VISION2CODE: A MULTI-DOMAIN BENCHMARK FOR EVALUATING IMAGE-TO-CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Image-to-code generation tests whether a vision-language model (VLM) can write executable code to reproduce an image. Existing benchmarks suffer in several ways, including narrow scope, dependence on source code, and generic evaluation criteria across very different domains. To address this, we introduce **Vision2Code**, a multi-domain and reference-code-free benchmark with 539 test examples from 15 datasets spanning six domains: charts and plots, geometry, graphs, scientific imagery, documents, and 3D spatial scenes. We also create a dataset-specific rubric that judges image reconstruction quality with emphasis on criteria relevant to that dataset. Taken together, our evaluation framework aligns better with human judgments from four annotators than generic VLM rubrics and pixel, embedding, and perceptual baselines. We study the robustness of our evaluation framework by computing agreement between two judges, evaluator repeatability, and the sensitivity to rubric weights and domain-specific errors. Results show that both VLM judges have low variance across repeated runs and preserve the same ranking of all nine evaluated models. Most of the introduced perturbations are also correctly detected by both judges using our rubric. We show that benchmark performance varies substantially across domains. Models do well on charts and graphs but struggle with spatial scenes, chemistry, documents, and scientific diagrams. To improve image-to-code generation capability, we introduce evaluator-guided self-training, where evaluator scores are used to select model-generated programs for fine-tuning. This improves Qwen3.5-9B from 1.60 to 1.86 and Phi-4-MM from 0.58 to 0.74 on test, with gains also holding under additional evaluators. Vision2Code provides a multi-domain testbed for measuring, diagnosing, and improving executable visual reconstruction.

1 INTRODUCTION

Image-to-code generation tests whether a vision-language model (VLM) can write executable code to reproduce an image. Unlike direct pixel generation, the output is editable, verifiable, interpretable, and reusable. This type of structured representation is also what makes tools like LaTeX, Excalidraw, and Figma useful. Generating code that recreates an image is also a strict test of visual understanding. A model might describe a chart correctly while getting its axis scale wrong, or recognize the nodes in a graph while reversing an edge. Recreating the image in code requires these details to be preserved. Prior image-to-code benchmarks have made substantial progress in studying image-to-code generation capabilities, but remain limited in several ways. First, most only cover a narrow visual domain. ChartMimic (Yang et al., 2025a), Plot2Code (Wu et al., 2025), RealChart2Code (Zhang et al., 2026a), and Chart2Code (Tang et al., 2025) focus on charts and plots, while Design2Code (Si et al., 2025), Web2Code (Yun et al., 2024), and Widget2Code (Zhang et al., 2026b) focus on webpages and UI elements. Second, broader benchmarks such as Omni-I2C (Zhou et al., 2026) rely on paired executable reference code. Paired source code makes code-level evaluation possible, but it also limits which images can be included. For many visual datasets, the original rendering program simply does not exist. Third, rendered outputs are typically evaluated using generic visual metrics or criteria shared across samples (Si et al., 2025; Roberts et al., 2024; Zhou et al., 2026). These can miss errors that are specific to a visual domain, such as incorrect graph topology, chemical bonds, geometric relations, or spatial depth. This makes it very difficult to evaluate the same model across different image domains, especially when the rendering source code is unavailable.

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.

We introduce **Vision2Code**, a multi-domain reference-code-free benchmark for image-to-code generation with 539 test examples across 15 heterogeneous datasets and six domains: charts and plots, geometry, graphs, scientific imagery, documents, and 3D spatial scenes. Generated code is executed, and the resulting image is compared with the source for reconstruction quality using dataset-specific rubrics and guardrails. Human evaluation with four annotators shows that these rubrics align more closely with human judgments than generic VLM rubrics and other baselines like SSIM, PSNR, CLIP ViT-B/32, DreamSim, and Qwen3-VL-Embedding-8B cosine similarity. We test the evaluator from several angles. First, we score benchmark performance using two different VLM judges and compare their rankings. We also rescore the same examples to see how much the scores vary, change the rubric weights and guardrails, and manually introduce domain-specific errors into otherwise good reconstructions. The two judges give the same ranking of the nine models, their scores change little across repeated runs, and most of the introduced errors lower the intended rubric category. Our evaluation of nine open-weight and proprietary models shows that image-to-code performance remains highly domain-dependent. Models do well on charts and graphs but struggle with spatial scenes, chemistry, documents, and scientific diagrams. Additionally, we develop an evaluator-guided self-training framework to improve image-to-code generation capabilities in the absence of ground truth source code. Base model generated image and code pairs are filtered by the evaluator scores for cases where the rendered output accurately matches the source but the model cannot reproduce that successful output consistently, thereby providing useful training signal. This filtering strategy allows for data curation that improves Qwen3.5-9B from 1.60 to 1.86 and Phi-4-MM from 0.58 to 0.74 on finetuning, with results remaining consistent under independent evaluators.

In summary, our contributions include: (1) **Vision2Code**, a multi-domain reference-code-free benchmark for image-to-code generation across 15 heterogeneous datasets and six domains: charts and plots, geometry, graphs, scientific imagery, documents, and 3D spatial scenes, (2) a domain-aware evaluation framework that scores rendered outputs using dataset-specific rubrics and guardrails, showing strong agreement with human annotators and across VLM judges and (3) an evaluator-guided self-training pipeline that uses rendered-output scores to filter model-generated image and code pairs into useful supervision, improving VLM image-to-code performance in the absence of ground truth code.

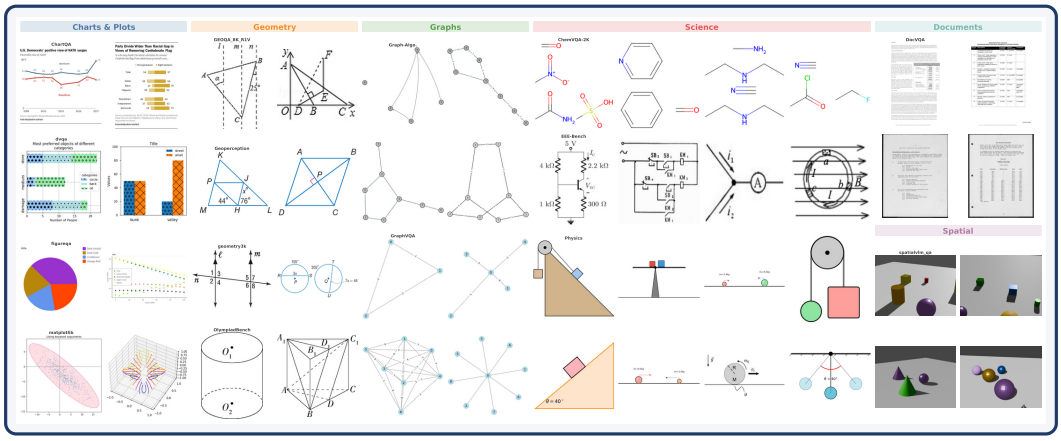


Figure 1: Vision2Code spans 15 heterogeneous datasets across six domains: charts and plots, geometry, graphs, science, documents, and spatial scenes. The benchmark is reference-code-free and multi-domain.

2 Related Work

Code as a Medium for Visual Reasoning. Several VLM systems use code as part of the reasoning process. VisProg (Gupta and Kembhavi, 2023), ViperGPT (Surís et al., 2023), and CodeVQA (Subramanian et al., 2023) generate code that combine perception modules to answer visual questions. Chameleon (Lu et al., 2023) and CodeAct (Wang et al., 2024) use code more broadly for planning and tool execution. Code can also serve as an intermediate visual representation. RECODE (Shen et al., 2025) derenders images into executable code, CodePlot-CoT (Duan et al., 2025) generates

Table 1: Comparison with recent image-to-code benchmarks. N is the number of evaluation samples; Multi indicates multiple domains; Ref.-free indicates evaluation without paired source code; Fine eval. indicates category-level or multi-criteria reconstruction evaluation; Domain eval. indicates evaluation criteria tailored to a visual domain or dataset. Render diag. indicates separate render-failure reporting; Human val. indicates validation against humans. ✓ = completely or partially supported, ✗ = not supported.

Benchmark	Scope	Samples	Multi	Ref.-free	Fine eval.	Domain eval.	Render diag.	Human val.
ChartMimic	Charts	4,800	✗	✓	✓	✓	✓	✓
Plot2Code	Scientific plots	368	✗	✓	✗	✗	✓	✓
RealChart2Code	Real-world charts	2,896	✗	✓	✓	✓	✓	✓
Chart2Code	Charts	2,186	✗	✓	✓	✓	✓	✓
Design2Code	Web pages	484	✗	✓	✓	✓	✗	✓
Web2Code	Web pages	1,198	✗	✓	✓	✓	✗	✗
Widget2Code	UI widgets	1,000	✗	✓	✓	✓	✗	✓
Image2Struct	Web, LaTeX, music	2,400	✓	✓	✗	✗	✓	✗
Omni-I2C	Technical graphics	1,080	✓	✗	✓	✗	✓	✓
Vision2Code	6 visual domains	539	✓	✓	✓	✓	✓	✓

plots as part of mathematical reasoning, and Visual Sketchpad (Hu et al., 2024a) lets VLMs draw through code. Visual Program Distillation (Hu et al., 2024b) studies how much programmatic reasoning can be distilled into a standalone VLM. In most of these works, the generated code is a means to solve another task. Vision2Code instead evaluates the generated code by how faithfully its execution recreates the source image.

Image-to-Code Benchmarks. Prior image-to-code benchmarks mainly focus on a particular type of output, restricting them to narrow scope. ChartMimic (Yang et al., 2025a), Plot2Code (Wu et al., 2025), RealChart2Code (Zhang et al., 2026a), and Chart2Code (Tang et al., 2025) evaluate charts and scientific plots. Design2Code (Si et al., 2025), Web2Code (Yun et al., 2024), WebSight (Laurençon et al., 2024), Widget2Code (Zhang et al., 2026b), and Figma2Code (Gui et al., 2026) focus on webpages and UI generation. Image2Struct (Roberts et al., 2024) looks at webpages, LaTeX, and musical scores, while UniSVG (Li et al., 2025a), VCode (Lin et al., 2025), SVGenius (Chen et al., 2025), VectorGym (Rodriguez et al., 2026), and OmniSVG (Yang et al., 2025b) study vector graphics. Concurrent with our work, Omni-I2C (Zhou et al., 2026) evaluates several types of technical graphics and multiple programming languages, but relies on reference code. Vision2Code does not require such reference programs, which allows us to include datasets where only the image is available. HumanEval-V (Zhang et al., 2024a) and MMCode (Li et al., 2024) also take images as inputs, but the goal there is to solve a programming problem shown in the image rather than reconstruct the image itself.

Evaluation of Image-to-Code Generation. Code generated by a VLM can execute successfully and still produce a poor reconstruction, requiring other image-level metrics. Standard image metrics provide useful baselines, but they measure different things. SSIM, PSNR, and MSE are sensitive to pixel-level differences, whereas CLIP (Radford et al., 2021) and DreamSim (Fu et al., 2023) measure broader visual similarity. Neither directly checks the structural details that matter for every dataset. Existing benchmarks use different methods for evaluation. Plot2Code (Wu et al., 2025) reports code pass rate, text-match ratio, and GPT-4V scores. ChartMimic (Yang et al., 2025a) evaluates both the generated code and rendered chart. Design2Code (Si et al., 2025) uses visual metrics together with human evaluation and Omni-I2C (Zhou et al., 2026) combines perceptual and symbolic measures. MLLMs have also been used directly as judges (Chen et al., 2024a). Vision2Code instead uses manually designed, dataset-specific criteria for evaluating image reconstruction quality. We test this evaluator against human ratings, a second VLM judge, standard image similarity metrics, and controlled reconstruction errors.

3 BENCHMARK

Task. Vision2Code evaluates image reconstruction capabilities in VLMs through image-to-code generation. Given a source image, a model must generate code whose execution recreates the image as faithfully as possible. The goal is not exact pixel matching, but preserving the semantic and

structural information in the image such as plotted values, labels, topology, notation, document layout, and spatial relations. Models are prompted to generate Python code using Matplotlib and NumPy, and are evaluated based on the rendered output rather than the specific code that produced it.

Domain coverage. Vision2Code includes six domains that test different parts of reconstruction. For charts and plots, models need to preserve numerical values, axes, scales, labels and the mapping between legends and plotted values. Geometry and graph examples place more emphasis on shape, proportions, connectivity and direction. Scientific figures are sensitive to small notation like chemical symbols, current direction and object placement, that can change the meaning of the image. Documents require both text preservation and layout recovery, while spatial scenes test object identity, depth, and relative 3D placement. This gives the benchmark a mix of quantitative, symbolic, text-heavy, and spatial images.

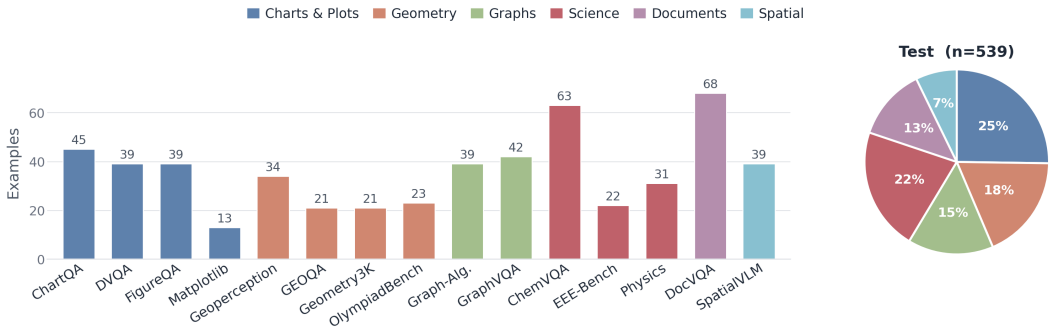


Figure 2: Vision2Code composition. Left: examples per dataset. Right: distribution across six visual domains.

Source data and sampling. Vision2Code uses 15 public visual datasets covering the six mentioned domains. We use the official test split when one is available. For the remaining datasets, we create a held-out pool with a fixed seed. After manual filtering, the final benchmark contains 539 test examples. Figure 2 shows how these examples are distributed across datasets and domains. Appendix B gives the source datasets, split construction, sampling procedure, and filtering details.

4 Evaluation Framework

Code execution. Evaluation begins by executing the code generated by the model. The benchmark uses Python with Matplotlib and NumPy because one runtime supports all six domains. The evaluation itself, however, is not tied to Python; it only requires the generated artifact to produce an image. Appendix M shows examples using Excalidraw scene JSON and standalone LaTeX. The generated code runs in a fixed environment and the rendered image is saved at the same resolution as the source. A score of 0 is directly given to code that fails, times out, or produces no valid image. Otherwise, the rendered image is passed to the VLM rater for evaluation. Figure 3 shows the evaluation pipeline.

Dataset-specific rubric. For each sample, the rater sees the source image, the rendered image, reference metadata, and the rubric for that dataset. Each rubric has four categories with weights that sum to one. Each category is scored from 0 to 5 in 0.1 increments, and the final score is computed as their weighted sum. The first two categories evaluate the most important aspects of the image. If either receives a very low score, defined by a threshold, the final rating is capped. We also use additional dataset-specific caps to account for severe domain-specific errors. If the rater returns an invalid response or leaves out a required category, we ask it to score the same pair again. The complete rules and prompts are listed in Appendix A.

Validation protocol. We compare our dataset-specific evaluator with human ratings and other baselines. As a direct baseline, we keep the VLM rater unchanged but replace our dataset-specific rubric with four equal-weight criteria used for every example: *core information fidelity*, *structure*

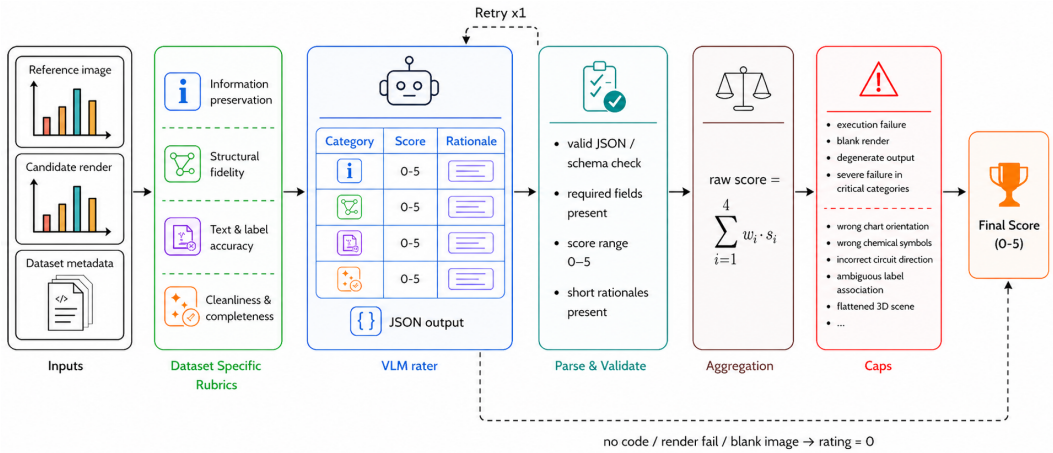


Figure 3: Vision2Code evaluation pipeline. Generated code is rendered into an image, evaluated by comparing against the source image with a dataset-specific rubric consisting of four categories, and aggregated with deterministic caps.

and layout fidelity, *text and annotation accuracy*, and *visual completeness*. This is done to isolate the effect of using dataset-specific criteria. We also include SSIM, PSNR, CLIP ViT-B/32, DreamSim, and Qwen3-VL-Embedding-8B. For Qwen3-VL, we test the image embedding alone and a version that also includes short dataset-specific text describing what should be preserved. We compare each score with the mean human rating using Pearson and Spearman correlation. Agreement among the four human raters is measured with Intraclass Correlation Coefficient (ICC). Appendix E gives the rating instructions, interface, and annotator details.

