UNIDIAGCODE: A MULTI-FORMAT BENCHMARK FOR IMAGE-TO-CODE DIAGRAM GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Image-to-code diagram generation converts rasterized diagrams into editable code representations. Prior work largely targets a single format, despite substantial differences in expressivity, code complexity, and downstream compatibility. We introduce UNIDIAGCODE, a unified six-format benchmark comprising 341K real-world rendered-image/code pairs and 6.3K hand-drawn sketches. We evaluate large vision language models across formats and reveal substantial differences in visual fidelity, renderability, and generation cost. We further formulate target-format selection as an input-dependent decision, allowing the model to choose an appropriate format for each diagram. This model-driven selection achieves a better quality–cost trade-off than fixed-format generation, while fine-tuning on our data substantially improves performance under both format specification and selection. By treating the output format as a model decision rather than a predefined constraint, our work offers a flexible and practical framework for code-based diagram generation.

1 INTRODUCTION

Diagrams communicate complex information across research, education, business, and many other domains. Code-based diagram generation has attracted growing attention because it produces resolution-independent vector graphics whose source code can be inspected, edited, and reused (Belouadi et al., 2024a;b; Rodriguez et al., 2025; Zou et al., 2024). Existing work includes both text-to-code (Belouadi et al., 2024a; 2025) and image-to-code generation (Belouadi et al., 2024b; Saito et al., 2025), with the latter supporting practical applications such as reusing existing diagrams and converting sketches into clean diagrams. Image-to-code systems can target domain-specific formats, such as Mermaid, PlantUML, and Graphviz, or general-purpose formats, such as TikZ, SVG, and draw.io. Domain-specific formats concisely represent particular diagram types but offer limited coverage, whereas general-purpose formats cover broader domains at the cost of longer code. The appropriate format therefore depends on the input diagram and affects both generation quality and cost. Nevertheless, most prior work focuses on a single format (Shbita et al., 2025; Scholz & Sadeghi, 2026; Belouadi et al., 2024a;b; Greisinger & Eger, 2026; Rodriguez et al., 2025; Yang et al., 2025b; Su et al., 2026). Even multi-format studies specify the target format in advance and evaluate each format independently (Zou et al., 2024; Yang et al., 2026; Zhao et al., 2026; Ni et al., 2026), leaving model-driven format selection largely unexplored.

We introduce UNIDIAGCODE, a unified image-to-code benchmark covering Mermaid, PlantUML, Graphviz, TikZ, SVG, and draw.io. It contains 341K real-world rendered-image/code pairs and 6.3K manually drawn sketches from diverse domains (Figure 1). For the same input diagrams, we evaluate original-format specification, general-purpose-format specification, and model-driven format selection (Figure 2). This design enables controlled analysis of format coverage, generation quality, code length, and selection behavior. We further show that fine-tuning on the collected data substantially improves performance over the base model in both the format-specification and format-selection settings. Our contributions are threefold: (1) we introduce UNIDIAGCODE, a six-format benchmark containing both rendered images and hand-drawn sketches; (2) we systematically compare format specification and selection, showing that SVG provides the most consistent cross-domain coverage among general-purpose formats, while input-dependent selection achieves a more favorable quality–cost trade-off; and (3) we fine-tune a single multi-format model, substantially

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

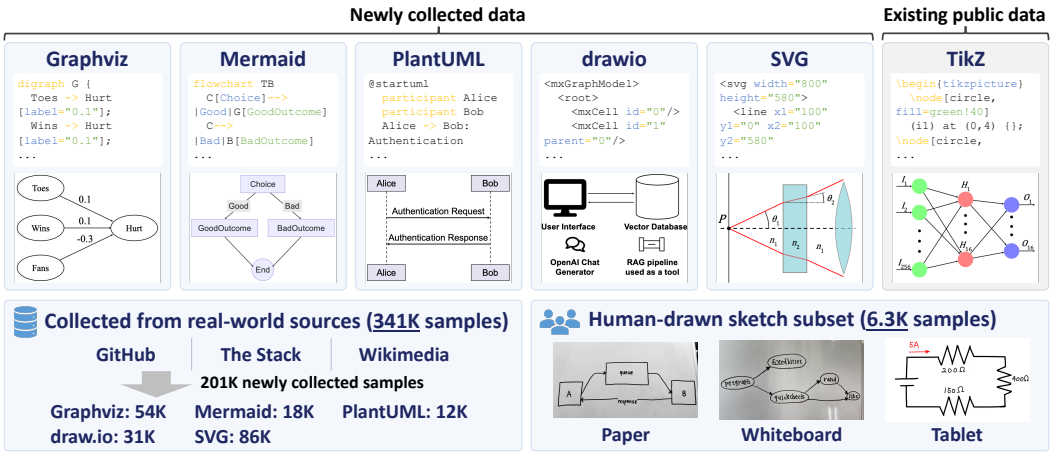


Figure 1: Overview of UNIDIAGCODE. A dataset of diagram code–image pairs covering six formats is constructed alongside sketches. For five formats (Graphviz, Mermaid, PlantUML, draw.io, and SVG), we collect 201K rendered-image/code pairs and create 3.3K hand-drawn sketches.

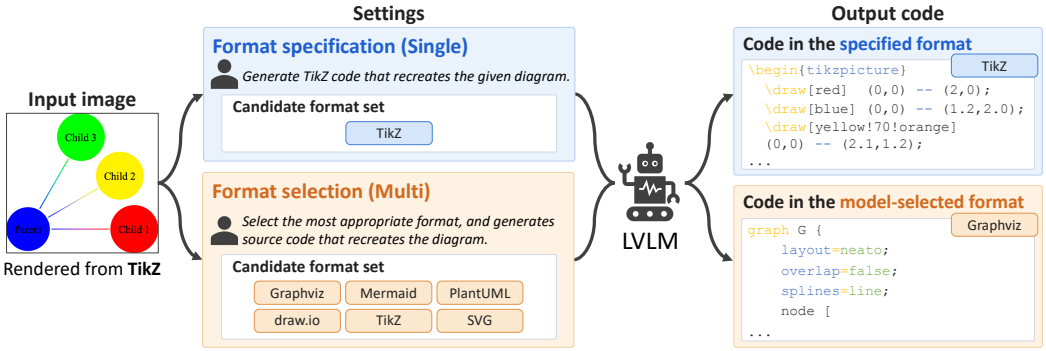


Figure 2: Overview of the task settings. The model receives a diagram image and generates code that reproduces it. We compare two settings: format specification (`Single`), in which a single candidate format is given and the model generates code in that format, and format selection (`Multi`), in which all six candidate formats are given and the model generates code in the format it selects.

improving all formats, surpassing much larger zero-shot models on some formats, and matching models trained separately for individual formats.

