on DesignBench, where only PRISMACODER-9B adds the requested story points, per-column sums, and header menu while keeping the board.

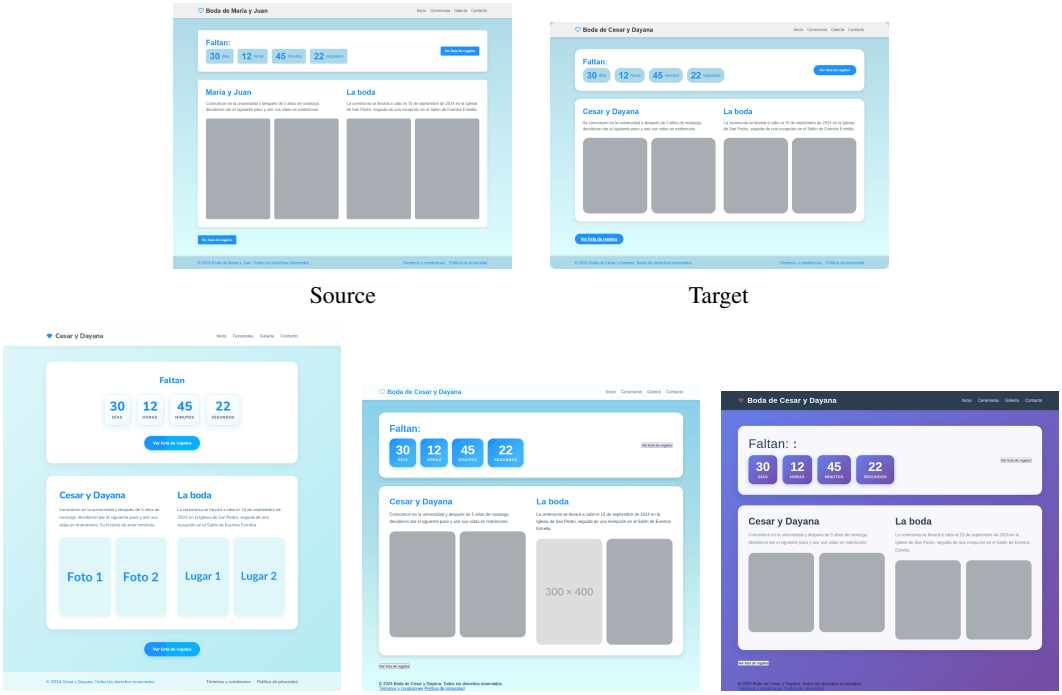


Figure 21: Front-end editing on DesignBench, where only PRISMACODER-35B applies the requested renaming, rounded corners, and countdown restyling.