5 Experiments

We evaluate nine VLMs on Vision2Code, including both open-weight and proprietary models. The open-weight models are Qwen3.5-9B, Qwen3.5-397B-A17B, and Kimi-K2.5, with reasoning variants of the latter two. The proprietary models are GPT-5.4, GPT-5.4 Mini with and without reasoning, and Gemini-3.1-Flash-Lite-Preview. For every model, we generate code from the source image, execute it, and score the resulting render using the procedure in Section 4. Each reconstruction is scored separately by Qwen3.5-122B-A10B-GPTQ-Int4 and Kimi-K2.6. A failed render receives a score of 0. We report the macro-average across the 15 datasets as the overall benchmark score.

5.1 Benchmark Results

Table 2 reports the per-dataset scores and overall macro-average for all nine models on test. Results from both Qwen3.5-122B and Kimi-K2.6 are included to check whether the ranking changes with the choice of evaluator. Appendix J shows example reconstructions from each model.

Overall performance. On test, GPT-5.4 achieves the highest overall score, followed closely by GPT-5.4 Mini with reasoning. Among open-weight models, Kimi-K2.5 with reasoning performs best, while Qwen3.5-9B is substantially weaker. This leaves a clear gap between the strongest proprietary and open-weight models. At the same time, even the best models remain well below a perfect reconstruction score, showing that image-to-code generation is far from solved.

Domain variation. Models are strongest on charts, plots, and graph-like images, where reconstruction often relies on common visual primitives such as bars, lines, nodes, and labels. One possible reason is that such visuals are common in code and vision-language pretraining data, making their structure more familiar to current models. Performance drops on domains that require more precise symbolic, geometric, or spatial structure, including chemistry, circuits, documents, and 3D scenes. SpatialVLM-QA is the hardest dataset by a large margin, with even the best model scoring only 2.43 under the primary judge. The same broad pattern appears across both VLM judges,

Table 2: Vision2Code leaderboard using two VLM judges: Qwen3.5-122B-A10B-GPTQ-Int4 and Kimi-K2.6. Avg. is the macro-average across datasets. Abbreviations: +R = reasoning enabled; MPL = Matplotlib; GeoP = Geoperception; GEOQA = GEOQA_8K_R1V; Geom3K = Geometry3K; GAlg = Graph-Algorithms; GVQA = GraphVQA-Swift; Chem = ChemVQA-2K; EEE = EEE-Bench; Olymp = OlympiadBench; Doc = DocVQA; Spat = SpatialVLM-QA.

VLM Judge: Qwen3.5-122B-A10B-GPTQ-Int4

Model	Chart	DVQA	FigQA	MPL	GeoP	GEOQA	Geom3K	GAlg	GVQA	Chem	EEE	Phys	Olymp	Doc	Spat	Avg.
Open-weight models																
Qwen3.5-9B	3.10	3.39	2.83	1.91	2.00	2.00	0.98	0.69	1.79	0.07	0.50	2.08	1.28	1.23	0.10	1.60
Kimi-K2.5	3.07	4.42	3.77	3.63	3.32	3.18	3.48	2.86	3.73	1.73	2.72	4.18	2.34	2.44	1.11	3.06
Kimi-K2.5+R	3.39	4.57	4.15	3.55	3.61	3.55	3.31	2.59	4.08	3.02	2.87	4.38	2.71	2.92	0.70	3.29
Qwen3.5-397B-A17B	3.22	3.51	2.93	2.40	2.81	2.81	2.44	2.93	3.61	0.92	1.41	2.63	2.42	1.54	0.54	2.41
Qwen3.5-397B-A17B+R	3.91	4.49	3.84	2.89	3.51	3.52	2.48	3.01	4.03	1.59	2.72	3.42	3.27	2.07	0.27	3.00
Proprietary models																
GPT-5.4	4.60	4.84	4.28	3.71	4.64	4.07	4.43	3.81	4.69	3.83	4.20	4.61	4.64	4.08	1.30	4.12
GPT-5.4 Mini	3.62	4.70	4.45	3.39	4.50	3.56	4.43	3.89	4.04	3.48	3.56	4.51	4.23	3.36	1.98	3.85
GPT-5.4 Mini+R	4.45	4.70	4.78	4.35	4.56	3.47	4.10	4.12	4.26	3.80	4.00	4.46	4.14	3.90	2.43	4.10
Gemini-3.1-Flash-LitePr	4.17	4.71	4.21	3.48	3.66	3.41	3.64	3.22	4.29	0.73	2.92	3.81	3.10	2.14	0.05	3.17

VLM Judge: Kimi-K2.6

Model	Chart	DVQA	FigQA	MPL	GeoP	GEOQA	Geom3K	GAlg	GVQA	Chem	EEE	Phys	Olymp	Doc	Spat	Avg.
Open-weight models																
Qwen3.5-9B	2.92	3.14	2.63	2.11	1.68	2.02	1.15	0.72	1.86	0.11	0.82	2.27	1.50	1.22	0.12	1.62
Kimi-K2.5	2.83	4.23	3.39	3.78	2.88	2.97	2.91	2.77	3.67	1.62	2.86	3.95	2.45	2.21	1.18	2.91
Kimi-K2.5+R	3.05	4.31	3.58	3.67	2.90	2.89	2.92	2.72	4.06	2.62	2.98	4.02	2.52	2.68	0.92	3.06
Qwen3.5-397B-A17B	2.62	3.02	2.67	2.59	2.35	2.77	1.92	2.98	3.51	0.76	1.51	2.68	2.52	1.43	0.78	2.27
Qwen3.5-397B-A17B+R	3.76	4.12	3.39	2.99	2.86	2.81	2.13	2.93	3.84	1.36	2.58	3.33	3.03	1.85	0.29	2.75
Proprietary models																
GPT-5.4	4.32	4.74	3.99	3.60	3.84	3.71	3.69	3.82	4.44	3.41	3.76	4.55	4.23	3.84	1.46	3.83
GPT-5.4 Mini	3.43	4.44	4.03	3.38	3.81	3.20	3.70	3.73	3.96	3.15	3.16	4.45	3.99	3.12	1.97	3.57
GPT-5.4 Mini+R	4.37	4.47	4.36	4.09	3.89	3.35	3.69	4.01	3.98	3.51	3.38	4.11	4.03	3.66	2.25	3.81
Gemini-3.1-Flash-LitePr	3.96	4.54	3.85	3.37	3.18	2.97	3.08	3.13	4.19	0.88	2.69	3.45	2.97	1.96	0.09	2.95

suggesting that these domain differences are not specific to a particular evaluator. We analyze the corresponding domain-specific failure modes qualitatively in Section 5.3.2.

Reasoning effects. Reasoning improves the overall score for all three paired models under both VLM judges. However, the gains are not uniform across datasets. Some datasets get worse with reasoning, and several of these drops appear under both judges. SpatialVLM-QA is the clearest case, where reasoning reduces performance for both Kimi-K2.5 and Qwen3.5-397B-A17B. Similar decreases appear on Matplotlib and Graph-Algorithms for Kimi-K2.5, and on Geometry3K and Physics for GPT-5.4 Mini. One possible explanation is that reasoning helps with visual understanding, image decomposition and code planning, but can lose precise perceptual information about position, geometry, or layout.

5.2 Evaluator Robustness

Human alignment. We evaluate human alignment on 180 source–output pairs from test, balanced across six models and all 15 datasets. Four annotators independently rate all 180 pairs on a 0–5 scale, yielding 720 ratings. We use the mean of the four ratings as the human score. Human agreement is strong, with ICC(2,1) = 0.753 for a single annotator and ICC(2,4) = 0.924 for the four-rater mean. Table 3(a) compares both our evaluators with perceptual, pixel-level, embedding and generic-rubric baselines. The dataset-specific rubric has the highest correlation with human judgments under both VLM judges, reaching Pearson/Spearman correlations of 0.849/0.861 with Kimi-K2.6 and 0.836/0.826 with Qwen3.5-122B, with both outperforming their corresponding generic-rubric variant. DreamSim is the strongest non-VLM baseline, while CLIP, embedding similarity, SSIM, and PSNR show lower agreement with human ratings. Additional details are provided in Appendix E.

Judge agreement. To test whether the leaderboard depends on the choice of VLM evaluator, we rescore all images using Kimi-K2.6 (Table 2). The two judges assign slightly different scores with

Table 3: Evaluator validation. (a) Human alignment and (b) Rubric sensitivity, measured by Pearson r and Spearman ρ. (c) Judge Repeatability, original score (Orig.), 3-run mean, and variance (Var., in units of 10^{-3}).

(a) Human alignment

Metric	r	ρ
SSIM	-0.054	-0.023
PSNR	0.057	0.092
CLIP	0.443	0.460
Cosine (image)	0.627	0.716
Cosine (+text)	0.576	0.663
DreamSim	0.686	0.726
Generic (Qwen)	0.785	0.817
Generic (Kimi)	0.807	0.820
Ours (Qwen)	**0.836**	**0.826**
Ours (Kimi)	**0.849**	**0.861**

(b) Rubric sensitivity

Variant	r	ρ
VLM Judge: Qwen3.5-122B		
Uniform	0.835	0.823
Uncapped	0.817	0.794
Complete	**0.836**	**0.826**
VLM Judge: Kimi-K2.6		
Uniform	0.844	0.860
Uncapped	0.832	0.836
Complete	**0.849**	**0.861**

(c) Judge repeatability

Model	Qwen judge			Kimi judge		
	Orig.	Mean	Var.	Orig.	Mean	Var.
Qwen3.5-9B	2.086	2.077	1.052	1.981	1.979	0.012
Kimi-K2.5	3.413	3.421	0.290	3.110	3.097	0.144
Kimi-K2.5+R	3.625	3.638	0.973	3.340	3.360	0.636
Qwen3.5-397B	2.555	2.627	4.053	2.545	2.539	0.068
Qwen3.5-397B+R	3.635	3.615	0.844	3.212	3.211	0.506
GPT-5.4	4.133	4.124	0.223	3.750	3.726	0.795
GPT-5.4 Mini	3.950	3.967	0.306	3.770	3.760	0.300
GPT-5.4 Mini+R	4.310	4.276	1.812	3.823	3.787	1.012
Gemini Flash-Lite	3.348	3.341	0.181	3.022	3.041	0.337
All pairs	**3.453**	**3.456**	**0.016**	**3.175**	**3.169**	**0.029**

Kimi-K2.6 generally assigning lower scores, but preserve exactly the same ranking across all models. Their model-level scores have a Pearson correlation of 0.999, while scores across the 4,845 outputs with scores available from both judges have a Pearson correlation of 0.936. We also find no clear evidence of model-family preference. Despite Kimi-K2.6 serving as a judge, the Kimi-K2.5 models do not improve in rank or receive unusually favorable score shifts compared with other models. Kimi-K2.6 also aligns slightly better with human ratings than Qwen3.5-122B for both rubric types (Table 3(a)).

Rubric sensitivity. We recompute scores from the category-level scores to study how much the final results depend on the rubric design and chosen weights. In the uniform variant, all four categories receive equal weight while the caps and guardrails are kept unchanged. In the uncapped variant, we keep the original weights but remove the caps and guardrails. As shown in Table 3(b), the complete rubric has the highest Pearson and Spearman correlation for both judges. Uniform weighting changes the correlations only slightly for both judges. Removing the caps and guardrails causes a larger drop. This suggests that the exact category weights matter less than the rules that prevent a severe error in one important category from being compensated by high scores in the others.

Judge repeatability. We check whether the VLM scores change when the same examples are rated again. For each judge, we score the same 539 source–output pairs three times: the original run and two reruns. The pairs were distributed as evenly as possible across all nine models. As shown in Table 3(c), the three-run means stay close to the original scores and the run-to-run variance is small. For the two reruns, the individual-output Pearson/Spearman correlations are 0.985/0.983 for Qwen3.5-122B and 0.983/0.974 for Kimi-K2.6. Their mean absolute score differences are 0.089 and 0.157, and 95.0% and 93.3% of scores differ by at most 0.5 points. Across all three runs, both judges achieve a single-run agreement ICC of 0.981. Kimi keeps the same ranking of all nine models in every run, while Qwen has one swap between the models at ranks four and five.

Domain-specific error sensitivity. To test whether the evaluators detect the domain-specific errors represented by our rubric, we selected 71 reconstructions spanning 14 datasets across the nine evaluated models. Every selected reconstruction received a score of at least 4.5 from both judges. We then modified each program to introduce one controlled error targeting a single rubric category while leaving the remaining content unchanged. The intended category score decreased in 65/71 cases for Qwen3.5-122B and 68/71 for Kimi-K2.6, with mean decreases of 2.36 and 2.02 points on the 0–5 scale. The final score decreased in 66/71 cases for Qwen and all 71 cases for Kimi, and no perturbed image received a higher score. Both judges struggled most with geometry, missing things like a shortened trapezoid segment, a broken parallelogram connection, and a removed right-angle label. Qwen also missed an absent chart bar, a changed chemical bond, and a circuit gap, while Kimi missed a modified graph edge weight. Qualitative examples are provided in Appendix L.

Table 4: Execution failure analysis on test. Render success is reported as a percentage, while failure columns report counts among failed renders. API = hallucinated APIs or invalid arguments; Shape/3D = shape mismatch or 3D geometry errors; Other = runtime errors, no-output exits, or timeouts; Syntax = syntax errors or truncated code; MissDep = missing dependencies or nonexistent files.

Model	Render success (%)	Counts by failure type					
		Total	API	Shape/3D	Other	Syntax	MissDep
Qwen3.5-9B	75.1	134	28	37	22	45	2
Kimi-K2.5	88.5	62	21	8	3	13	17
Kimi-K2.5+R	90.9	49	5	11	28	4	1
Qwen3.5-397B-A17B	87.9	65	16	22	12	14	1
Qwen3.5-397B-A17B+R	81.6	99	40	12	27	14	6
GPT-5.4	97.6	13	4	5	3	1	0
GPT-5.4 Mini	94.1	32	7	6	18	0	1
GPT-5.4 Mini+R	98.9	6	1	5	0	0	0
Gemini-3.1-Flash-LitePr	92.2	42	37	1	2	2	0
All models	89.7	502	159	107	115	93	28

5.3 Failure Analysis

We separate failures into two types. Some generated programs fail to run, time out, or produce no valid image. Others execute successfully but still reconstruct the source poorly. We analyze these execution and reconstruction failures separately.

5.3.1 Execution Failures

Table 4 summarizes render success and counts of failure types for each model. Render success ranges from 75.1% for Qwen3.5-9B to 98.9% for GPT-5.4 Mini with reasoning. However, a high render success does not necessarily mean better reconstruction quality. Qwen3.5-397B-A17B, for example, renders less often with reasoning even though its benchmark score is higher, while GPT-5.4 and GPT-5.4 Mini with reasoning both render almost all examples but still differ in reconstruction quality. The kinds of failures also differ by model. Qwen3.5-9B fails most often because of syntax errors, whereas Gemini-Flash-Lite-Preview mostly fails through invalid or hallucinated API calls. GPT-5.4 Mini without reasoning mainly encounters runtime or no-output failures, whereas its reasoning variant has very few failures. Reasoning can change the type of failure rather than simply reduce the total number. For Qwen3.5-397B-A17B, Shape/3D errors become less common with reasoning, but API-related failures increase and overall render success drops.