2 RELATED WORK

Image-to-code diagram generation. Image-to-code generation has been studied for webpage HTML (Beltramelli, 2018; Si et al., 2025) and mathematical LaTeX (Deng et al., 2017). Image-to-code diagram research has also grown but remains largely format-specific. Work on TikZ progressed from text-to-code generation to image-based reconstruction (Belouadi et al., 2024a;b), while other work has explored large-scale image-to-SVG generation (Rodriguez et al., 2025; Yang et al., 2025b) and draw.io conversion and evaluation (Cui et al., 2025; Su et al., 2026). These studies establish feasibility but lack a unified framework for comparing output formats. Recently, several multi-format datasets have emerged (Ni et al., 2026; Yang et al., 2026), yet few focus on diagrams while providing large-scale, real-world data across formats. Table 1 compares UNIDIAGCODE with representative multi-format datasets. VisCode-Multi-679K (Ni et al., 2026) primarily covers charts and plots; its real-world diagram subset contains 44K SVG samples. OmniDiagram (Yang et al., 2026) is entirely synthetic. Many synthetic datasets rely on proprietary LLMs such as GPT and Claude. By contrast, UNIDIAGCODE collects real-world data from GitHub and Wikimedia across five formats and

retains samples with licenses suitable for model training. Prior multi-format work aggregates formats without systematically comparing their representational scope and characteristics. Our work addresses this gap with unified format-specification and model-driven format-selection settings that expose format-specific strengths, limitations, and quality–cost trade-offs.

Format selection. Model-driven format selection has been explored outside diagram generation. Selecting among code, logical expressions, and structured text can improve reasoning efficiency and reduce communication costs between models (Chen et al., 2024). In code generation, models select Python in over 90% of cases when no language is specified (Twist et al., 2026a) and may implement solutions in a language different from the one they recommend (Twist et al., 2026b). Visual-document systems similarly select among HTML, LaTeX, and Markdown according to the input document type (Xiao et al., 2025). However, model-driven output-format selection has not been systematically studied for diagram generation.

3 Task Definition

3.1 Task Formulation and Output-format Settings

Given a diagram image I, a model M generates code C in a format $f \in \mathcal{F}$ whose rendering $\hat{I} = R_f(C)$ reconstructs I. $\mathcal{F}$ contains the specialized formats Graphviz, Mermaid, and PlantUML, which support only certain diagram types, and the general-purpose formats $\mathcal{F}_{\text{general}} = \{$draw.io, TikZ, SVG $\}$. Let f_{src} denote the format of the source code of I.

Format specification (`Single`). Given a target format f, the model generates $C = M(I, f)$. We consider two choices of f:

- **Original format** ($f = f_{\text{src}}$; Original): the model writes code in the same format as the source, measuring its competence in each format.
- **General-purpose format** ($f \in \mathcal{F}_{\text{general}}$): the model writes the diagram in one general-purpose format regardless of its source format, revealing the trade-off between reconstruction quality and code length across formats that can represent any diagram.

Format selection (`Multi`). The model selects a format and generates code as $(f, C) = M(I, \mathcal{F})$. This setting evaluates both generation performance and input-dependent format selection.

3.2 Input Image Types

Rendered images. I_{raster} is rendered from the source code, reflecting the reuse or migration of existing diagrams; the model must faithfully reproduce geometry and text placement.

Hand-drawn sketches. I_{sketch} is a photo or digital image of a hand-drawn diagram, reflecting the conversion of rough drafts into clean diagrams; the model must recover the logical structure while correcting drawing irregularities.

4 Dataset Construction

Collection of diagram code and rendered images. Among the six formats considered in this work, we newly collect data for five formats: Mermaid, PlantUML, Graphviz, SVG, and draw.io. For TikZ, we use DaTikZ-v3 (Belouadi et al., 2025). For the five newly collected formats, we obtain real-world diagram code from diverse open repositories: Mermaid and draw.io via keyword-based searches on GitHub and Markdown blocks in the-stack-dedup[1]; PlantUML and Graphviz from the-stack-dedup subsets; and SVG from Wikimedia Commons covering physics, mathematics, chemistry, and computer science. UniDiagCode provides diversity not only in output formats but also in the domains represented by the diagrams. Specifically, domain-specific formats primarily cover network and dependency structures (Graphviz), business logic and flowcharts (Mermaid),

[1] https://huggingface.co/datasets/bigcode/the-stack-dedup

Table 1: Comparison with VisCode-Multi-679K (Ni et al., 2026) and M^{3}Diagram (Yang et al., 2026).

		VisCode	M^{3}Diagram	**Ours**
Domain	Chart	✓	✗	✗
	Diagram	✓	✓	✓
Source	Synthetic	✓	✓	✗
	Real	✓	✗	✓
Image	Rendered	✓	✓	✓
	Sketch	✗	✗	✓
#Formats		12	3	6
#Real samples		290K	–	**347K**

Table 2: Dataset sizes and token-length statistics by format in UNIDIAGCODE.

	Image Type		Length
Format	Rendered	Sketch	Median
Newly collected			
Graphviz	53,623	1,354	846
Mermaid	18,478	192	94
PlantUML	11,932	298	176
draw.io	31,203	180	4,596
SVG	85,808	1,270	4,704
From DaTikZ-v3 (Belouadi et al., 2025)			
TikZ	139,844	2,967	469

and software design diagrams such as class and sequence charts (PlantUML). In contrast, general-purpose formats encompass academic and mathematical figures (TikZ), scientific illustrations across natural sciences (SVG), and system architecture layouts with nested containers (draw.io). Detailed retrieval queries, repository URLs and license filtering are provided in Appendix B.

Creation of hand-drawn sketches. Following Sketch2Diagram (Saito et al., 2025), annotators manually redrew a subset of the collected diagrams on paper, whiteboards, and stylus-equipped tablets using the corresponding rendered images as references, thereby capturing realistic variations in drawing tools, geometric distortions, and lighting. For TikZ, we used SkeTikZ (Saito et al., 2025).

Dataset Statistics. Table 2 summarizes UNIDIAGCODE: 341K (201K) rendered-image/code pairs from real-world repositories and documents, plus 6.3K (3.3K) manually drawn sketches across six formats; parenthesized values denote data newly collected or created for five formats in this work. To our knowledge, it is the largest fully real-world multi-format diagram corpus and the first to include manually drawn sketches. We split each format into training, validation, and test sets and manually filtered the web-collected test data for visual quality and readability. The final test sets contain 300 rendered diagrams per format (1,800 total) and hand-drawn versions of 150 of them per format (900 total), enabling paired comparison between rendered and sketched inputs.

5 EVALUATION METRICS

We evaluate diagram generation models at two levels: code-level evaluation, which measures rendering success and code efficiency, and image-level evaluation, which measures fidelity to the input diagram and visual quality. Let N denote the total number of samples in the test set.

5.1 CODE-LEVEL EVALUATION

Rendering failure rate (RFR). RFR measures the percentage of generated code that fails to render and therefore does not yield an output diagram image. This metric captures fatal failures in diagram generation:

$$\mathrm{RFR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\text{failed_to_render}(C_{\text{gen},i})] \times 100\%. \tag{1}$$

Cases in which code generation was truncated upon reaching the specified maximum length were also counted as rendering failures.

Code length (CL). CL measures the amount of code required by a model to represent a diagram and evaluates code efficiency and generation cost across formats. For each sample, we measure code length as the number of tokens in the generated diagram code. However, because tokenization schemes differ across models, raw token counts are not directly comparable across models. We

therefore compute token-count ratios across output-format settings separately for each model, using the original setting as the reference, and use these normalized ratios to compare relative differences in code length.

5.2 IMAGE-LEVEL EVALUATION

We compare the rendered output $\hat{I}$ with the input image I using Gemini 3.1 Flash Lite as an LLM judge (Yang et al., 2025a; Wu et al., 2025). Validation on 300 human-rated samples confirmed strong correlations with human judgments; evaluation prompts and protocols are provided in Appendices C.2 and C.3. We evaluate each output using the following two metrics on a scale from 1 to 5 with increments of 0.5. A sample that fails to render receives the minimum score of 1. We calculate the mean score over all samples in the test set.

Image-level alignment (IA). IA evaluates how faithfully the generated diagram $\hat{I}$ preserves the content of the input diagram image I. For hand-drawn-sketch input, differences resulting from the regularization of hand-drawn distortions, refinement of geometric elements, or alignment of the layout are not penalized as long as the logical meaning of the diagram is preserved.

Diagram quality (DQ). DQ evaluates visual quality and readability independently of input fidelity, focusing on overlaps and text occlusion for rendered-image inputs and assessing clean-diagram quality for sketches based on line consistency, shape regularity, spacing, alignment, and overall visual organization.

6 EXPERIMENTS

6.1 EXPERIMENTAL SETUP

Models. We evaluate LVLMs spanning diverse architectures and parameter scales, including commercial frontier models from the Gemini, GPT, and Claude families and representative open-weight models from the Qwen3.5, Qwen3-VL, Gemma 4, and GLM-V families.

Inference settings. All models use structured JSON decoding for syntactic stability and automatic evaluation. In the format-specification setting, the model recreates the input in the specified format and returns {`"code": "..."`}; in the format-selection setting, it chooses among six described formats, prioritizing structural, semantic, and visual fidelity and using conciseness only as a tie-breaker, and returns {`"format": "...", "code": "..."`}. Complete prompts are provided in Appendix D.1. We determined the maximum output length for each format from the empirical code-length distributions (Table 11). We used the default inference-time sampling parameters for each model or API.

Fine-tuning. To assess the utility of the collected data, we fine-tune Qwen3-VL-8B. To improve robustness to sketch inputs, we train on both rendered images and versions augmented with synthetic sketch-like noise. For `Single` training, we use the same prompt template as at inference time, explicitly specifying a single target format (e.g., "Generate LaTeX TikZ code that recreates the given diagram"), and train the model on paired images and corresponding code to reconstruct each diagram in the specified format. For `Multi` training, we further fine-tune the `Single`-trained model for 250 steps to add input-dependent format selection to its format-specific generation capabilities. We use 10,000 samples per format, excluding draw.io, with each sample's source format and corresponding code as targets. Preliminary experiments showed that fine-tuning on the collected SVG code in its raw form degraded model performance because of its complexity and verbosity. We therefore normalized the code with SVGO and simplified it using Gemma-4-31B before training. Further details of the fine-tuning procedure and its hyperparameters are provided in Appendix D.2.

6.2 RESULTS AND ANALYSIS ON RENDERED IMAGES

How well do models perform when the original format is specified? Table 3 reports IA, DQ, and RFR in the `Single` (Original) setting. Proprietary models outperformed open-weight models

Table 3: Performance with rendered images as input and the original format specified (`Single`, Original). The best value in each column is shown in bold.

Model	Graphviz			Mermaid			PlantUML			draw.io			TikZ			SVG		
	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR
Open-weight models																		
Qwen3-VL-4B	2.0	2.7	44.7	2.3	2.8	47.3	1.6	2.0	69.3	1.0	1.0	99.7	1.6	2.2	38.7	1.5	2.1	**0.0**
Qwen3-VL-8B	2.3	3.2	38.0	2.7	3.4	34.7	2.0	2.6	51.7	1.0	1.0	99.7	1.8	2.5	28.7	1.8	2.8	**0.0**
Qwen3-VL-30B	2.4	3.2	36.3	3.4	4.0	19.7	1.9	2.5	54.0	1.0	1.0	100.0	2.2	2.9	19.7	1.8	2.4	**0.0**
Gemma-4-31B	3.7	4.5	4.0	4.0	4.5	9.0	3.0	3.6	29.3	2.7	3.0	33.7	2.9	3.3	12.3	3.2	3.6	**0.0**
Proprietary models																		
GPT-5.4	3.7	4.6	1.0	3.7	4.2	14.3	2.7	3.6	28.0	3.9	4.0	**1.0**	3.9	4.1	**1.3**	4.2	4.4	**0.0**
Claude Opus 4.7	4.2	4.8	0.7	4.2	4.6	5.0	3.8	4.6	4.0	3.6	4.1	7.3	3.8	4.2	4.3	3.9	**4.5**	**0.0**
Gemini 3.1 Pro	**4.6**	**4.9**	**0.0**	**4.5**	**4.8**	**3.7**	**4.5**	**4.8**	**2.7**	**4.5**	**4.6**	2.7	**4.5**	**4.7**	2.7	**4.3**	4.4	0.3
Fine-tuned model (Ours)																		
Qwen3-VL-8B	4.2	4.6	4.3	3.7	4.1	19.0	3.8	4.3	11.7	3.2	3.4	3.0	2.4	2.9	14.7	3.0	3.4	**0.0**

on nearly every format. Gemini 3.1 Pro achieved the highest IA across all formats and the highest DQ except on SVG, where Claude Opus 4.7 scored marginally higher, while consistently maintaining low RFR. Across individual formats, models performed relatively well on Graphviz and Mermaid: among open models, Gemma-4-31B achieved IA and DQ scores between 3.7 and 4.5 with RFR below 10%, while Qwen3-VL-30B-A3B reached DQ scores above 3 on both formats. In contrast, open-weight models frequently generated invalid draw.io and PlantUML code, likely reflecting the relative scarcity of web data for these formats. SVG and TikZ had lower RFR than draw.io and PlantUML but remained weak in IA and DQ.

How effective is fine-tuning on the collected data? The fine-tuned model, shown in the bottom row of Table 3, substantially reduced RFR across all formats, indicating more reliable syntax generation. For Graphviz and Mermaid, IA and DQ rose to approximately 4, surpassing Qwen3-VL-30B and matching or approaching GPT-5.4. On PlantUML and draw.io, they reached approximately 3–4, substantially exceeding Qwen3-VL-30B but leaving room for improvement. SVG and TikZ improved less relative to their training-set sizes, remaining around 2.5–3.5. This likely reflected their intrinsic diagram diversity and numerous valid syntactic representations, making fundamental improvements in these expressive formats an open challenge.

How well do models perform when a general-purpose format is specified? Table 4 and Figure 3 compare aggregate performance across all input formats between the `Single` (Original) setting, where each diagram is generated in its original format, and the `Single` general-purpose setting, where all diagrams are generated in SVG, TikZ, or draw.io. Performance differed sharply: Gemini 3.1 Pro performed strongly across all three formats, whereas SVG was the only consistently robust target for other models. Except for Gemini 3.1 Pro and our fine-tuned model, generating all diagrams in SVG improved IA over Original while reducing RFR to nearly zero. Fine-tuning reversed this pattern, with Original yielding higher IA. TikZ and draw.io underperformed SVG and produced substantially higher RFR, especially for open-weight models, indicating frequent syntax failures. These disparities may reflect differences in the abundance of web training data across formats.

Do domain-specific formats offer advantages? Figure 3 shows how performance varies with the output format for each input domain. Within their intended scope, domain-specific formats generally outperformed SVG for open-weight models while producing shorter code. Qwen3-VL-8B achieved higher DQ on the Graphviz and Mermaid subsets and higher IA on Mermaid using their respective formats. Fine-tuning strengthened this advantage, substantially improving performance on the Graphviz, Mermaid, and PlantUML subsets. These formats also avoided SVG's 1.5–3.1$\times$ increase in token count (Table 4). However, this quality advantage was not universal among proprietary models: on the Graphviz subset, GPT-5.4 achieved higher IA with SVG than with Graphviz (4.5 vs. 3.7), indicating that the benefit depended on each model's proficiency with general-purpose alternatives.