5.3.2 Reconstruction Failures

In charts and plots, models sometimes flip axes, fail to preserve plot trends, or produce overlapping legends and labels. Geometry examples often contain incorrect proportions, missing components, wrong shapes, and overlapping text. Graph examples show reversed arrow directions, incorrect overall topology, and wrong node or edge labels. Chemistry examples frequently contain incorrect molecular shapes, missing double or triple bonds, wrong atom identities, and heavy overlap. Physics examples show spatial inconsistencies such as objects overlapping slopes or appearing to float. Circuit examples often contain missing components, malformed component shapes or disconnected paths. Document examples may capture the rough content while failing to preserve formatting, relative proportions, and layout. Spatial scenes are particularly difficult, with models often failing to preserve depth and relative object placement. Qualitative examples are provided in Appendix I

5.4 Evaluator-Guided Self-Training

Most visual datasets do not include the code that produced each image, which makes them difficult to use directly for image-to-code supervision. We therefore ask whether a model's own successful generations can be used as training examples for supervised finetuning. Figure 4 summarizes the

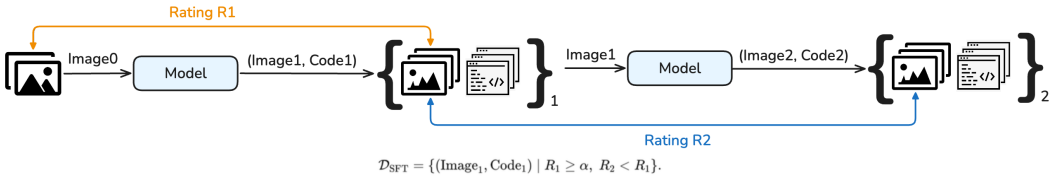


Figure 4: Evaluator-guided self-training pipeline. First-stage image-code pairs are used for SFT when the render scores well against the source ($R_1 \geq \alpha$) but is not stably reconstructed in the second stage ($R_2 < R_1$).

pipeline. Starting from the source image I_0, the model generates corresponding code C_1, which renders to I_1. We then give I_1 back to the same model and prompt it to generate a second code C_2, corresponding to the reconstruction of I_1, producing I_2. We define $R_1 = R(I_0, I_1)$ as the quality of the first reconstruction and $R_2 = R(I_1, I_2)$ as how well the model can reconstruct its own output a second time. Both ratings are provided by the same VLM evaluator. We use (R_1, R_2) to decide which first-stage pairs (I_1, C_1) can provide useful training signal for fine-tuning. Our main filter, $R_1 \geq \alpha,\ R_2 < R_1$, keeps examples where the first reconstruction is good but the model is less successful when asked to reproduce that render again. These examples give us code for images that the model does not yet reconstruct consistently and we use that to our advantage. Table 5 compares this rule with four alternatives. The $R_1 \geq \alpha,\ R_2 < R_1$ subset gives the best result for both tested models. Qwen3.5-9B improves from 1.60 to 1.86 and Phi-4-MM from 0.58 to 0.74. Since the same Qwen3.5-122B evaluator is used both to select self-training samples and to evaluate the resulting fine-tuned model, the improvement may partially reflect optimization toward evaluator-specific preferences. To test this, we use 2 additional evaluators, Kimi-K2.6 and Qwen3.6-Plus, and see that the improvement is consistent across judges. The other selection rules are less effective, suggesting that combining a high-quality first reconstruction with a drop in self-reconstruction provides a more useful training subset. Appendix F gives the data counts and fine-tuning details.

Table 5: Evaluator-guided self-training. The ablation reports test scores under the primary Qwen3.5-122B evaluator. The final three columns report the best-filter improvement over baseline under the primary judge and two additional evaluators, Kimi-K2.6 and Qwen3.6-Plus. Here, $\alpha = 4$.

	Performance of different evaluator-guided filters on test						Best-filter improvement over baseline		
Model	**Baseline**	**All Valid**	$R_1 \geq \alpha$ $R_2 \geq R_1$	$R_1 < \alpha$	$R_1 \geq \alpha$	$R_1 \geq \alpha$ $R_2 < R_1$	**Qwen3.5-122B Δ**	**Kimi-K2.6 Δ**	**Qwen3.6-Plus Δ**
Qwen3.5-9B	1.5970	1.7233	1.7722	1.6962	1.8567	**1.8645**	**+0.2675**	**+0.2113**	**+0.2118**
Phi-4-MM	0.5756	0.4097	0.5450	0.6349	0.6299	**0.7414**	**+0.1658**	**+0.1655**	**+0.1895**

6 Conclusion

Vision2Code evaluates image-to-code generation across 15 datasets and six visual domains without requiring paired source code. The dataset-specific evaluator agrees more closely with human ratings than the generic rubric and the other automatic metrics we test. The benchmark also shows that current models are much stronger on regular charts and graphs than on spatial scenes, scientific diagrams, and documents. We further show that evaluator-selected model outputs can be used for self-training, improving both Qwen3.5-9B and Phi-4-MM. There are still several limitations. The benchmark inherits the coverage of its source datasets, the main leaderboard uses only Python with Matplotlib and NumPy, and the evaluator is still based on VLM judgments that can miss subtle errors. These are useful directions for extending the benchmark in future work.

AI use statement

We used generative AI tools to assist with refining experimental methodology and developing analysis code. We also used these tools for literature search and summarization, polishing manuscript text, and improving the presentation of tables and figures. All AI-assisted code was executed and verified by the authors, and all reported results, claims, and citations were checked against our ex-

periments and source materials. The authors reviewed and take responsibility for the final content of the paper.

Ethics statement

Vision2Code is constructed from existing public datasets and the Matplotlib gallery. Source licenses, access conditions, provenance, and redistribution terms are documented for each dataset. Generated code and rendered outputs are handled under the same data-governance policy as the corresponding benchmark images. Our human evaluation involved four annotators rating reconstruction fidelity; no additional personal or sensitive information was required for the evaluation.

Reproducibility statement

The appendix documents benchmark construction and splits, generation and rendering settings, evaluation prompts and dataset-specific rubrics, human validation, self-training, additional analyses and failure taxonomy. Our release will include benchmark manifests, source-image references or images where redistribution permits, provenance and license metadata, split-construction scripts, generation and evaluation prompts, rendering code, rubric definitions and cap rules, model-output manifests, render-success diagnostics, and scripts for computing benchmark summaries.

References

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.

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*, pages 3006–3028, 2025.

Jiajun Zhang, Yuying Li, Zhixun Li, Xingyu Guo, Jingzhuo Wu, Leqi Zheng, Yiran Yang, Jianke Zhang, Qingbin Li, Shannan Yan, et al. Realchart2code: Advancing chart-to-code generation with real data and multi-task evaluation. *arXiv preprint arXiv:2603.25804*, 2026a. URL https://arxiv.org/abs/2603.25804.

Jiahao Tang, Henry Hengyuan Zhao, Lijian Wu, Zijian Zhang, Yifei Tao, Dongxing Mao, Yang Wan, Jingru Tan, Min Zeng, Min Li, and Alex Jinpeng Wang. Chart2code: Chart-to-code generation with multi-modal large language models. *arXiv preprint arXiv:2510.17932*, 2025. URL https://arxiv.org/abs/2510.17932.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In Luis Chiruzzo, Alan Ritter, and Lu Wang, editors, *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)*, pages 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/.

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.

Cheng Zhang, Wenjing Wang, Hao Wang, Nuo Chen, Yinheng Chen, Qifan Wang, Tsung-Yi Ho Chen, and Xiang Ren. Widget2code: Benchmarking and developing code generation from visual

widgets. *arXiv preprint arXiv:2512.19918*, 2026b. URL https://arxiv.org/abs/2512.19918.

Jiawei Zhou, Chi Zhang, Xiang Feng, Qiming Zhang, Haibo Qiu, Lihuo He, Dengpan Ye, Xinbo Gao, and Jing Zhang. Omni-i2c: A holistic benchmark for high-fidelity image-to-code generation. *arXiv preprint arXiv:2603.17508*, 2026.

Josselin S Roberts, Tony Lee, Chi H Wong, Michihiro Yasunaga, Yifan Mai, and Percy Liang. Image2struct: Benchmarking structure extraction for vision-language models. *Advances in Neural Information Processing Systems*, 37:115058–115097, 2024.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14953–14962, 2023.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11888–11898, 2023.

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular visual question answering via code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 747–761, 2023.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 43447–43478. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/871ed095b734818cfba48db6aeb25a62-Paper-Conference.pdf.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Junhong Shen, Mu Cai, Bo Hu, Ameet Talwalkar, David A Ross, Cordelia Schmid, and Alireza Fathi. Recode: Reasoning through code generation for visual question answering. *arXiv preprint arXiv:2510.13756*, 2025.

Chengqi Duan, Kaiyue Sun, Rongyao Fang, Manyuan Zhang, Yan Feng, Ying Luo, Yufang Liu, Ke Wang, Peng Pei, Xunliang Cai, et al. Codeplot-cot: Mathematical visual reasoning by thinking with code-driven images. *arXiv preprint arXiv:2510.11718*, 2025.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379, 2024a.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9590–9601, June 2024b.

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.

Yi Gui, Jiawan Zhang, Yina Wang, Tianran Ma, Yao Wan, Shilin He, Dongping Chen, Zhou Zhao, Wenbin Jiang, Xuanhua Shi, et al. Figma2code: Automating multimodal design to code in the wild. *arXiv preprint arXiv:2604.13648*, 2026.

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*, pages 13156–13163, 2025a.

Kevin Qinghong Lin, Yuhao Zheng, Hangyu Ran, Dantong Zhu, Dongxing Mao, Linjie Li, Philip Torr, and Alex Jinpeng Wang. Vcode: a multimodal coding benchmark with svg as symbolic visual representation. *arXiv preprint arXiv:2511.02778*, 2025.

Siqi Chen, Xinyu Dong, Haolei Xu, Xingyu Wu, Fei Tang, Hang Zhang, Yuchen Yan, Linjuan Wu, Wenqi Zhang, Guiyang Hou, et al. Svgenius: Benchmarking llms in svg understanding, editing and generation. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 13289–13296, 2025.

Juan Rodriguez, Haotian Zhang, Abhay Puri, Tianyang Zhang, Rishav Pramanik, Meng Lin, Xiaoqing Xie, Marco Terral, Darsh Kaushik, Aly Shariff, et al. Vectorgym: A multitask benchmark for svg code generation, sketching, and editing. *arXiv preprint arXiv:2603.29852*, 2026.

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

Fengji Zhang, Linquan Wu, Huiyu BAI, Guancheng Lin, Xiao Li, Xiao Yu, Yue Wang, Bei Chen, and Jacky Keung. Humaneval-v: Evaluating visual understanding and reasoning abilities of large multimodal models through coding tasks, 2024a. URL https://openreview.net/forum?id=KRdiRGSNc9.

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*, pages 736–783, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

Stephanie Fu, Netanel Yakir Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=DEiNSfh1k7.

Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. MLLM-as-a-judge: Assessing multimodal LLM-as-a-judge with vision-language benchmark. In *Proceedings of the 41st International Conference on Machine Learning*, pages 6562–6595, 2024a.

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*, pages 2263–2279, Dublin, Ireland, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. DVQA: Understanding data visualizations via question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, 2018. doi: 10.1109/CVPR.2018.00592. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Kafle_DVQA_Understanding_Data_CVPR_2018_paper.html.

Samira Ebrahimi Kahou, Vincent Michalski, Adam Atkinson, Ákos Kádár, Adam Trischler, and Yoshua Bengio. FigureQA: An annotated figure dataset for visual reasoning. In *International Conference on Learning Representations Workshop*, 2018. URL https://arxiv.org/abs/1710.07300.

The Matplotlib Development Team. Matplotlib gallery. https://matplotlib.org/stable/gallery/index.html, 2026. Accessed 2026-05-02.

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

Jiarui Zhang, Ollie Liu, Tianyu Yu, Jinyi Hu, and Willie Neiswanger. Euclid: Supercharging multimodal llms with synthetic high-fidelity visual descriptions, 2024b.

Jiaqi Chen, Jianheng Tang, Jinghui Qin, Xiaodan Liang, Lingbo Liu, Eric Xing, and Liang Lin. GeoQA: A geometric question answering benchmark towards multimodal numerical reasoning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 513–523, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.46. URL https://aclanthology.org/2021.findings-acl.46/.

Lai Wei, Yuting Li, Chen Wang, Yue Wang, Linghe Kong, Weiran Huang, and Lichao Sun. First SFT, second RL, third UPT: Continual improving multi-modal LLM reasoning via unsupervised post-training, 2025. URL https://arxiv.org/abs/2505.22453. Associated with the WaltonFuture GeoQA-8K-direct-synthesizing dataset release.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-GPS: Interpretable geometry problem solving with formal language and symbolic reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6774–6786, Online, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.528. URL https://aclanthology.org/2021.acl-long.528/.

Ang Li, Charles L. Wang, Deqing Fu, Kaiyu Yue, Zikui Cai, Wang Bill Zhu, Ollie Liu, Peng Guo, Willie Neiswanger, Furong Huang, Tom Goldstein, and Micah Goldblum. Zebra-CoT: A dataset for interleaved vision-language reasoning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c6XIVI3TiQ. Used for Zebra-CoT Graph-Algorithms and Physics subsets.

chandrabhuma. GraphVQA-Swift. Hugging Face dataset, 2026. URL https://huggingface.co/datasets/chandrabhuma/GraphVQA-Swift. Dataset artifact; no verified academic parent publication identified; accessed 2026-05-02.

Chandra Mohan Bhuma. ChemVQA-2K: A visual question answering dataset for molecular understanding. Hugging Face / IEEE DataPort dataset artifact, 2025. URL https://huggingface.co/datasets/chandrabhuma/ChemVQA-2K.

Ming Li, Jike Zhong, Tianle Chen, Yuxiang Lai, and Konstantinos Psounis. EEE-Bench: A comprehensive multimodal electrical and electronics engineering benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13337–13349, 2025b. URL https://openaccess.thecvf.com/content/CVPR2025/html/Li_EEE-Bench_A_Comprehensive_Multimodal_Electrical_And_Electronics_Engineering_Benchmark_CVPR_2025_paper.html.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.211/.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. DocVQA: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2199–2208, 2021. doi: 10.1109/WACV48630.2021.00225. URL https://openaccess.thecvf.com/content/WACV2021/html/Mathew_DocVQA_A_Dataset_for_VQA_on_Document_Images_WACV_2021_paper.html.

Litian2002. Synthetic spatial vision-language question answering dataset. Hugging Face dataset, 2025. URL https://huggingface.co/datasets/Litian2002/spatialvlm_qa. Dataset artifact; accessed 2026-05-02.

Boyuan Chen, Zhuo Xu, Sean Kirmani, Brian Ichter, Dorsa Sadigh, Leonidas Guibas, and Fei Xia. SpatialVLM: Endowing vision-language models with spatial reasoning capabilities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14455–14465, 2024b. URL https://openaccess.thecvf.com/content/CVPR2024/html/Chen_SpatialVLM_Endowing_Vision-Language_Models_with_Spatial_Reasoning_Capabilities_CVPR_2024_paper.html.

A VLM EVALUATION RUBRICS

We specify the rubric-based VLM evaluation protocol used for the reported benchmark scores. The same rubric prompt is used across raters and the default rater is Qwen3.5-122B-A10B-GPTQ-Int4. The evaluator receives the reference image, the rendered candidate image, dataset metadata, and a dataset-specific rubric. It returns only category-level scores and short rationales; the final score is computed deterministically outside the model using the rubric weights and additional guardrails.

A.1 RATER PROMPT

System prompt

You are a rigorous but calibrated evaluator for image recreation quality. Compare the reference image against the candidate rendered image. Judge only what is visibly present in the images.

Use the dataset-specific rubric and instructions provided by the user message. Be strict about semantically wrong recreations, but do not over-penalize differences that the rubric explicitly says to treat as secondary.

Scoring anchors on a 0.0 to 5.0 scale: 5.0 = near-exact or clearly excellent for that dataset; 4.0 = strong recreation with minor issues; 3.0 = good but with noticeable structural, text, or semantic problems; 2.0 = partial recreation with important errors; 1.0 = major mismatch; 0.0 = missing, unreadable, blank, or fundamentally broken.

Return JSON only. Do not include markdown fences.

For each sample, the user message presents the reference image, the candidate rendered image, and the following instruction template:

User prompt template

Evaluate the candidate recreation against the reference image using the dataset-specific rubric below. The prompt includes the rubric version, resolved dataset name, dataset-specific guidance, dataset-specific notes, category definitions with weights, and reference metadata.

Return JSON with exactly these top-level keys: `category_scores`, `rationales`, `strengths`, `issues`, and `overall_summary`. The `category_scores` object must contain every required category id with a numeric score from 0.0 to 5.0, using 0.1 increments when needed. The `rationales` object must contain the same category ids, with one short rationale per category. `strengths` should contain 1–3 short bullets, `issues` should contain 1–5 short bullets, and `overall_summary` should contain 1–3 concise sentences.

Judge what matters for the dataset-specific rubric, not superficial similarity alone. For easy datasets, preserve recreation structure, not just semantic content. For hard or noisy references, be strict but do not penalize differences that the source itself makes ambiguous. Use 0.0 or 1.0 for severe failures. Do not compute the final 0.0 to 5.0 rating yourself; provide category scores only.

If the rater output is not valid JSON or is missing required category ids, we issue a repair prompt asking for corrected JSON only with the same top-level keys and the exact required category ids. The repair prompt does not change the images or rubric.

A.2 AGGREGATION AND GUARDRAILS

Each rubric contains four weighted categories whose weights sum to 1.0. The rater assigns a score $s_i \in [0, 5]$ to each category, rounded to one decimal place. We compute the provisional visual rating as

$$r_{\text{raw}} = \sum_i w_i s_i.$$

The final score is r_{raw} after deterministic caps. If code execution fails or no valid render is produced, the final score is 0.0. If the candidate image is unreadable, missing, blank, near-monochrome, or otherwise degenerate, the score is capped at 0.5. The first two rubric categories are treated as critical semantic categories; low scores on these categories trigger the profile-dependent caps in Table 6. Dataset-specific caps in Table 7 further prevent high final scores when a key semantic failure is present.

Table 6: Cap profiles used by the rubric aggregator. A cap limits the final score to at most the listed value when its condition is met.