Table 4: Performance for each output-format setting, with rendered images as input. Each cell averages over the combined test sets of the six original formats, 1,800 examples in total.

Model	Single															Multi			
	Original			draw.io				TikZ				SVG							
	IA	DQ	RFR	IA	DQ	RFR	CL	IA	DQ	RFR	CL	IA	DQ	RFR	CL	IA	DQ	RFR	CL
Open-weight models																			
Qwen3-VL-4B	1.7	2.1	49.9	1.0	1.0	99.8	1.8	1.6	2.0	38.2	1.3	1.9	2.3	**0.0**	3.1	1.5	1.8	72.1	0.6
Qwen3-VL-8B	1.9	2.6	42.1	1.0	1.0	99.2	1.4	1.8	2.2	37.1	1.0	2.3	2.7	**0.0**	2.6	1.8	2.3	55.1	0.7
Qwen3-VL-30B	2.1	2.7	38.3	1.0	1.0	100.0	—	1.9	2.3	39.0	1.1	2.5	3.0	**0.0**	3.0	2.1	2.7	42.0	0.9
Gemma-4-31B	3.3	3.7	14.7	2.7	3.0	31.2	2.8	2.7	2.9	19.3	0.8	3.5	3.6	**0.0**	1.9	3.2	3.7	13.2	0.6
Proprietary models																			
GPT-5.4	3.7	4.1	7.6	3.7	3.9	**1.0**	2.4	3.6	3.8	5.4	1.1	4.3	4.4	**0.0**	1.5	4.0	4.3	3.6	1.1
Claude Opus 4.7	3.9	4.5	3.6	3.4	3.9	8.3	2.4	3.8	4.1	3.8	1.0	4.2	**4.6**	**0.0**	1.6	3.9	4.5	1.8	0.6
Gemini 3.1 Pro	**4.5**	**4.7**	**2.0**	**4.3**	**4.4**	3.7	2.8	**4.4**	**4.5**	**2.8**	0.8	**4.4**	4.5	0.1	2.6	**4.5**	**4.7**	**1.4**	0.6
Fine-tuned model (Ours)																			
Qwen3-VL-8B	3.4	3.8	8.8	3.0	3.2	2.8	2.8	2.0	2.4	23.3	0.6	3.1	3.2	**0.0**	1.4	3.2	3.7	11.4	0.8

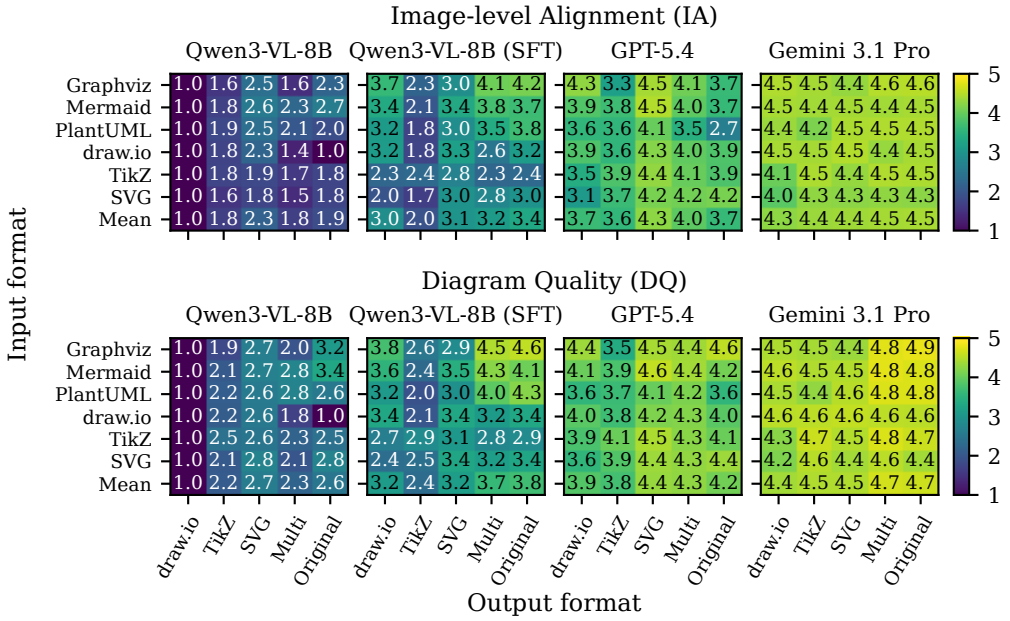


Figure 3: Performance comparison across output-format settings for rendered-image inputs: `Multi` and `Single` (Original and general-purpose formats). The upper row shows image-level alignment (IA) and the lower row shows diagram quality (DQ), with four models side by side. Within each panel, rows represent the original format of the input diagram and columns represent the specified output format. The `Mean` row averages over the six input formats.

In summary, domain-specific formats provide a compelling advantage in both generation quality and code conciseness within their applicable domains, particularly for open-weight models.

Is model-driven format selection effective? The `Multi` column of Table 4 presents the results obtained under the format-selection setting. For Gemini 3.1 Pro, the `Multi` setting achieved IA and DQ scores comparable to Original and higher than those obtained using SVG alone. Moreover, its CL was 0.6, demonstrating superior cost efficiency by achieving comparable fidelity with only 60% of the code length required by the Original setting. Unlike SVG-only generation, which produced strong performance but inflated code length by 1.5–3.1×, the `Multi` setting achieved high fidelity

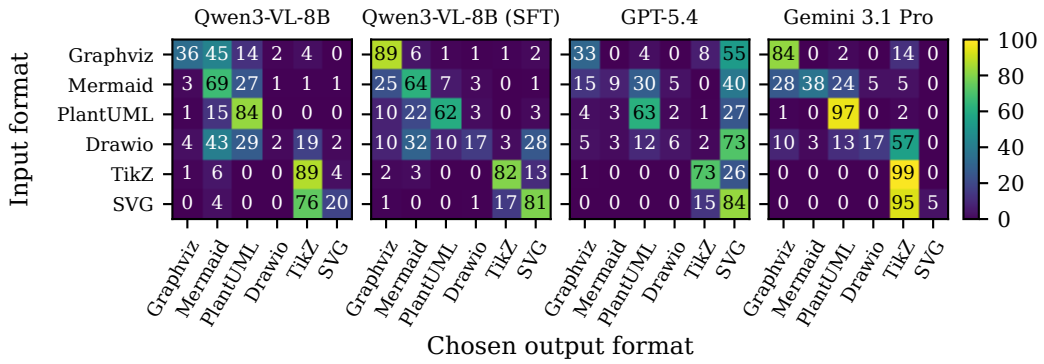


Figure 4: Distribution of the output format chosen under the format-selection (`Multi`) setting, with rendered images as input. Within each panel, rows represent the original format of the input diagram, and columns represent the output format chosen by the model.

without incurring excessive generation cost. For most other models, including the fine-tuned model, `Multi` likewise reduced code length while broadly matching the performance of the Original setting, with GPT-5.4 being the sole exception (CL = 1.1). For smaller open-weight models, however, letting the model select the format raised RFR substantially (from 49.9% to 72.1% for Qwen3-VL-4B and from 42.1% to 55.1% for Qwen3-VL-8B), indicating that the benefit of format selection requires baseline competence in generating valid code across multiple formats. These results demonstrate that, for sufficiently capable models, adaptively selecting the output format for each diagram maintains or improves performance while substantially reducing generation cost, achieving an optimal quality–cost trade-off.