Condition	Strict	Balanced	Hard
Any critical category $\leq$ threshold	$\leq 1.5 \Rightarrow 2.5$	$\leq 1.5 \Rightarrow 2.8$	$\leq 1.2 \Rightarrow 3.0$
Both critical categories $\leq$ threshold	$\leq 2.5 \Rightarrow 3.5$	$\leq 2.5 \Rightarrow 4.0$	$\leq 2.2 \Rightarrow 4.0$
Near-perfect guardrail	$r_{\text{raw}} \geq 4.8$ and critical $< 4.6 \Rightarrow 4.5$	$r_{\text{raw}} \geq 4.8$ and critical $< 4.4 \Rightarrow 4.5$	$r_{\text{raw}} \geq 4.8$ and critical $< 4.2 \Rightarrow 4.5$

A.3 Dataset-Specific Rubrics

Table 7 gives the complete dataset-specific rubrics. The table uses human-readable category names; the evaluator prompt uses corresponding machine-readable category ids with the same weights and order. The category order is meaningful: the first two categories are the critical categories used by the guardrails above.

Table 7: Dataset-specific rubrics. Category weights are shown in parentheses. Caps are applied after weighted aggregation.

Dataset	Categories and weights	Guidance and dataset-specific caps
ChartQA *Strict*	Numeric information (.30) Structure and layout (.30) Text, labels, and legibility (.25) Cleanliness and completeness (.15)	Easy chart recreation; preserve information and chart structure. Penalize wrong orientation/type, crowded labels, illegible text, bad title placement, or legend overlap. Cap at 3.5 for orientation/type mismatch when numeric information is mostly preserved but structure is weak; cap at 2.5 for material numeric or text errors.
DVQA *Strict*	Numeric information (.30) Structure and layout (.30) Text, labels, and legibility (.25) Cleanliness and completeness (.15)	Easy chart recreation; preserve exact plotted information, faithful structure, and readable labels. Cap at 3.5 for orientation/type mismatch when numeric information is mostly preserved but structure is weak; cap at 2.5 for material numeric or text errors.
FigureQA *Strict*	Numeric information (.30) Structure and layout (.30) Text, labels, and legibility (.25) Cleanliness and completeness (.15)	Easy chart recreation; semantic preservation alone is insufficient for top scores. Penalize meaningful plot-structure changes and crowded or illegible labels. Cap at 3.5 for orientation/type mismatch when numeric information is mostly preserved but structure is weak; cap at 2.5 for material numeric or text errors.
Matplotlib *Strict*	Layout and composition (.30) Plot elements (.25) Text, ticks, and annotations (.25) Style and rendering (.20)	Coding-heavy recreation with ground-truth plotting code. High scores require genuinely close layout, plot elements, ticks, annotations, and rendering. Unknown library-specific style differences matter less than missing structure. Cap at 3.0 for wrong layout or missing major plot elements.
Geoperception *Strict*	Geometric structure (.35) Numbers, labels, and markers (.30) Association and placement (.20) Readability and completeness (.15)	Geometry-heavy figures; structure, labels, markers, and clear annotation binding are central. Small label shifts are acceptable only when the target remains unambiguous. Cap at 2.5 for wrong geometric relations or ambiguous label association.
GEOQA_8K R1V *Strict*	Geometric structure (.35) Numbers, labels, and markers (.30) Association and placement (.20) Readability and completeness (.15)	Geometry-heavy figures; precise structure, labels, markers, numbers, and annotation association matter. Slight label shifts are acceptable if association remains unambiguous. Cap at 2.5 for wrong geometric relations or ambiguous label association.
Geometry3K *Strict*	Geometric structure (.35) Numbers, labels, and markers (.30) Association and placement (.20) Readability and completeness (.15)	Geometry problem diagrams; approximate correctness is not sufficient for high scores. Angle or length text may move slightly only if association remains obvious. Cap at 2.5 for wrong geometric relations or ambiguous label association.
Graph-Algorithms *Strict*	Graph structure (.40) Ordering and direction semantics (.25) Labels and numbers (.20) Readability and completeness (.15)	Graph topology, directions, ordering, weights, and labels are central. Cosmetic layout differences matter less than exact structure. If the source is cluttered, avoid over-penalizing ambiguity already present. Cap at 2.5 for wrong topology, ordering, or direction semantics.
GraphVQA-Swift *Strict*	Graph structure (.35) Labels and numbers (.30) Attribute mapping (.20) Readability and completeness (.15)	Graph structure, label precision, and attribute-to-node/edge binding are central. Wrong label ordering or wrong attribute attachment is important. Cap at 2.5 for wrong label ordering or attribute binding.

Continued on next page

Table 7: Dataset-specific VLM rating rubrics (continued).

Dataset	Categories and weights	Guidance and dataset-specific caps
ChemVQA-2K *Balanced*	Molecular structure (.40) Chemical symbols (.30) Bond lines and markers (.20) Readability and completeness (.10)	Focus on chemistry semantics: molecular structure, chemical symbols, bond lines, and semantically meaningful colored difference markers. Small orientation changes matter less than structure and symbols. Cap at 2.5 for wrong molecular structure or wrong chemical symbols.
EEE-Bench *Balanced*	Circuit topology and connectivity (.35) Component identity and symbols (.25) Labels, values, and directions (.25) Readability and completeness (.15)	Circuit topology, component identity, labels, values, signs, and current/voltage direction matter most. Be lenient on cosmetic symbol style when identity is clear. Cap at 2.5 for wrong topology, labels/values, or current/voltage direction.
Physics *Balanced*	Physical setup (.35) Vectors, labels, and numbers (.30) Object geometry consistency (.20) Readability and completeness (.15)	Focus on physical setup, object placement, vectors, labels, and numerical givens. Wrong force direction or physically inconsistent placement should be penalized strongly. Cap at 2.5 for wrong vector direction or physically inconsistent placement.
OlympiadBench *Balanced*	Problem structure and semantics (.35) Labels, numbers, and symbols (.25) Spatial precision (.20) Readability and completeness (.20)	Harder and denser problem figures. Judge whether the recreated figure supports the same downstream reasoning problem. Do not over-penalize small cosmetic differences that do not change meaning. Cap at 2.5 for wrong core problem semantics.
DocVQA *Balanced*	Text information (.35) Document structure and layout (.30) Legibility and non-crowding (.20) Key visual elements (.15)	Judge text and structure first. Be strict but fair for dense or difficult infographics; exact decorative styling matters less than information and structure. Cap at 2.5 for materially wrong key text or numbers.
SpatialVLM-QA *Hard*	Object shape and identity (.25) Spatial relations (.35) 3D depth rendering (.25) Color and scene completeness (.15)	Hard 3D spatial reconstruction. Judge object identity, spatial relations, depth, and scene completion carefully. Correct objects with a flattened 2D recreation should not score highly. Cap at 2.5 for flattened 2D scenes that preserve objects but lack depth; cap at 2.0 for wrong core spatial relations.

B Benchmark Construction

Data sources. Vision2Code is built from 15 visual source datasets. Each example is converted to a common schema containing a source image, source metadata, source provenance, dataset identity, and a stable sample identifier. The sources are ChartQA (Masry et al., 2022), DVQA (Kafle et al., 2018), FigureQA (Kahou et al., 2018), Matplotlib Gallery examples (The Matplotlib Development Team, 2026; Hunter, 2007), Geoperception (Zhang et al., 2024b), GEOQA_8K_R1V from the GeoQA-8K direct-synthesizing release associated with GeoQA and MM-UPT (Chen et al., 2021; Wei et al., 2025), Geometry3K from Inter-GPS (Lu et al., 2021), Graph-Algorithms and Physics from Zebra-CoT (Li et al., 2026), GraphVQA-Swift (chandrabhuma, 2026), ChemVQA-2K (Bhuma, 2025), EEE-Bench (Li et al., 2025b), OlympiadBench (He et al., 2024), DocVQA (Mathew et al., 2021), and SpatialVLM-QA (Litian2002, 2025; Chen et al., 2024b). All sources except Matplotlib are obtained from public Hugging Face releases; the Matplotlib examples are scraped from the official gallery, retaining only successfully rendered figures. For release, we document the exact source URI, license or access status, split policy, and redistribution status for each dataset in the accompanying license and provenance table.

Train/test splits. Native train/test splits are preserved when available (ChartQA, ChemVQA-2K, DocVQA, GEOQA_8K_R1V, GraphVQA-Swift, Geometry3K). The remaining sources are deterministically re-split with seed 42 to produce held-out pools. We sample 600 candidate test examples from these held-out pools. Each dataset receives a minimum quota to preserve small sources, with the remaining budget allocated proportionally to held-out pool size via the largest-remainder method. Training pools are not used for leaderboard evaluation and are reserved for self-training experiments.

Filtering. All candidate test examples are manually reviewed for suitability as reconstruction targets. A reviewer sees only the source image and accepts or rejects the example; examples are rejected if they are blank, corrupted, dominated by unreadable fine print, dependent on hidden metadata, or otherwise infeasible to reconstruct from visible structure alone. Rejected examples are not replaced. Of the 600 candidate examples, 539 are retained as the final Vision2Code test set.

Table 8: Per-dataset benchmark counts and split policy after manual filtering.

Dataset	Domain	Counts	Split policy
ChartQA	Charts & Plots	45	Native split
DVQA	Charts & Plots	39	Deterministic, seed 42
FigureQA	Charts & Plots	39	Deterministic, seed 42
Matplotlib	Charts & Plots	13	Deterministic, seed 42
Geoperception	Geometry	34	Deterministic, seed 42
GEOQA_8K_R1V	Geometry	21	Native split
Geometry3K	Geometry	21	Native split
OlympiadBench	Geometry	23	Deterministic, seed 42
Graph-Algorithms	Graphs	39	Deterministic, seed 42
GraphVQA-Swift	Graphs	42	Native split
ChemVQA-2K	Scientific Imagery	63	Native split
EEE-Bench	Scientific Imagery	22	Deterministic, seed 42
Physics	Scientific Imagery	31	Deterministic, seed 42
DocVQA	Documents	68	Native split
SpatialVLM-QA	3D Spatial Scenes	39	Deterministic, seed 42
Total	–	**539**	–

C Generation and Rendering Protocol

Generation prompt. All benchmarked models receive the same image-to-code instruction. For local Qwen models the prompt is applied through the model chat template with `enable_thinking=False`; local decoding is greedy (`do_sample=False`). API-hosted models use the same system and user content, with provider-specific message formatting only.

Code-generation system prompt

You are an expert Python plotting assistant. Given an input image, write Python code to recreate it using matplotlib/numpy as needed. Save the image only to the Python variable OUTPUT_PATH. Do not quote OUTPUT_PATH and do not append a file extension to it. Return code only.

Code-generation user prompt

Generate Python code to recreate this image. Return code only.

Code normalization. Model outputs are normalized before execution. We strip leading and trailing whitespace, remove `<think>` blocks when present, and extract the final Python code block when the model returns markdown fences. The normalized string is written as `generated_code.py`. No semantic repair is applied before the primary benchmark render.

Rendering environment. Generated code is executed in a temporary directory with `MPLBACKEND=Agg`. The evaluator defines `OUTPUT_PATH` and monkey-patches Matplotlib `savefig` calls so that the output is saved to the expected path at the same pixel dimensions as the source image. The renderer sets the figure size from the source width and height with target DPI 100, enforces `bbox_inches=None` and `pad_inches=0`, and times out executions after 30 seconds unless overridden by a run configuration. If code execution fails, exits without producing an image, or produces an unreadable image, the sample is counted as a render failure and receives benchmark score 0.0.

Software stack. The main local rendering environment used Python 3.10.19, Matplotlib 3.10.6, NumPy 2.2.6, Pillow 12.1.0, and PyTorch 2.10.0+cu128. Cluster launchers set Hugging Face, Torch, Triton, Matplotlib, and package caches to writable scratch directories to avoid shared-cache lock contention.

D GENERIC RUBRIC AND EMBEDDING BASELINES

D.1 GENERIC RUBRIC BASELINE

The generic rubric baseline uses the same VLM rater, image inputs, output schema, and render-failure rule as the dataset-specific evaluator. The only difference is the rubric: instead of using a source-dataset-specific rubric, every example is scored with the same four equal-weight categories as shown in Table 9. This baseline tests whether the improvement in human alignment comes from dataset-specific evaluation criteria rather than from the rater model alone. The evaluation prompts are shown below.

Generic-rubric system prompt

You are a rigorous but calibrated evaluator for image recreation quality. Compare the reference image against the candidate rendered image. Judge only what is visibly present in the images.

Use the rubric provided by the user message. It applies to every dataset. Be strict about semantically wrong recreations, but do not over-focus on small cosmetic differences.

Scoring anchors on a 0.0 to 5.0 scale: 5.0 = near-exact or clearly excellent; 4.0 = strong recreation with minor issues; 3.0 = good but with noticeable structural, text, or semantic problems; 2.0 = partial recreation with important errors; 1.0 = major mismatch; 0.0 = missing, unreadable, blank, or fundamentally broken.

Return JSON only. Do not include markdown fences.

Generic-rubric user prompt template

Evaluate the candidate recreation against the reference image using this rubric.

Rubric version: `generic_recreation_rubric_v1_equal_weight`

Rubric dataset: `generic`

Return JSON with exactly these top-level keys: `category_scores`, `rationales`, `strengths`, `issues`, and `overall_summary`.

The `category_scores` object must contain every category id with a numeric score from 0.0 to 5.0. The `rationales` object must contain the same category ids, with one short rationale per category. Use the same four generic categories for every dataset. Do not compute the final 0.0 to 5.0 rating yourself; provide category scores only.

Table 9: Rubric categories used for the generic-rubric evaluator.

Category	Weight	Meaning
Core information fidelity	0.25	Whether the rendered image preserves the main objects, values, symbols, and visual facts.
Structure and layout fidelity	0.25	Whether spatial arrangement, object relationships, axes, topology, or document organization are preserved.
Text and annotation accuracy	0.25	Whether labels, numbers, legends, symbols, and annotations are correct and legible.
Visual completeness and cleanliness	0.25	Whether the render is complete, non-degenerate, readable, and free from severe artifacts.

For the generic baseline, the final score is computed outside the rater as the weighted average of the four category scores. The same render-failure rule is used as in the main evaluator: if the generated code fails to execute or does not produce a valid image, the final score is set to 0.0. Unlike the dataset-specific evaluator, the generic baseline does not use source-dataset-specific critical-category caps or domain-specific guardrails.

D.2 Embedding Similarity baseline

We also compare against embedding-similarity baselines using Qwen3-VL-Embedding-8B. These baselines do not use a VLM rater or rubric. Instead, they embed the source image and rendered output and compute cosine similarity between the two embeddings. We evaluate two variants. The *image-only* variant embeds each image directly. The *image+text* variant embeds each image together with a short dataset-specific focus text that describes the visual information that should matter for that source dataset. The same focus text is used for the source image and rendered output of a given example. Appendix D.2.1 lists the focus texts. For both embedding variants, the common embedding instruction is: "Represent the input for comparing the visual fidelity of a recreated figure to its source."

D.2.1 Embedding Focus Texts

For the image+text cosine-similarity validation metric, each source-render pair is embedded with the same dataset-specific focus text. These short prompts specify the visual attributes that should dominate the embedding comparison for each dataset.

Table 10: Dataset-specific focus texts used for the image+text cosine-similarity validation metric.

Dataset	Focus text
ChartQA	chart type, plotted values, axis labels, legend entries, overall layout.
DVQA	chart structure, plotted values, readable labels, legend entries, information content.
FigureQA	chart structure, plotted trends, labels, legend entries, visual relationships.
Matplotlib	plot layout, data traces, axes, titles, legends, annotations, figure styling.
Geoperception	geometry structure, labels, markers, relationships, annotation placement.
GEOQA_8K_R1V	geometry structure, point labels, lengths, angles, markers, annotation placement.
Geometry3K	geometry shapes, point labels, markers, numerical annotations, exact relations.
Graph-Algorithms	graph topology, node labels, edge directions, edge weights, ordering.
GraphVQA-Swift	graph structure, node labels, edge labels, attributes, label binding.
ChemVQA-2K	molecular structure, chemical symbols, bond lines, annotations, color-coded semantics.
EEE-Bench	circuit topology, component identity, labels, values, symbols, current or voltage direction.
Physics	object placement, vectors, labels, numerical givens, physically correct structure.
OlympiadBench	geometry structure, labels, givens, markers, precise relationships.
DocVQA	text accuracy, document layout, table structure, labels, visual organization.
SpatialVLM-QA	object identity, spatial relations, depth cues, layout, scene completeness.

E Human Validation

We evaluate human alignment on 180 source–output pairs from test. The set is balanced across six models and all 15 datasets, with 30 outputs per model, and 12 per dataset. The six models are Qwen3.5-9B, Kimi-K2.5, Qwen3.5-397B-A17B, GPT-5.4, GPT-5.4 Mini, and Gemini-3.1-Flash-LitePreview. Four annotators independently rate all 180 pairs, yielding 720 ratings. Model and dataset identities are hidden, and examples are shown in a different random order for each annotator. Annotators compare the source and rendered image and rate reconstruction quality from 0–5, considering structure, layout, shapes, positions, text, labels, and missing or incorrect elements while ignoring minor cosmetic differences. The scale ranges from 5 (almost identical) to 0 (no meaningful match). Figures 5 and 6 show the annotation interface.

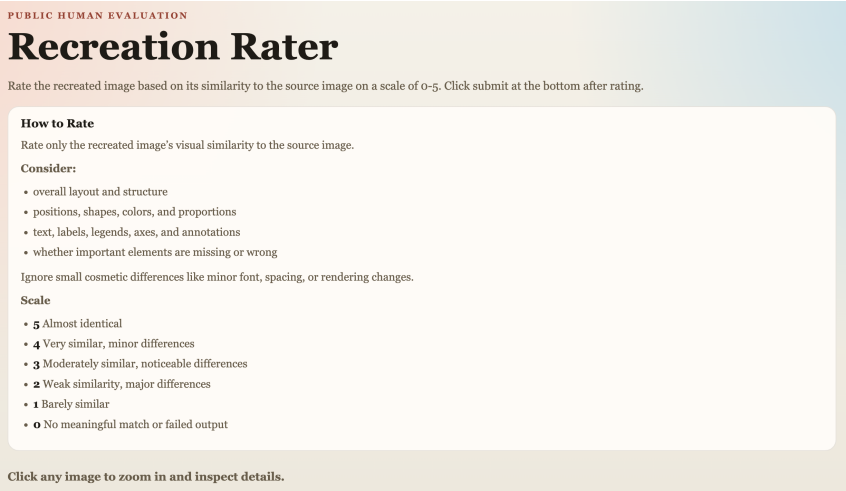


Figure 5: Rater web interface: general rating instructions shown to human annotators.

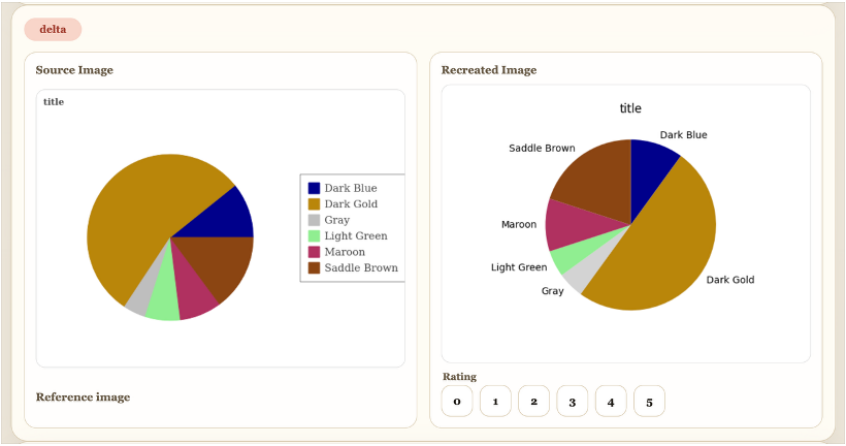


Figure 6: Rater web interface: source image (left) and candidate render (right) with the 0–5 rating controls.

E.1 Inter-Annotator Agreement

Because all four annotators rate the same examples, we use a two-way random-effects, absolute-agreement ICC. ICC(2,1) measures the reliability of one annotator, while ICC(2,4) measures the reliability of the four-rater mean used in our correlation analysis. Confidence intervals are computed by bootstrapping the 180 examples. Pairwise Pearson correlations range from 0.754 to 0.879, Spearman correlations from 0.760 to 0.869, and quadratic-weighted Cohen's κ from 0.619 to 0.863. The high ICC(2,4) supports using the mean of the four ratings as the human target.

Table 11: Inter-annotator agreement across four human annotators.

Measure	Result	95% CI
ICC(2,1), single annotator	0.753	[0.701, 0.795]
ICC(2,4), four-rater mean	0.924	[0.904, 0.939]
All scores within 1 point	63.3%	–
All scores within 2 points	91.7%	–

E.2 Correlation Analysis

We compute Pearson and Spearman correlations between each automatic metric and the mean of the four human ratings over the same 180 examples. All metrics are available for all examples, so no values are removed or imputed. We compare SSIM, PSNR, CLIP, DreamSim, Qwen3-VL embedding similarity with and without dataset-specific focus text, and both generic and dataset-specific rubrics evaluated with Qwen3.5-122B and Kimi-K2.6. As reported in Table 3(a), the dataset-specific rubric achieves the highest correlation with human judgments under both VLM judges. It also consistently improves over the corresponding generic rubric, showing that the gain is not only due to using a VLM judge but also to the dataset-specific evaluation criteria.

F Evaluator-Guided Self-Training Details

For each self-training source image I_0, the model first generates code C_1, whose render is I_1. The model is then given I_1 and generates code again, producing a second render I_2. The evaluator assigns R_1 to (I_0, I_1) and R_2 to (I_1, I_2). Thus, R_1 measures fidelity to the source, while R_2 measures whether the model can consistently re-encode its own rendered output. We fine-tune on (I_1, C_1) pairs: high-R_1 samples provide code that reconstructs the source accurately, while $R_2 < R_1$ identifies visual structures that the model produces once but cannot reconstruct consistently.

We generate separate sets of 20,000 trajectories for Qwen3.5-9B and Phi-4 Multimodal using the same source-dataset training cohort, which is disjoint from the 539-example Vision2Code test set. For Qwen3.5-9B, 12,973 trajectories have valid paired ratings; 5,215 are missing a paired rating and 1,812 have a non-ok or missing rating status. For Phi-4, 9,199 trajectories have valid paired ratings; 6,761 are missing a paired rating and 4,040 have a non-ok or missing rating status.

We set $\alpha = 4$ and construct five filtered datasets for each model. To compare filters using equal training-set sizes within each model, we match all variants to the size of the smallest filter. This gives 1,412 examples per Qwen variant, split into 1,271 training and 141 development examples, and 122 examples per Phi-4 variant, split into 110 training and 12 development examples. Table 12 reports the available candidates and the resulting dataset sizes.

Table 12: Self-training dataset filters. Variants are sampled with seed 42 from valid stage-paired trajectories and matched in size within each model.

Variant	Filter	Candidates	Train	Dev
Qwen3.5-9B				
All valid rated samples	Any valid paired (R_1, R_2)	12,973	1,271	141
Self-consistent high	$R_1 \geq \alpha,\ R_2 \geq R_1$	1,808	1,271	141
Self-drop high	$R_1 \geq \alpha,\ R_2 < R_1$	1,412	1,271	141
Low first rating	$R_1 < \alpha$	9,753	1,271	141
All high first rating	$R_1 \geq \alpha$	3,220	1,271	141
Phi-4 Multimodal				
All valid rated samples	Any valid paired (R_1, R_2)	9,199	110	12
Self-consistent high	$R_1 \geq \alpha,\ R_2 \geq R_1$	541	110	12
Self-drop high	$R_1 \geq \alpha,\ R_2 < R_1$	122	110	12
Low first rating	$R_1 < \alpha$	8,536	110	12
All high first rating	$R_1 \geq \alpha$	663	110	12

Fine-tuning configuration. Each variant fine-tunes its corresponding base checkpoint using full fine-tuning with DeepSpeed ZeRO-3 on four A5000 GPUs. Both Qwen3.5-9B and Phi-4 use a maximum sequence length of 2,048, 6 epochs, a learning rate of 5×10^{-6}, per-device training and evaluation batch sizes of 1, evaluation and checkpointing every 50 steps, and seed 42. Qwen uses gradient accumulation 8 and logs every 10 steps. Because each Phi-4 variant contains only 110 training examples, Phi-4 uses gradient accumulation 1 and logs every 5 steps. For each variant, we select the checkpoint with the lowest development loss.

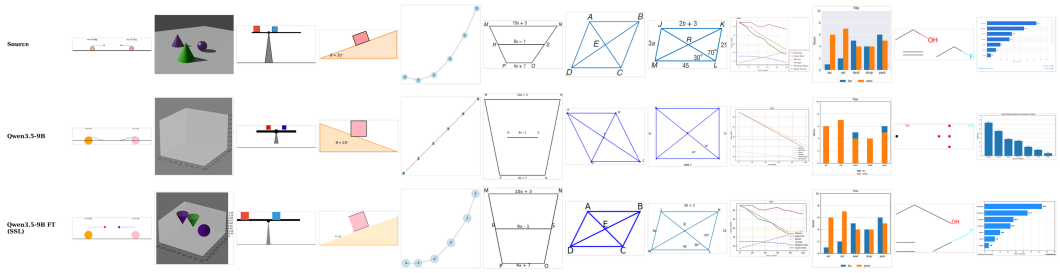


Figure 7: Qualitative examples before and after rater-filtered self-training. Rows show source images, off-the-shelf Qwen3.5-9B renders, and renders after fine-tuning on the $(R_1 \geq \alpha,\ R_2 < R_1)$ subset.

Figure 7 provides a qualitative check on the self-training result. The fine-tuned model often improves visible structure rather than only changing style: examples show more complete layouts, better object placement, and fewer collapsed or missing elements compared with the base Qwen3.5-9B outputs.

Independent evaluation. Training-data selection uses Qwen3.5-122B as the evaluator. To check whether the measured improvement is specific to this judge, we separately rescore the baseline and best-filter models with Kimi-K2.6 and Qwen3.6-Plus. Neither additional judge is used for selecting the self-training data.

G Failure Taxonomy

Render failures are separated from semantic reconstruction errors. Render failures receive score 0.0 because no valid candidate image is available for the rubric evaluator. We classify failures using the execution status and traceback in Table 13.

Table 13: Execution failure taxonomy used in render-success diagnostics.

Failure type	Definition	Typical cause
Syntax/truncation	Code cannot be parsed or stops mid-expression.	Output length limit, unterminated strings, unmatched brackets.
Missing dependency or file	Code imports unavailable packages or reads absent local files.	Assumed external assets, unavailable plotting libraries.
No image saved	Code exits without producing the expected output file.	Missing save call, saving to another path, interactive-only display.
Hallucinated API	Code calls nonexistent methods or unsupported keyword arguments.	Invented Matplotlib helpers, invalid patch/axes methods.
Shape/3D error	Runtime failure from incompatible array shapes or invalid 3D geometry calls.	Incorrect coordinate dimensionality, wrong patch type for 3D axes.
Other runtime error	Any execution failure not covered above.	Type errors, value errors, or unhandled corner cases.

H Additional Analyses

Test-time scaling. We also test whether repeated refinement over four stages improves reconstruction at inference time. At stage 1, the model receives only the source image. At later stages, it receives the source image together with its previous code and rendered output, and is asked to improve the reconstruction. For Qwen3.5-9B, the first refinement gives a small gain, but later stages do not improve

Table 14: Test-time scaling on test across four inference stages.

Stage	Qwen3.5-9B	GPT-5.4 Mini
1	1.60	3.85
2	1.63	3.84
3	1.61	3.83
4	1.63	3.81

further. For GPT-5.4 Mini, repeated refinement slightly reduces performance. One possible reason is that after the first pass, the main challenge is no longer generating code but identifying exactly what is wrong with the rendered image. Without a separate verifier, the model has to judge its own mistakes and may keep modifying an already incorrect structure.

Stage 1 uses the standard image-to-code prompt. For stages 2–4, the model receives the original reference image, the previous rendered output, and the previous Python code, and is asked to rewrite the code so that the rendered output better matches the reference.

Refinement system prompt

You are an expert Python plotting assistant. Write Python code to recreate the reference image using matplotlib/numpy as needed. Save the image only to the Python variable OUTPUT_PATH. Do not quote OUTPUT_PATH and do not append a file extension to it. Return code only.

Refinement user prompt

You are improving Python code that recreates a reference image.

The first image is the reference image to recreate. The second image is the previous rendered output.

Previous Python code: {`previous_code`}

Rewrite the Python code so the rendered output better matches the reference image. Save the image only to the Python variable OUTPUT_PATH. Do not quote OUTPUT_PATH and do not append a file extension to it. Return code only.

Code format. Vision2Code does not require a specific programming language or rendering tool; it only requires an executable artifact that can be rendered into an image. To test this, we rerun GPT-5.4 Mini on a separate 292-example DocVQA held-out set with LaTeX instead of the default Python/Matplotlib target, then compile and rasterize the LaTeX output before applying the same DocVQA rater and rubric. LaTeX performs worse in both render success and reconstruction score, likely because many DocVQA pages contain scanned forms, dense tables, and irregular layouts that are difficult to reproduce with LaTeX. We also test Excalidraw on a small set of 48 web-scraped examples, where it reaches 100% render success and a generic-rubric score of 3.90. Appendix M shows qualitative reconstruction examples for LaTeX and Excalidraw.

Table 15: Code format ablation on a separate 292-example DocVQA held-out set for GPT-5.4 Mini.

Target	Render success	Score
Python	84.9%	3.35
LaTeX	65.8%	1.52

I RECONSTRUCTION FAILURES

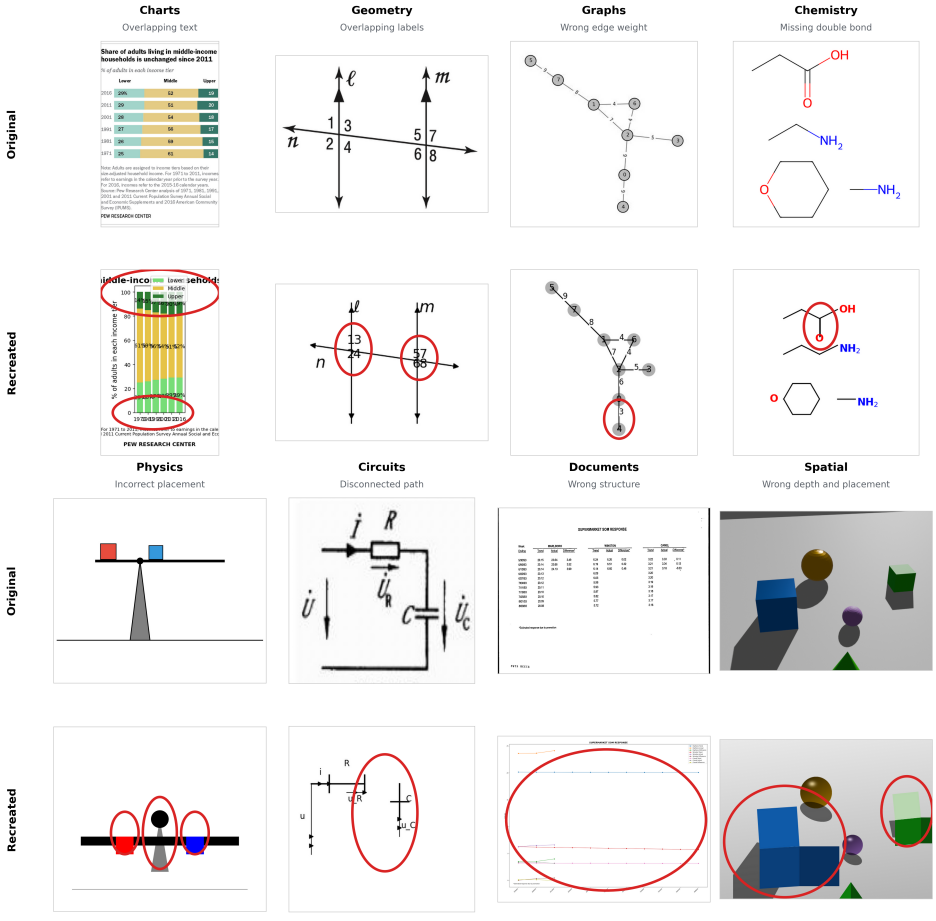


Figure 8: Representative reconstruction failures. Each original image is shown above its reconstruction, with the principal error circled in red. The examples show overlapping chart text, overlapping geometry labels, an incorrect graph edge weight, a missing chemical double bond, incorrect object placement in a physics diagram, a disconnected circuit, a document table rendered as a plot, and incorrect depth and relative placement in a spatial scene.