How do models select the output format? Figure 4 shows output-format choices under the `Multi` setting. Gemini 3.1 Pro retained Graphviz, Mermaid, and PlantUML inputs in their source formats but converted verbose SVG and draw.io inputs to TikZ, favoring the most concise format capable of faithful representation. In contrast, GPT-5.4 favored SVG regardless of the source format. Qwen3-VL-8B retained Mermaid, PlantUML, and TikZ but redirected draw.io and SVG inputs to Mermaid or TikZ. Fine-tuning increased source-format retention, particularly for Graphviz and SVG, which the base model had often redirected to Mermaid and TikZ. However, it continued to avoid draw.io, the most verbose format, reducing code length while maintaining performance.

6.3 Results and Analysis on Sketch Images

How well do models perform when the original format is specified? Table 5 reports results for hand-drawn sketches under the `Single` (Original) setting. Trends largely mirrored rendered-image results: Gemini 3.1 Pro performed strongly across all formats, while open-weight models showed high RFR on PlantUML and draw.io. Although SVG and TikZ had substantially lower failure rates, their IA and DQ remained comparatively low for smaller open-weight models. Our fine-tuned model substantially outperformed its base model and matched GPT-5.4 on Graphviz, Mermaid, and PlantUML, demonstrating effective generalization to the visual-style shift of sketches. Despite improvements across all formats, scores on expressive formats (draw.io, TikZ, and SVG) remained around 3, leaving complex, diverse diagram generation by smaller models an open challenge.

Is model-driven format selection effective? Table 6 presents results averaged over the entire dataset under the `Single` and `Multi` settings. Consistent with findings on rendered images, proprietary models under the `Multi` setting matched the fidelity of the Original and SVG settings while requiring substantially less code (CL = 0.5–0.9). Our fine-tuned model likewise reduced code length (CL = 0.6) while maintaining performance broadly comparable to specifying the original format. For open-weight models, however, `Multi` exacerbated rendering failures even more severely than on rendered images: for instance, RFR for Qwen3-VL-30B rose from 35.1% to 47.9% on sketches, compared to an increase from 38.3% to 42.0% on rendered inputs. These results indicate that model-driven format selection remains effective for sketch inputs when models possess

Table 5: Performance with hand-drawn sketches as input and the original format specified. The best value in each column is shown in bold.

Model	Graphviz			Mermaid			PlantUML			draw.io			TikZ			SVG		
	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR
Open-weight models																		
Qwen3-VL-8B	2.9	3.6	21.3	3.0	3.5	26.0	2.4	2.9	42.0	1.0	1.0	100.0	2.3	2.7	22.0	2.1	2.6	**0.0**
Qwen3-VL-30B	3.1	3.5	25.3	3.5	3.9	18.0	2.3	2.6	49.3	1.0	1.0	100.0	2.4	2.9	18.0	1.6	1.8	**0.0**
Proprietary models																		
GPT-5.4	4.0	4.5	0.7	4.0	4.3	10.7	3.5	3.9	16.0	4.2	4.1	0.7	4.2	4.3	0.7	4.1	4.0	**0.0**
Gemini 3.1 Pro	**4.8**	**4.9**	**0.0**	**4.8**	**4.9**	**0.0**	**4.8**	**4.7**	**0.7**	**4.8**	**4.8**	**0.0**	**4.8**	**4.8**	**0.0**	**4.8**	**4.8**	**0.0**
Fine-tuned model (Ours)																		
Qwen3-VL-8B	4.3	4.7	1.3	4.1	4.2	14.7	4.1	4.3	11.3	3.4	3.4	2.7	2.7	3.1	11.3	3.3	3.4	**0.0**

Table 6: Performance for each output-format setting, with hand-drawn sketches as input. Each cell averages over the combined test sets of the six input formats, 900 examples in total.

Model	Single															Multi			
	Original			draw.io				TikZ				SVG							
	IA	DQ	RFR	IA	DQ	RFR	CL	IA	DQ	RFR	CL	IA	DQ	RFR	CL	IA	DQ	RFR	CL
Open-weight models																			
Qwen3-VL-8B	2.3	2.7	35.2	1.0	1.0	98.9	1.7	2.2	2.4	26.1	1.1	2.7	2.8	**0.0**	2.9	2.0	2.4	49.4	0.6
Qwen3-VL-30B	2.3	2.6	35.1	1.0	1.0	100.0	—	2.3	2.5	32.2	1.2	2.5	2.7	**0.0**	3.4	2.2	2.5	47.9	0.6
Proprietary models																			
GPT-5.4	4.0	4.2	4.8	4.1	4.0	**0.8**	2.5	4.2	4.2	1.8	1.1	4.4	4.3	**0.0**	1.5	4.2	4.4	4.4	0.9
Gemini 3.1 Pro	**4.8**	**4.8**	**0.1**	**4.7**	**4.6**	1.6	2.9	**4.8**	**4.8**	**0.9**	0.9	**4.7**	**4.7**	**0.0**	2.0	**4.7**	**4.8**	**0.9**	0.5
Fine-tuned model (Ours)																			
Qwen3-VL-8B	3.6	3.9	6.9	3.4	3.4	2.6	3.0	2.4	2.7	17.1	0.6	3.5	3.4	**0.0**	1.4	3.3	3.6	13.0	0.6

sufficient multi-format competence, reproducing the favorable quality–cost trade-off observed on rendered images. Per-model heatmaps for both settings, broken down by input format, are provided in Appendix F.

7 Limitations

One limitation of this work is that many diagram-representation formats exist beyond the six formats evaluated in our benchmark. We focused on representative formats that have been widely adopted in prior research. Extending the benchmark to less widely used formats and investigating how models can effectively select among a broader range of tools remain important directions for future work.

8 Conclusion

We introduced UniDiagCode, a unified benchmark for consistent multi-format evaluation, collecting 341K rendered-image/code pairs across six formats. Beyond conventional target-format specification, we introduced model-driven format selection for each input. General-purpose formats such as SVG performed consistently well but incurred higher generation costs because of longer code. For models proficient across formats, input-dependent selection maintained or improved quality while reducing cost. Fine-tuning a single model on our data substantially improved all formats in both settings, outperforming much larger zero-shot models. These findings illuminate the largely unexplored question of which output format is best suited to code-based vector diagram generation.

REFERENCES

Jonas Belouadi, Anne Lauscher, and Steffen Eger. AutomaTikZ: Text-guided synthesis of scientific vector graphics with TikZ. In *The Twelfth International Conference on Learning Representations*, 2024a.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. DeTikZify: Synthesizing graphics programs for scientific figures and sketches with TikZ. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.

Jonas Belouadi, Eddy Ilg, Margret Keuper, Hideki Tanaka, Masao Utiyama, Raj Dabre, Steffen Eger, and Simone Ponzetto. TikZero: Zero-shot text-guided graphics program synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17793–17806, October 2025.

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '18, New York, NY, USA, 2018. Association for Computing Machinery.

Weize Chen, Chenfei Yuan, Jiarui Yuan, Yusheng Su, Chen Qian, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Beyond natural language: LLMs leveraging alternative formats for enhanced reasoning and communication. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10626–10641, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

Zhiqing Cui, Jiahao Yuan, Hanqing Wang, Yanshu Li, Chenxu Du, and Zhenglong Ding. Draw with thought: Unleashing multimodal reasoning for scientific diagram generation. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, pp. 5050–5059, New York, NY, USA, 2025. Association for Computing Machinery.

Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. Image-to-markup generation with coarse-to-fine attention. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 980–989, Sydney, NSW, Australia, 2017. JMLR.org.

Christian Greisinger and Steffen Eger. TikZilla: Scaling text-to-TikZ with high-quality data and reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yuansheng Ni, Songcheng Cai, Xiangchao Chen, Jiarong Liang, Zhiheng Lyu, Jiaqi Deng, Kai Zou, Ping Nie, Fei Yuan, Xiang Yue, and Wenhu Chen. VisCoder2: Building multi-language visualization coding agents. In *The Fourteenth International Conference on Learning Representations*, 2026.

Juan A Rodriguez, Abhay Puri, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. StarVector: Generating scalable vector graphics code from images and text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16175–16186, June 2025.

Itsumi Saito, Haruto Yoshida, and Keisuke Sakaguchi. Sketch2Diagram: Generating vector diagrams from hand-drawn sketches. In *The Thirteenth International Conference on Learning Representations*, 2025.

Simon Scholz and Mersedeh Sadeghi. CAS2UML: A handwritten sketch-to-PlantUML dataset for class and activity diagrams, 2026.

Basel Shbita, Farhan Ahmed, and Chad DeLuca. MermaidSeqBench: An evaluation benchmark for LLM-to-mermaid sequence diagram generation. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025.

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 (eds.), *Proceedings of the 2025 Conference of the Nations*

of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 3956–3974, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

Xiaoyan Su, Peijie Dong, Zhenheng Tang, Song Tang, Yuyao Zhai, Kaitao Lin, Liang Chen, Gai Yuhang, Yuyu Luo, Qiang Wang, and Xiaowen Chu. VCG-bench: Towards a unified visual-centric benchmark for structured generation and editing. In *Forty-third International Conference on Machine Learning*, 2026.

Lukas Twist, Mark Harman, Don Syme, Joost Noppen, Helen Yannakoudakis, Detlef Nauck, and Jie M Zhang. A study of LLMs' preferences for libraries and programming languages. In Maria Liakata, Viviane P Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 331–351, San Diego, California, United States, July 2026a. Association for Computational Linguistics.

Lukas Twist, Twm Stone, Helen Yannakoudakis, and Jie M Zhang. LangChoiceBench: Measuring and explaining programming-language choice in LLMs, 2026b.

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 Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3006–3028, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

Han Xiao, Yina Xie, Guanxin Tan, Yinghao Chen, Rui Hu, Ke Wang, Aojun Zhou, Hao Li, Hao Shao, Xudong Lu, Peng Gao, Yafei Wen, Xiaoxin Chen, Shuai Ren, and Hongsheng Li. Adaptive markup language generation for contextually-grounded visual document understanding. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 29558–29568, 2025.

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.

Haoyue Yang, Xuanle Zhao, Xuexin Liu, Feibing Jiang, and Yao Zhu. OmniDiagram: Advancing unified diagram code generation via visual interrogation reward. In Maria Liakata, Viviane P Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16430–16452, San Diego, California, United States, July 2026. Association for Computational Linguistics.

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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haoyue Yang, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. VinciCoder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning, 2026.

Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. VGBench: Evaluating large language models on vector graphics understanding and generation. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 3647–3659, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

A Disclosure of AI Use

In this work, we used generative AI tools to generate synthetic datasets, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, and clean and reformat datasets. We have not used generative AI tools to generate

synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, support qualitative and thematic data analysis, or interpret results that are not applicable to this work.

B Dataset Construction

B.1 Collection Procedure

As TikZ diagrams were adopted from existing datasets (Belouadi et al., 2025; Saito et al., 2025), source files for the remaining five formats were retrieved from public repositories as follows:

- **Mermaid**: Mermaid diagram scripts were retrieved via keyword search on the GitHub Search API, and additional diagram code blocks were extracted from the Markdown subset of `the-stack-dedup` (`https://huggingface.co/datasets/bigcode/the-stack-dedup`).
- **PlantUML**: Permissively licensed code was extracted from the PlantUML subset of `the-stack-v2-dedup` (`https://huggingface.co/datasets/bigcode/the-stack-v2-dedup`).
- **Graphviz**: Diagram source code written in the DOT language was extracted from `the-stack-dedup`.
- **SVG**: Technical diagrams spanning mathematics, physics, chemistry, and computer science were gathered from Wikimedia Commons (`https://commons.wikimedia.org/`).
- **draw.io**: Diagram XML files with `.drawio` and `.xml` extensions were retrieved via keyword search on the GitHub Search API.

B.2 License Filtering

For data sourced from GitHub and The Stack, we retained only samples licensed under MIT, Apache-2.0, BSD-2-Clause, BSD-3-Clause, CC0-1.0, CC-BY-4.0, CC-BY-SA-4.0, CC-BY-3.0, CC-BY-SA-3.0, or the Unlicense. For Mermaid and draw.io, which were collected directly from GitHub, we used the license of the source repository; for Graphviz and PlantUML, we used the per-file license recorded in The Stack. For Wikimedia data, we retained only samples with one of the following licenses or designations: Public Domain, CC BY-SA 4.0, CC BY-SA 3.0, CC0, CC BY 4.0, CC BY 3.0, Apache License 2.0, MIT, or BSD.

B.3 Rendering and Preprocessing

The collected code was rendered into images inside isolated container environments dedicated to each format. Prior to rendering, diagram code blocks were extracted from each collected source file. For draw.io, because the content of the `diagram` element is often stored in compressed form, it was decompressed and formatted into clean XML.

Table 7 summarizes the rendering engine, conversion pipeline, and resolution settings for each format. We explicitly specified a resolution of 300 dpi for Graphviz, TikZ, and SVG, whereas the remaining three formats were rendered using their engines' default scales. Margin cropping was applied only to TikZ and draw.io. A timeout limit of 600 seconds was enforced for each rendering conversion.

Rendered images were downscaled to ensure the longer side did not exceed 1,024 px while preserving the aspect ratio. Images were never upscaled, and any image whose shorter side fell below 32 px after resizing was discarded.

Code that failed to compile or render was excluded from all subsequent steps. To eliminate exact visual duplicates among the remaining samples, we computed cryptographic content hashes of the images and retained only a single representative per duplicate group.

Table 7: Rendering settings for each format. "Direct" means the renderer writes the PNG itself, whereas TikZ goes through an intermediate PDF.

Format	Renderer	Path	Resolution
Graphviz	Graphviz (dot)	direct	300 dpi
Mermaid	mermaid-cli (mmdc)	direct	default
PlantUML	PlantUML	direct	default
draw.io	diagrams.net CLI	direct	default
TikZ	pdflatex → pdftoppm	via PDF	300 dpi
SVG	Inkscape	direct	300 dpi

B.4 Splits and Manual Inspection

For each format, the authors manually inspected and selected 300 high-quality diagrams to construct the test set. Of these, 150 diagrams were paired with corresponding hand-drawn sketches, while the remaining 150 had no sketch counterpart. This pairing enables evaluating models on the exact same underlying diagrams under both clean rendered-image and hand-drawn sketch conditions.

Manual inspection followed two criteria: (1) an intact visual layout free of rendering artifacts, and (2) clearly legible text and diagram elements. These criteria effectively excluded corrupted renderings or illegibly dense diagrams common in raw web-collected data.