J RECREATION COMPARISONS

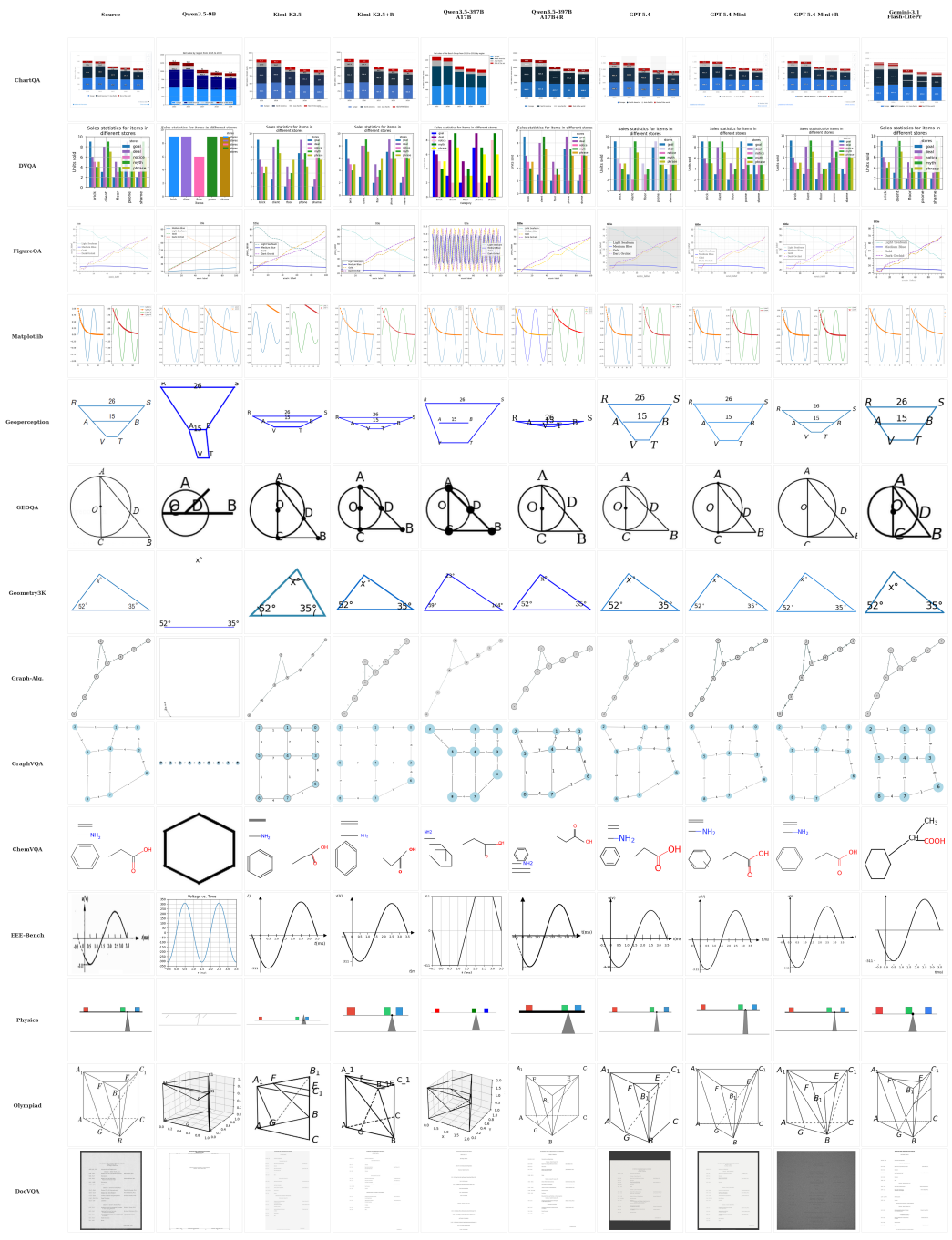


Figure 9: Source and rendered-output comparisons for the nine benchmarked models.

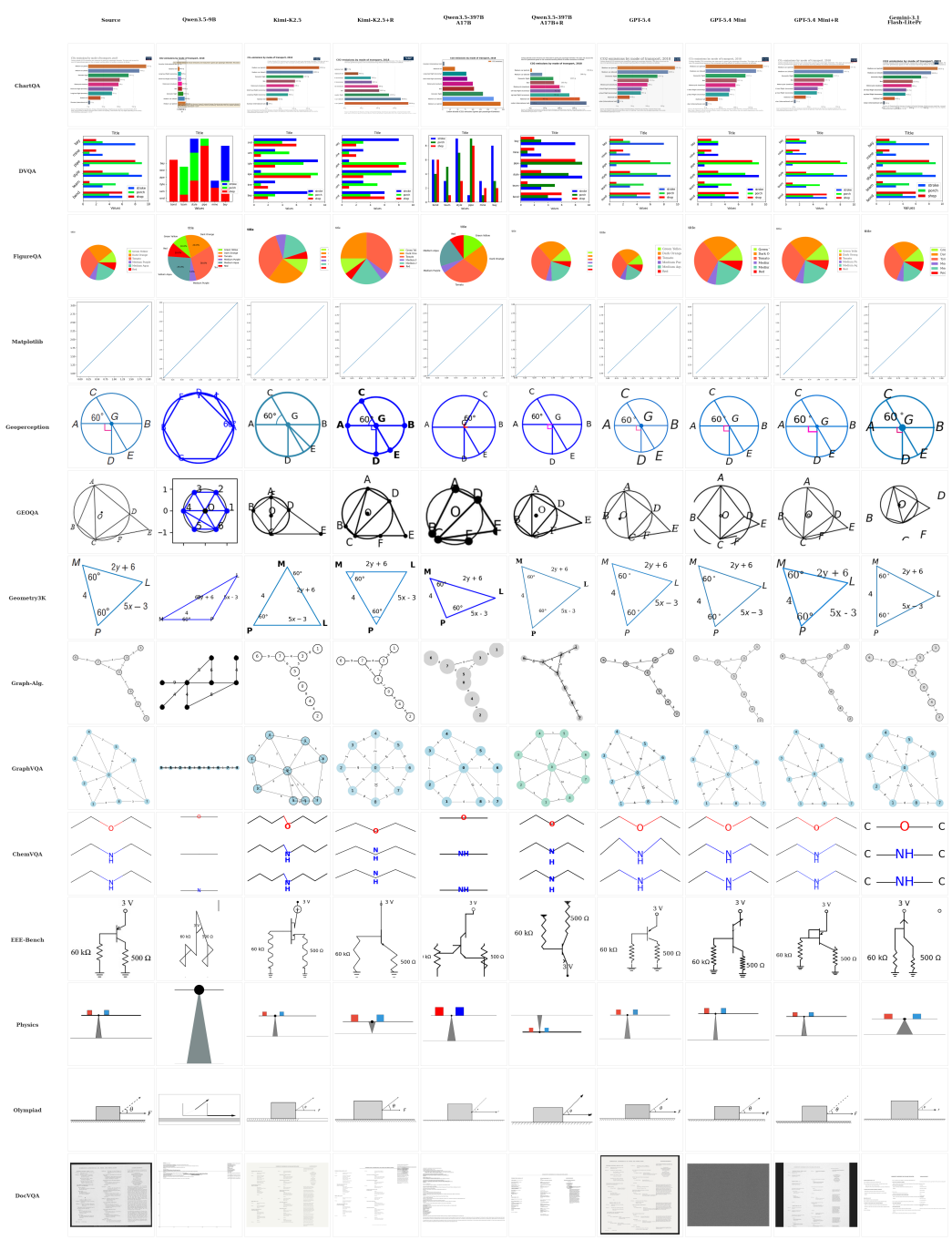


Figure 10: Additional source and rendered-output comparisons for the nine benchmarked models.

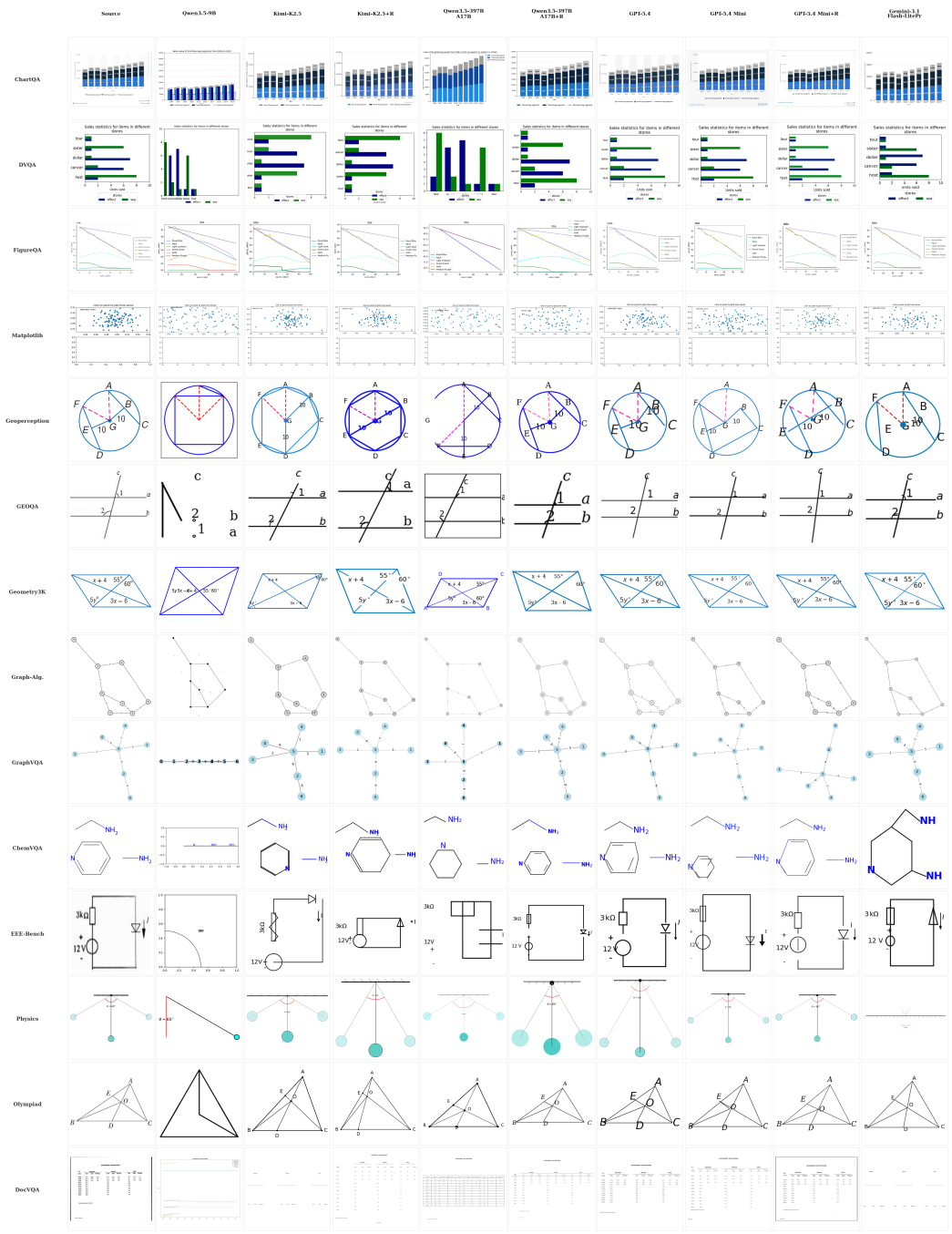


Figure 11: Additional source and rendered-output comparisons for the nine benchmarked models.

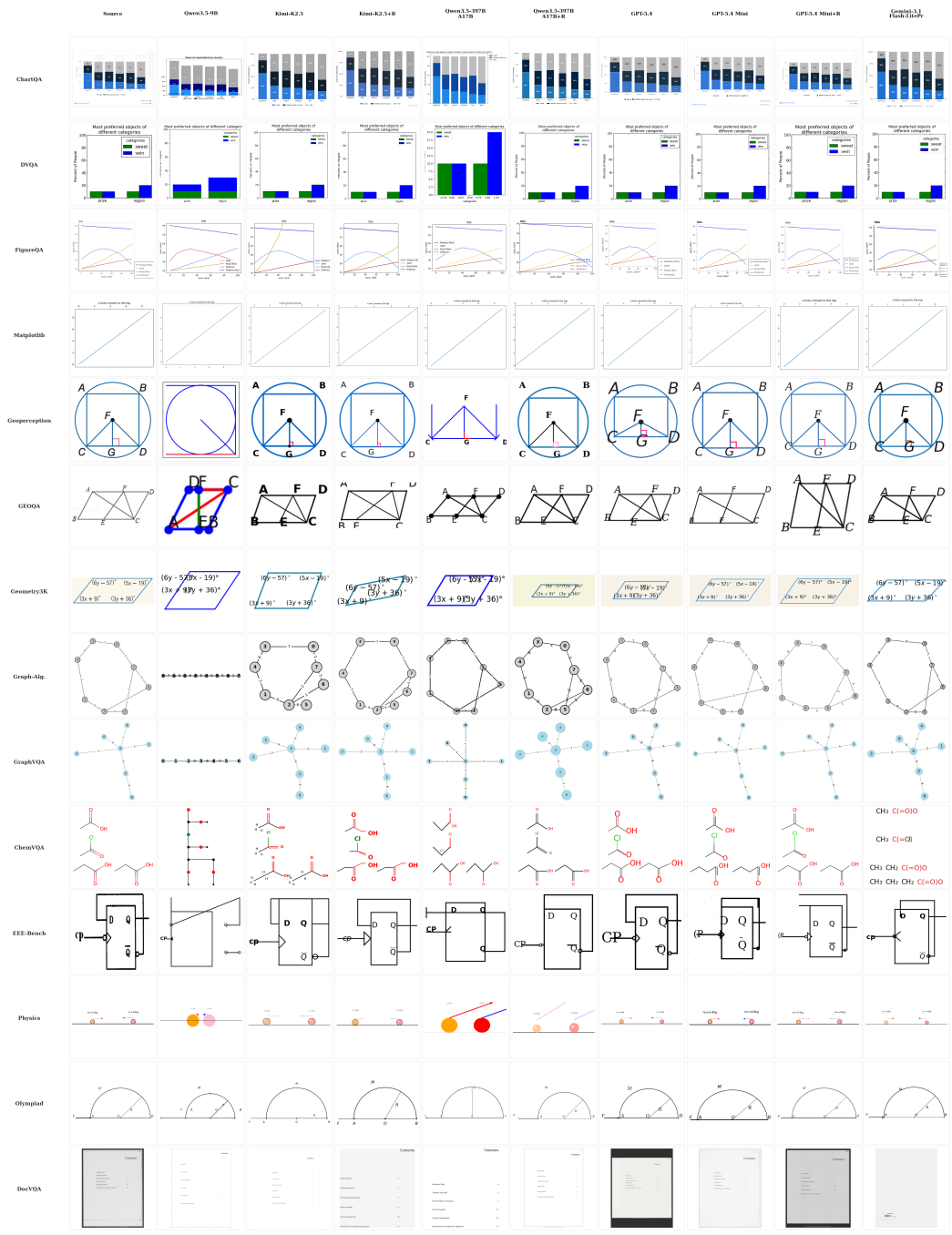


Figure 12: Additional source and rendered-output comparisons for the nine benchmarked models.

K QUALITATIVE RATING EXAMPLES

We include representative test examples showing the source image, model-rendered outputs, and the score assigned by the Qwen3.5-122B-A10B-GPTQ-Int4 rater. These examples illustrate how the rater distinguishes high-fidelity recreations from outputs with missing structure, incorrect visual elements, or degraded layout quality.

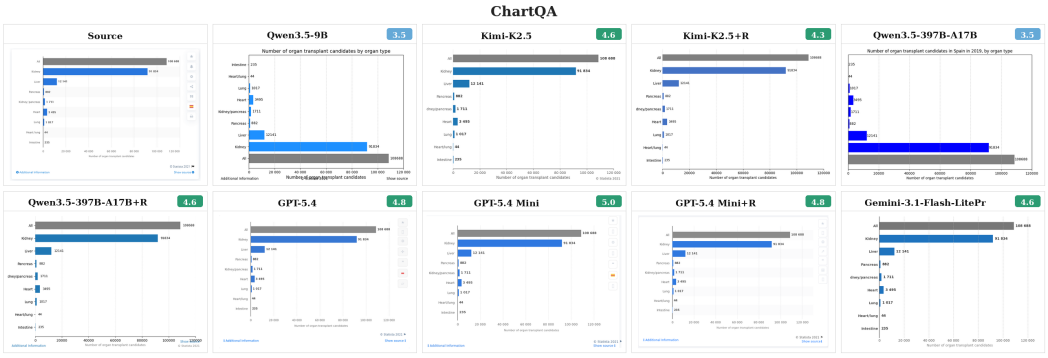


Figure 13: ChartQA rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

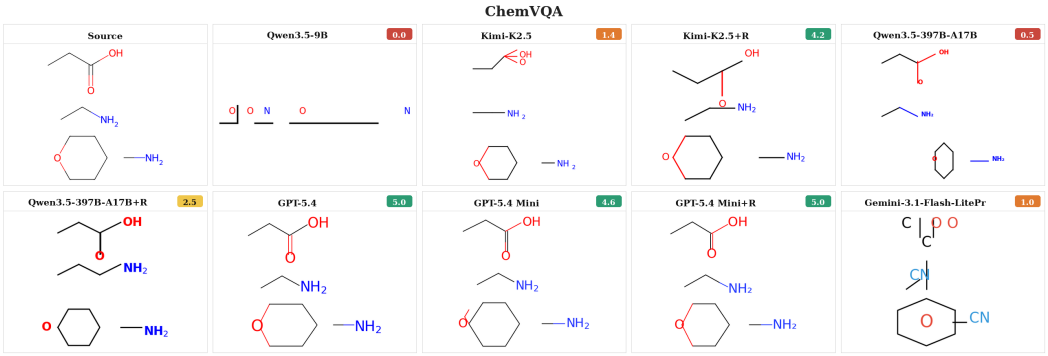


Figure 14: ChemVQA-2K rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

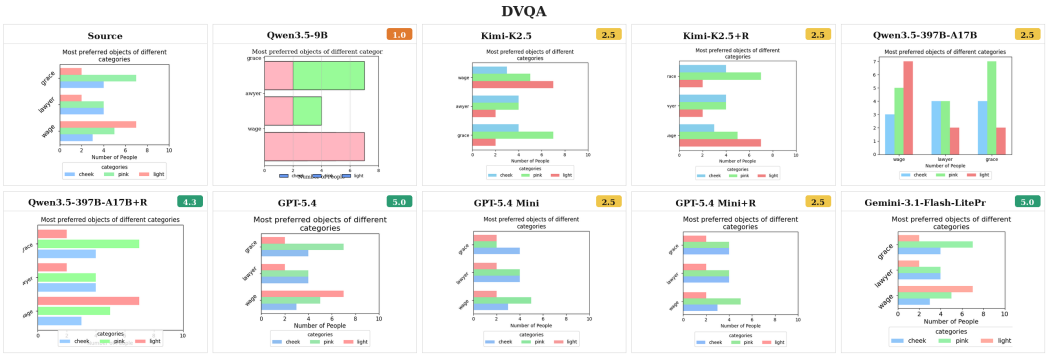


Figure 15: DVQA rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

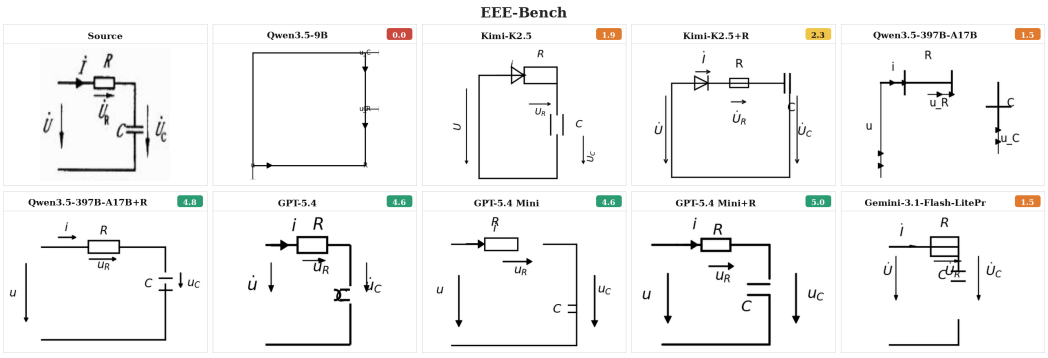


Figure 16: EEE-Bench rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

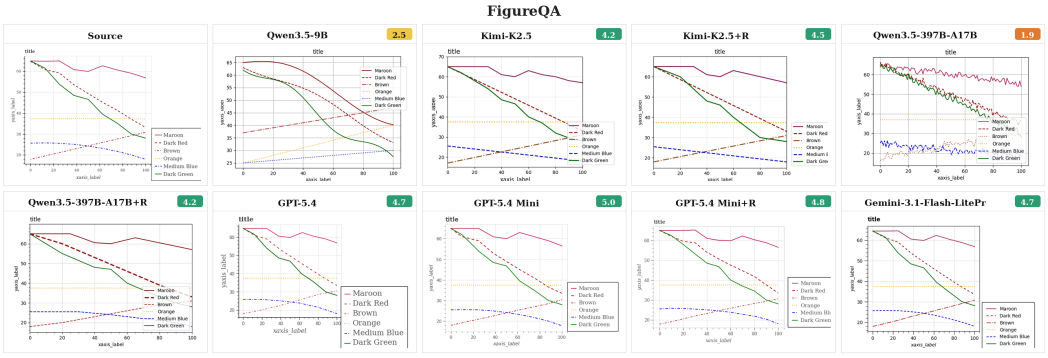


Figure 17: FigureQA rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

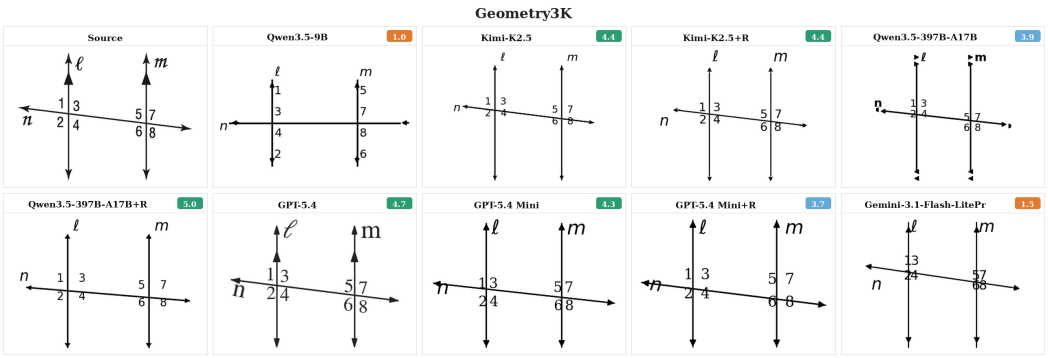


Figure 18: Geometry3K rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

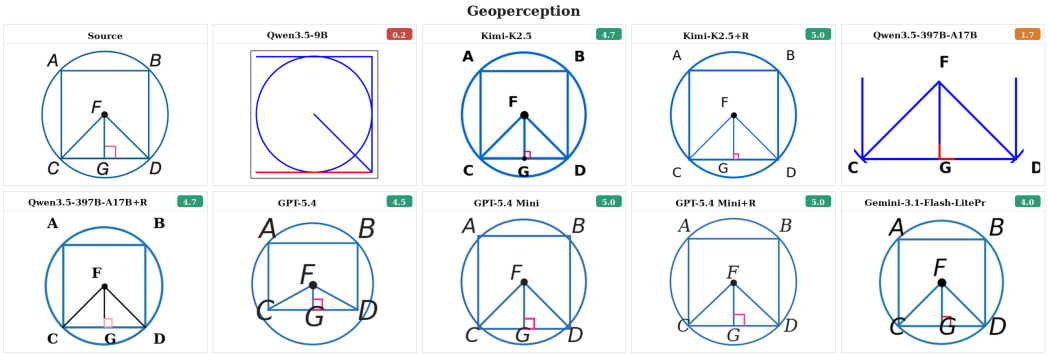


Figure 19: Geoperception rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

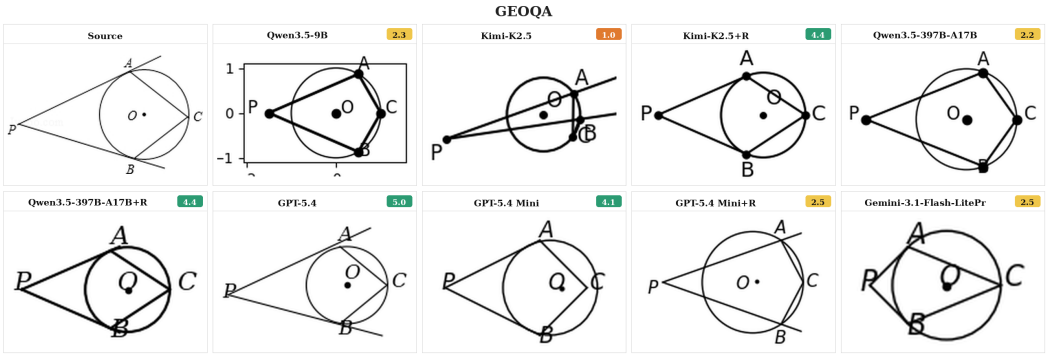


Figure 20: GeoQA-8K rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

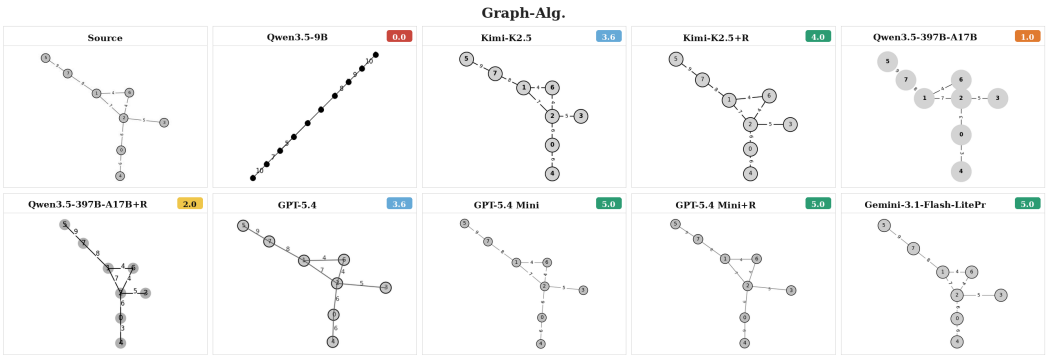


Figure 21: Graph Algorithms rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

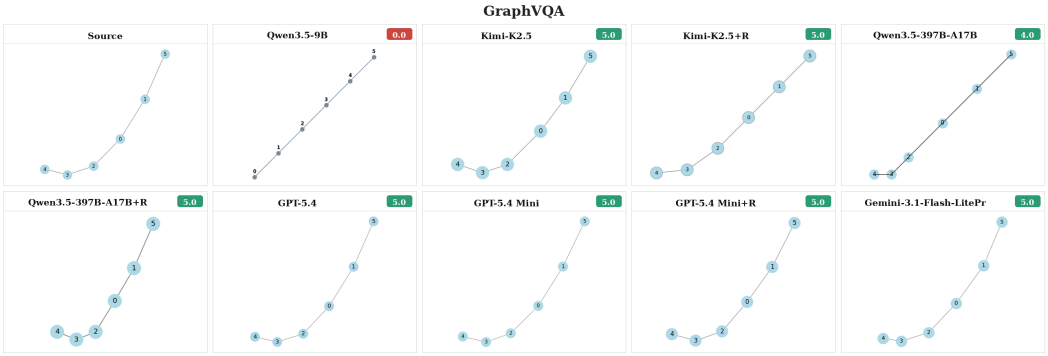


Figure 22: GraphVQA-Swift rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

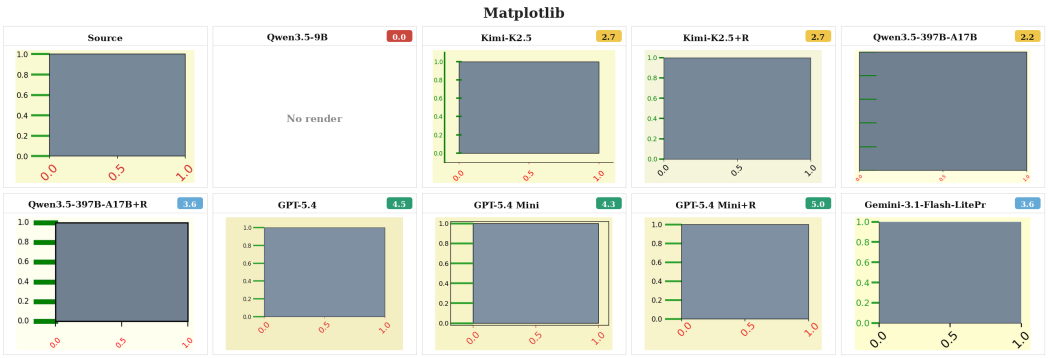


Figure 23: Matplotlib rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

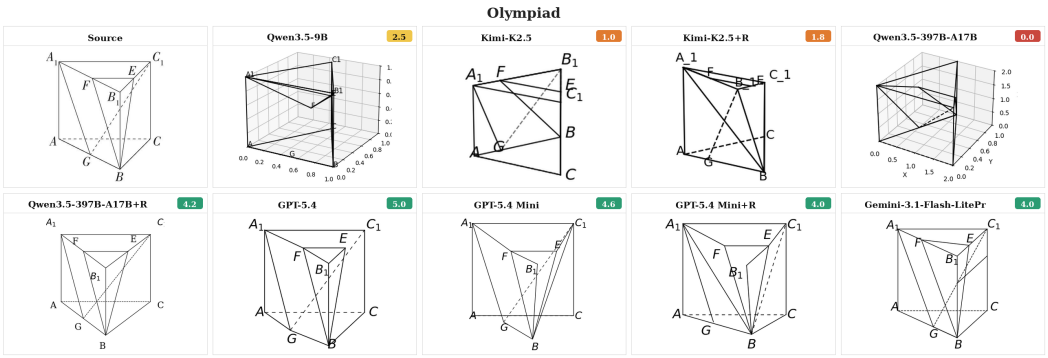


Figure 24: OlympiadBench rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

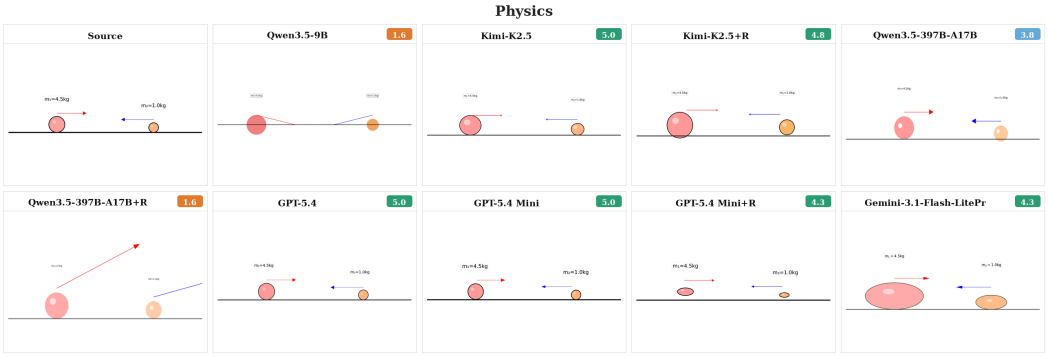


Figure 25: Physics rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

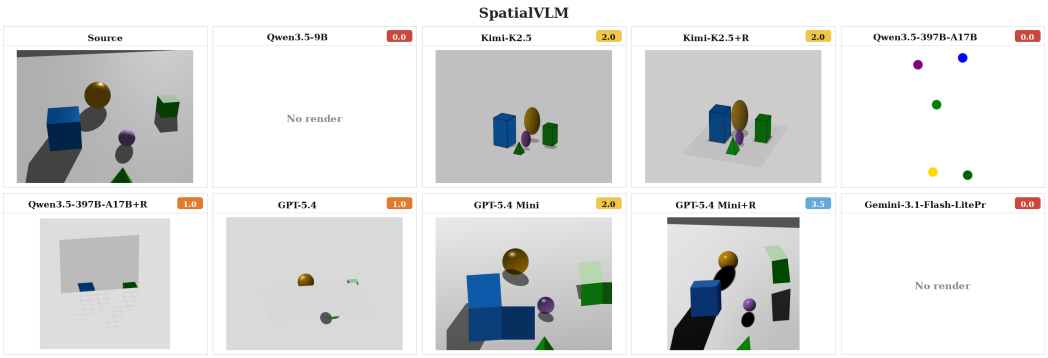


Figure 26: SpatialVLM-QA rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

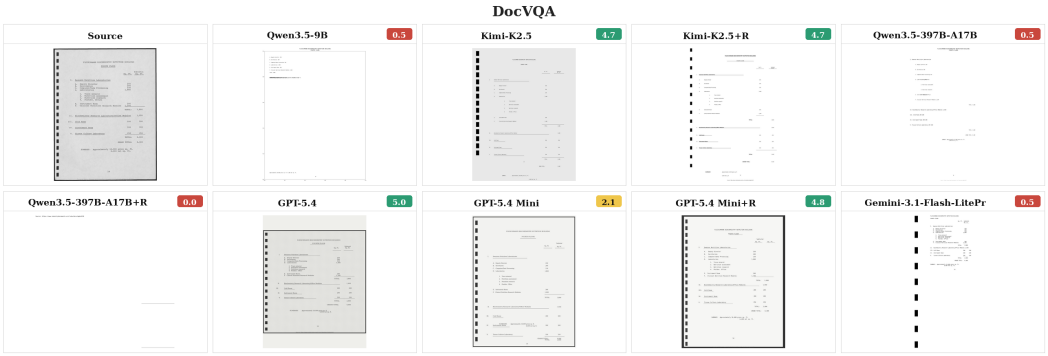


Figure 27: DocVQA rated renders by the Qwen3.5-122B-A10B-GPTQ-Int4 rater on test examples.

L Evaluator Robustness Qualitative Examples

Figures 28 and 29 show controlled perturbations evaluated by Qwen3.5-122B and Kimi-K2.6, respectively. Each perturbation introduces one targeted error while leaving the remaining image unchanged. The displayed values are the final rubric scores.

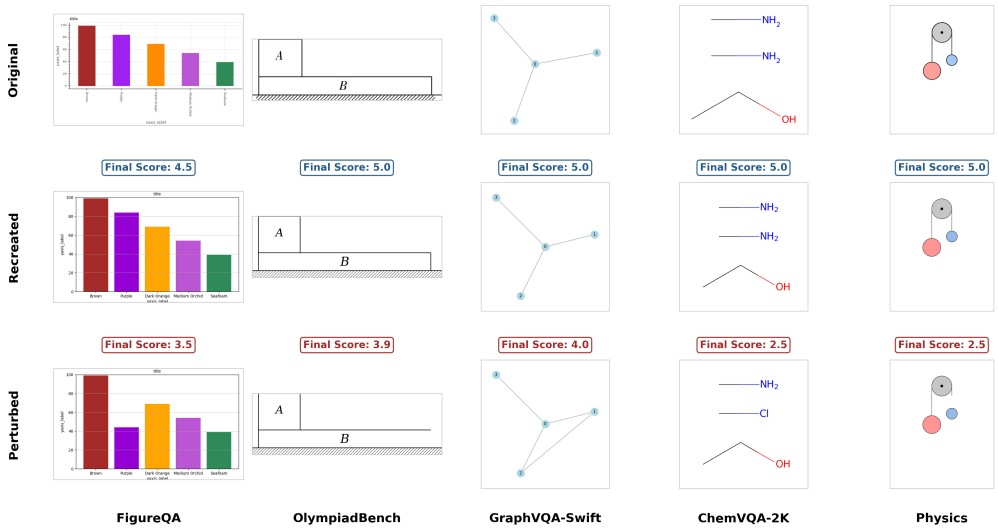


Figure 28: Qualitative examples evaluated by Qwen3.5-122B. The rows show the source image, its reconstruction, and the perturbed reconstruction. The targeted category scores decrease as follows: FigureQA, Numeric Information Fidelity ($5.0 \rightarrow 4.0$); OlympiadBench, Readability and Completeness ($5.0 \rightarrow 3.0$); GraphVQA-Swift, Graph Structure Fidelity ($5.0 \rightarrow 2.0$); ChemVQA-2K, Chemical Symbol Accuracy ($5.0 \rightarrow 2.0$); and Physics, Physical Setup Fidelity ($5.0 \rightarrow 2.0$).