From the remaining pool, we randomly sampled 300 diagrams per format for the validation set, allocating all remaining samples to the training set.

For TikZ, we followed the original partition of DaTikZ v3 (Belouadi et al., 2025). Specifically, we drew 300 test samples from its test split and 300 validation samples from its training split, matching the sample counts of the other five formats.

B.5 Examples from the Dataset

Figure 5 shows one example diagram from UniDiagCode for each format. The formats differ in the kinds of diagrams they predominantly describe: Graphviz for node-and-edge structures, Mermaid for process flows, PlantUML for UML diagrams, TikZ for academic figures containing mathematical expressions and geometric constructions, SVG for scientific schematics, and draw.io for freely arranged layouts. Each format can nonetheless express a wide variety of diagrams, and the tendencies above reflect their typical usage patterns in the source repositories.

Figure 6 shows hand-drawn sketches alongside their corresponding rendered reference images. The sketches were produced on three media—paper, whiteboards, and tablets—which differ in stroke width and background texture, and the two photographed physical media additionally introduce geometric distortion. In all cases, the structure of the original diagram is preserved while fine details are omitted and strokes deviate from straight lines, reflecting properties of realistic hand-drawn input.

C Evaluation Details

This section describes how the evaluation metrics are computed and aggregated. Section C.1 details the aggregation rules for the code-level metrics (RFR and CL); Section C.2 presents the prompts provided to the judge model for the image-level metrics (IA and DQ); and Section C.3 outlines the human evaluation protocol used to validate the judge model.

C.1 Code-level Evaluation

Truncation and fallback. When code generation reaches the maximum sequence length and is truncated, the code cannot be rendered and is counted as a failure in RFR. The renderers for SVG and draw.io, however, can fault-tolerantly parse and render incomplete code by completing missing elements, so for these two formats we apply a fallback procedure that attempts to render truncated code. When structured output is truncated, the response is not valid JSON and standard decoding

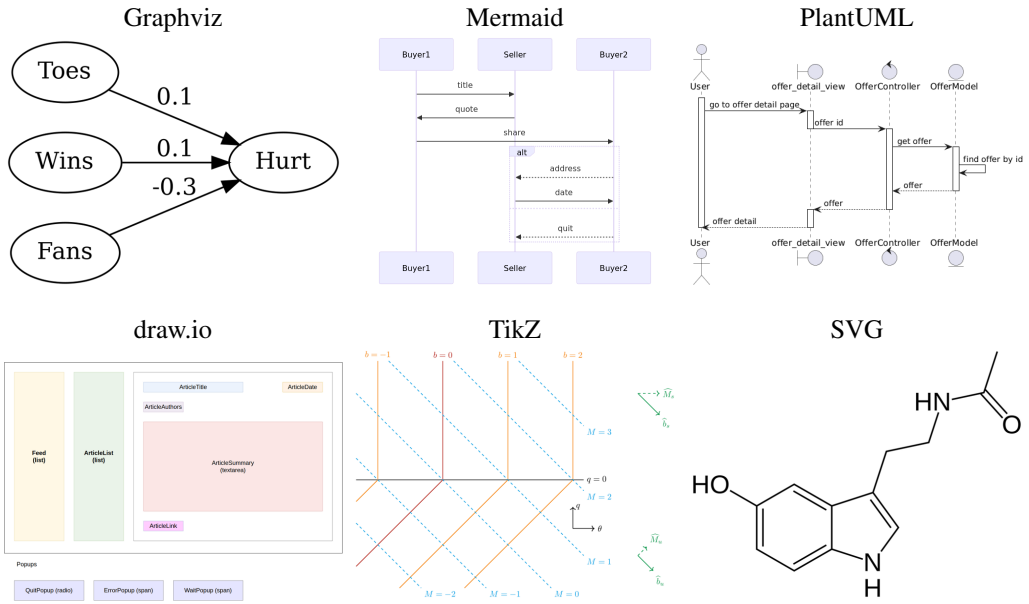


Figure 5: Examples of diagrams in UNIDIAGCODE. One example is shown from the test set of each of the six formats (Graphviz, Mermaid, PlantUML, draw.io, TikZ, and SVG).

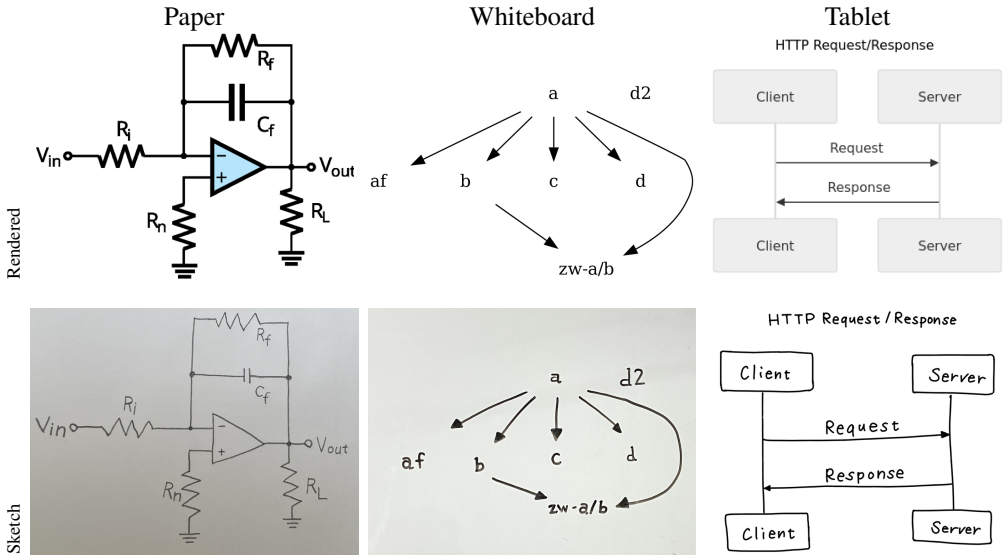


Figure 6: Correspondence between hand-drawn sketches and the rendered images they were drawn from. The sketches were produced on three media: paper, whiteboard, and tablet. Each column corresponds to the same sample; the upper row shows the rendered image and the lower row the corresponding sketch.

fails to extract the code; we therefore extract the code string directly from the raw response and pass it to the renderer. Because the renderers for the other formats lack such error tolerance and truncated code simply fails to compile or render, this fallback is not applied to them.

The same rule applies in the format-selection setting. Because the output schema places `format` before `code` and fields are generated in schema order, the declared format can still be identified from a response truncated within `code`. We apply this fallback recovery only when the declared format is SVG or draw.io.

Samples included in CL. CL is computed exclusively on samples that both produced valid structured output and rendered successfully. Including failed samples would cause repetitive degenerate outputs (which loop until reaching the maximum generation length) to dominate the average, confounding code length with generation failure.

Tokens are counted using each model's native tokenizer: the Hugging Face `AutoTokenizer` for the Qwen, Gemma, GLM, and fine-tuned models, and `tiktoken` for the OpenAI models. For the Anthropic and Google models, which do not provide public tokenizers, we use their official token-counting APIs. Because tokenization schemes differ across model families, raw token counts are not directly comparable; we therefore report, for each model, the ratio relative to the baseline setting where the original format is specified.

Samples that could not be scored. A very small number of samples rendered successfully but could not be scored because the generated image exceeded the input size limit of the judge model. This reflects a constraint of the evaluation judge rather than a generation failure, so we treat these samples as successful renders in RFR and exclude them when computing mean IA and DQ. Conversely, samples that fail to render are assigned the minimum score of 1.0 for both IA and DQ, and the reported means are computed over all samples in the test set.