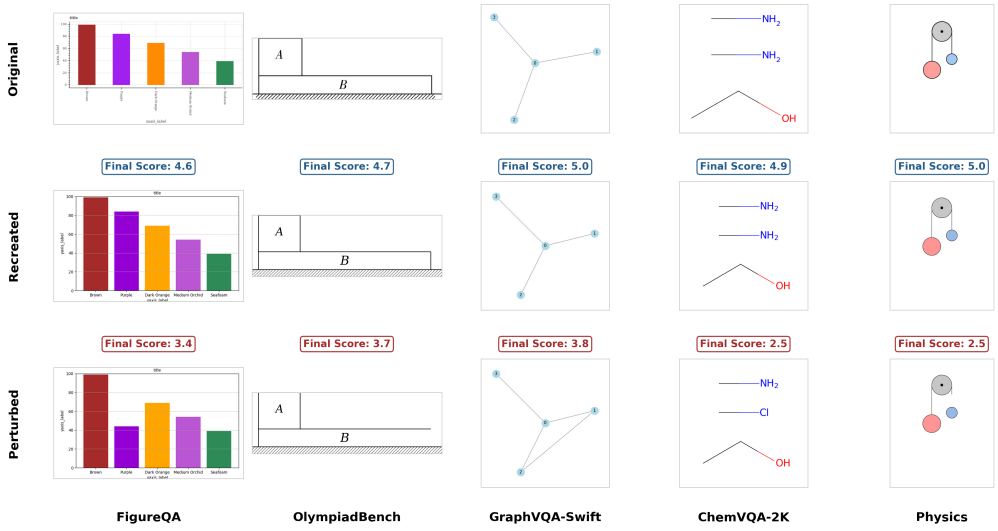


Figure 29: The same controlled perturbations evaluated by Kimi-K2.6. The targeted category scores decrease as follows: FigureQA, Numeric Information Fidelity ($5.0 \rightarrow 2.5$); OlympiadBench, Readability and Completeness ($4.6 \rightarrow 3.5$); GraphVQA-Swift, Graph Structure Fidelity ($5.0 \rightarrow 2.0$); ChemVQA-2K, Chemical Symbol Accuracy ($5.0 \rightarrow 2.0$); and Physics, Physical Setup Fidelity ($5.0 \rightarrow 2.5$).

M TOOL-USE ABLATION QUALITATIVE EXAMPLES

M.1 EXCALIDRAW RECREATION

Excalidraw illustrates a practical Vision2Code setting: the model predicts an editable structured artifact, which is rendered back to an image.

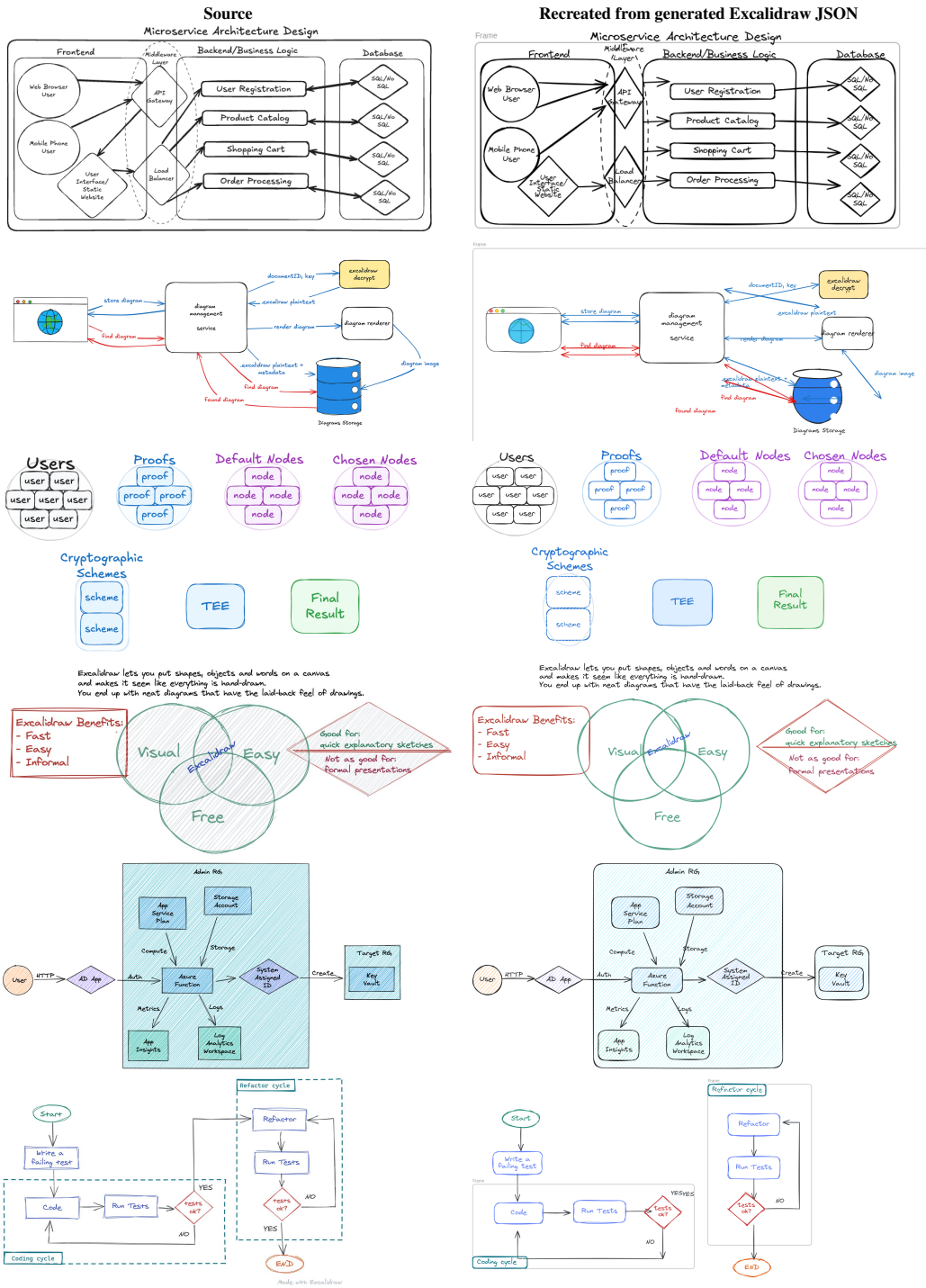
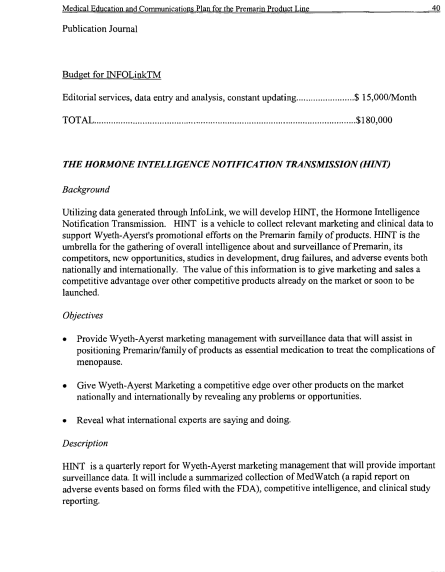


Figure 30: Qualitative examples from the Excalidraw tool-use ablation. The model writes an Excalidraw scene JSON object, which is rendered with the official Excalidraw renderer.

M.2 LaTeX Document Recreation

Source



Medical Education and Communications Plan for the Premarin Product Line 40

Publication Journal

Budget for INFOLinkTM

Editorial services, data entry and analysis, constant updating.....$ 15,000/Month

TOTAL.....$180,000

THE HORMONE INTELLIGENCE NOTIFICATION TRANSMISSION (HINT)

Background

Utilizing data generated through InfoLink, we will develop HINT, the Hormone Intelligence Notification Transmission. HINT is a vehicle to collect relevant marketing and clinical data to support Wyeth-Ayerst's promotional efforts on the Premarin family of products. HINT is the umbrella for the gathering of overall intelligence about and surveillance of Premarin, its competitors, new opportunities, studies in development, drug failures, and adverse events both nationally and internationally. The value of this information is to give marketing and sales a competitive advantage over other competitive products already on the market or soon to be launched.

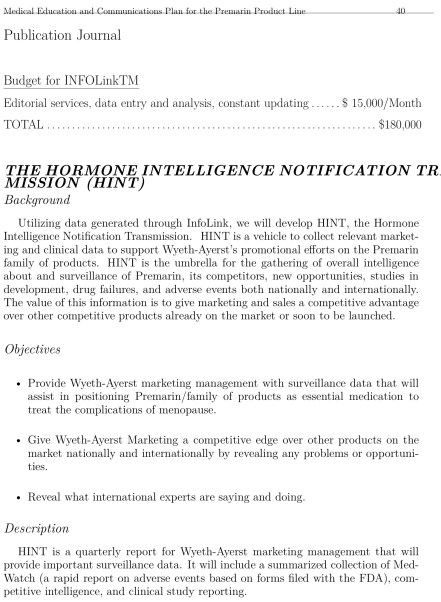
Objectives

- Provide Wyeth-Ayerst marketing management with surveillance data that will assist in positioning Premarin/family of products as essential medication to treat the complications of menopause.
- Give Wyeth-Ayerst Marketing a competitive edge over other products on the market nationally and internationally by revealing any problems or opportunities.
- Reveal what international experts are saying and doing.

Description

HINT is a quarterly report for Wyeth-Ayerst marketing management that will provide important surveillance data. It will include a summarized collection of MedWatch (a rapid report on adverse events based on forms filed with the FDA), competitive intelligence, and clinical study reporting.

Recreated from generated LaTeX



Medical Education and Communications Plan for the Premarin Product Line 40

Publication Journal

Budget for INFOLinkTM

Editorial services, data entry and analysis, constant updating ..... $ 15,000/Month

TOTAL $180,000

THE HORMONE INTELLIGENCE NOTIFICATION TRANSMISSION (HINT)

Background

Utilizing data generated through InfoLink, we will develop HINT, the Hormone Intelligence Notification Transmission. HINT is a vehicle to collect relevant marketing and clinical data to support Wyeth-Ayerst's promotional efforts on the Premarin family of products. HINT is the umbrella for the gathering of overall intelligence about and surveillance of Premarin, its competitors, new opportunities, studies in development, drug failures, and adverse events both nationally and internationally. The value of this information is to give marketing and sales a competitive advantage over other competitive products already on the market or soon to be launched.

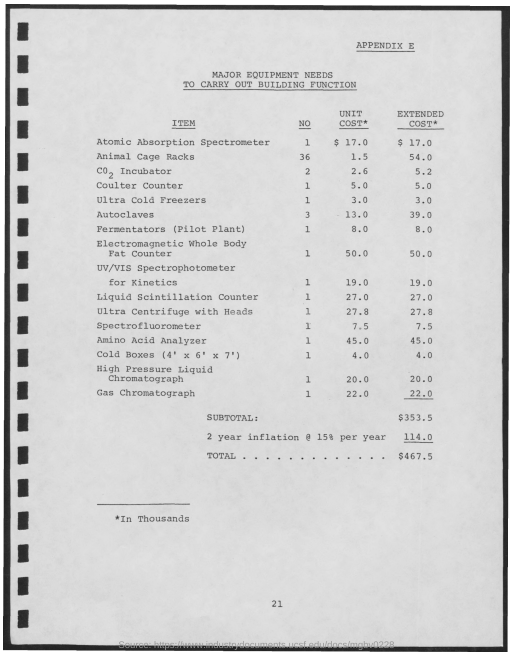
Objectives

- Provide Wyeth-Ayerst marketing management with surveillance data that will assist in positioning Premarin/family of products as essential medication to treat the complications of menopause.
- Give Wyeth-Ayerst Marketing a competitive edge over other products on the market nationally and internationally by revealing any problems or opportunities.
- Reveal what international experts are saying and doing.

Description

HINT is a quarterly report for Wyeth-Ayerst marketing management that will provide important surveillance data. It will include a summarized collection of MedWatch (a rapid report on adverse events based on forms filed with the FDA), competitive intelligence, and clinical study reporting.

Source: https://www.industrydocuments.ucsf.edu/docs/sjbw0217 DWRITE 065803



APPENDIX E

MAJOR EQUIPMENT NEEDS
TO CARRY OUT BUILDING FUNCTION

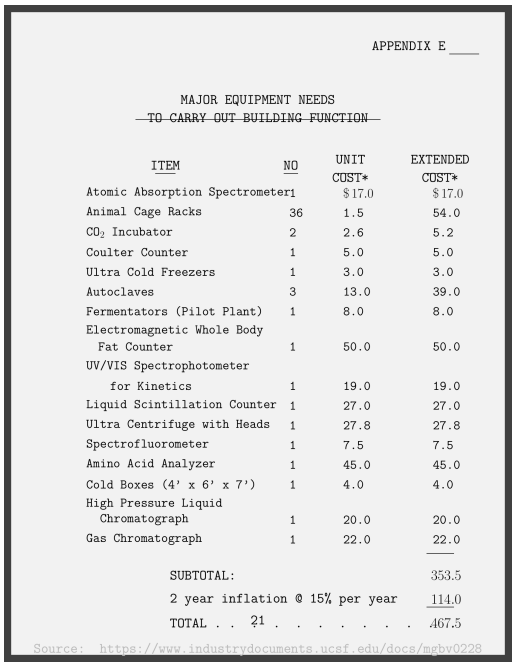
ITEM	NO	UNIT COST*	EXTENDED COST*
Atomic Absorption Spectrometer	1	$ 17.0	$ 17.0
Animal Cage Racks	36	1.5	54.0
CO_2 Incubator	2	2.6	5.2
Coulter Counter	1	5.0	5.0
Ultra Cold Freezers	1	3.0	3.0
Autoclaves	3	13.0	39.0
Fermentators (Pilot Plant)	1	8.0	8.0
Electromagnetic Whole Body Fat Counter	1	50.0	50.0
UV/VIS Spectrophotometer for Kinetics	1	19.0	19.0
Liquid Scintillation Counter	1	27.0	27.0
Ultra Centrifuge with Heads	1	27.8	27.8
Spectrofluorometer	1	7.5	7.5
Amino Acid Analyzer	1	45.0	45.0
Cold Boxes (4' x 6' x 7')	1	4.0	4.0
High Pressure Liquid Chromatograph	1	20.0	20.0
Gas Chromatograph	1	22.0	22.0
SUBTOTAL:			$353.5
2 year inflation @ 15% per year			114.0
TOTAL			$467.5

*In Thousands

21

Source: https://www.industrydocuments.ucsf.edu/docs/mgby0228

Source: https://www.industrydocuments.ucsf.edu/docs/ DW RITE 065



APPENDIX E

MAJOR EQUIPMENT NEEDS
TO CARRY OUT BUILDING FUNCTION

ITEM	NO	UNIT COST*	EXTENDED COST*
Atomic Absorption Spectrometer	1	$17.0	$17.0
Animal Cage Racks	36	1.5	54.0
CO_2 Incubator	2	2.6	5.2
Coulter Counter	1	5.0	5.0
Ultra Cold Freezers	1	3.0	3.0
Autoclaves	3	13.0	39.0
Fermentators (Pilot Plant)	1	8.0	8.0
Electromagnetic Whole Body Fat Counter	1	50.0	50.0
UV/VIS Spectrophotometer for Kinetics	1	19.0	19.0
Liquid Scintillation Counter	1	27.0	27.0
Ultra Centrifuge with Heads	1	27.8	27.8
Spectrofluorometer	1	7.5	7.5
Amino Acid Analyzer	1	45.0	45.0
Cold Boxes (4' x 6' x 7')	1	4.0	4.0
High Pressure Liquid Chromatograph	1	20.0	20.0
Gas Chromatograph	1	22.0	22.0
SUBTOTAL:			353.5
2 year inflation @ 15% per year			114.0
TOTAL . . 21			467.5

Source: https://www.industrydocuments.ucsf.edu/docs/mgby0228

Figure 31: Qualitative examples from the LaTeX document tool-use ablation. The model writes standalone LaTeX source, which is compiled and rasterized before evaluation.