C.2 Judge Prompts

To evaluate IA and DQ, we query the judge model with the reference image and the generated image, prompting it to output scores for both criteria in JSON format. We use two distinct prompts depending on the input type. For rendered-image inputs, the judge evaluates visual fidelity to the reference image; for sketch inputs, it uses criteria that do not penalize cleanup or regularization as long as the essential meaning of the diagram is preserved. We use Gemini 3.1 Flash Lite as the judge model, querying it once per sample.

Evaluation prompt for rendered-image input

```
You are an expert judge evaluating diagram alignment and readability.

You are given:
1. The reference diagram image
2. The generated diagram image

Compare the generated diagram with the reference diagram to evaluate how faithfully the
     generated diagram reproduces the reference diagram.

Evaluate based on the two criteria below and return ONLY a valid JSON object with numeric
     scores.

1. alignment (1-5)
   Evaluate how closely the generated diagram matches the reference diagram in terms of
        overall structure, relationships, layout, element placement, labels, and connections
        .
   Note:
   If the alignment falls between two levels, use half-point scores.
   For example, use 3.5 when the alignment is between Fair and Good (approximately 60%
        match), and use 4.5 when the alignment is between Good and Perfect (approximately
        90% match).

   5 (Perfect): The generated diagram matches the reference diagram almost perfectly.
   4 (Good): The generated diagram matches the reference diagram well, with only minor
        inconsistencies (approximately 75% match).
   3 (Fair): The generated diagram partially matches the reference diagram, but several
        noticeable inconsistencies remain (approximately 50% match).
   2 (Poor): The generated diagram matches only a limited portion of the reference diagram,
         with major structural or positional inaccuracies (approximately 25% match).
   1 (Very Poor): The generated diagram does not match the reference diagram at all.

2. diagram_readability (1-5)
   Evaluate how clearly the elements in the generated diagram can be visually identified
        and distinguished, focusing primarily on overlaps and occlusions between diagram
        elements.
   Focus only on visibility issues in the generated diagram, independent of alignment with
        the reference diagram.

   Consider:
```

```
   - overlaps between elements
   - overlaps between text and shapes
   - overlaps between connections and labels
   - occlusions
   - crowded regions
   - cut-off elements that reduce visibility

   Do NOT evaluate layout aesthetics, spacing quality, geometric refinement, or overall
        visual organization in this criterion unless they directly cause visibility problems
        .

   5 (Perfect): All elements are clearly distinguishable and readable, with no meaningful
        overlaps, occlusions, or visibility issues.
   4 (Good): Minor overlaps or occlusions exist, but all important elements remain clearly
        identifiable and readable.
   3 (Fair): Noticeable overlaps or occlusions affect readability in some regions, but the
        diagram remains understandable overall.
   2 (Poor): Major overlaps or occlusions make many elements difficult to distinguish or
        read.
   1 (Very Poor): Severe overlaps, occlusions, or visibility issues make the diagram
        largely unreadable or visually confusing.

Important Notes for Evaluation:
- Both alignment and diagram_readability scores may use 0.5 increments (e.g., 3.5 or 4.5).
     If a score falls between two levels, use the half-point value.
- Invalid Generated Diagram: If the generated output is not a diagram, please rate both
     criteria as 1 (Very Poor). Examples include a completely blank (white) image and a
     text-only image.
- Whitespace/Margins: If the diagram is present but surrounded by whitespace or margins,
     ignore the whitespace and evaluate only the diagram content itself.
- Cropped or Cut-Off Diagrams: If the diagram appears cropped or cut off (missing elements
     at edges), evaluate this under the diagram_readability criterion, as it directly
     affects visibility.

Return ONLY valid JSON (no markdown, no explanation):
{"alignment": <1-5, half-point increments allowed>, "diagram_readability": <1-5, half-point
      increments allowed>}
```

Evaluation prompt for sketch input

```
You are an expert judge evaluating sketch-to-diagram conversion quality.

You are given:
1. The original sketch diagram image
2. The generated diagram image

The goal is not to reproduce the sketch exactly as a hand-drawn image.
Instead, the goal is to convert the sketch into a clean, well-organized, machine-drawn
     diagram while preserving the essential diagram content and relationships.

Compare the original sketch diagram and the generated diagram to evaluate how well the
     generated diagram preserves the essential structure, relationships, layout, element
     placement, labels, and connections of the original diagram while presenting it in a
     clean diagram form.

Evaluate based on the two criteria below and return ONLY a valid JSON object with numeric
     scores.

1. alignment (1-5)
   Evaluate how well the generated diagram preserves the essential structure, relationships
        , layout, element placement, labels, and connections of the original diagram.
   Reasonable cleanup, regularization, spacing adjustment, geometric correction, and layout
         refinement should NOT be penalized if the essential diagram meaning is preserved.

   Note:
   If the alignment falls between two levels, use half-point scores.
   For example, use 3.5 when the alignment is between Fair and Good (approximately 60%
        match), and use 4.5 when the alignment is between Good and Perfect (approximately
        90% match).

   5 (Perfect): The generated diagram preserves the original diagram almost perfectly in
        terms of essential structure, relationships, layout, labels, and connections.
   4 (Good): The generated diagram preserves the original diagram well, with only minor
        inconsistencies (approximately 75% match).
   3 (Fair): The generated diagram partially preserves the original diagram, but several
        noticeable inconsistencies remain (approximately 50% match).
```

```
   2 (Poor): The generated diagram preserves only a limited portion of the original diagram
        , with major structural, positional, or relational inaccuracies (approximately 25%
        match).
   1 (Very Poor): The generated diagram fails to preserve the essential structure or
        meaning of the original diagram.

2. diagram_quality (1-5)
   Evaluate the overall visual quality, readability, and organization of the generated
        diagram as a clean machine-drawn diagram.
   Focus only on the generated diagram itself, independent of its alignment with the
        original sketch diagram.

   Consider:
   - readability and legibility
   - visibility of elements and text
   - absence of overlaps or occlusions
   - clean and consistent lines
   - regularized shapes
   - balanced spacing
   - proper alignment
   - organized layout
   - visually stable connections
   - overall visual coherence

   5 (Perfect): The generated diagram is highly readable, visually clean, well-organized,
        and professionally structured, with consistent geometry, spacing, and layout.
   4 (Good): The generated diagram is mostly clean and readable, with only minor visual or
        organizational issues.
   3 (Fair): The generated diagram is understandable overall, but contains noticeable
        readability, spacing, alignment, or organization issues.
   2 (Poor): The generated diagram is difficult to read or visually messy due to major
        layout, spacing, overlap, or consistency issues.
   1 (Very Poor): The generated diagram is severely disorganized, unreadable, or visually
        corrupted.

Important Notes for Evaluation:
- Both alignment and diagram_quality scores may use 0.5 increments (e.g., 3.5 or 4.5). If a
      score falls between two levels, use the half-point value.
- Invalid Generated Diagram: If the generated output is not a diagram, please rate both
     criteria as 1 (Very Poor). Examples include a completely blank (white) image and a
     text-only image.
- Whitespace/Margins: If the diagram is present but surrounded by whitespace or margins,
     ignore the whitespace and evaluate only the diagram content itself.
- Cropped or Cut-Off Diagrams: If the diagram appears cropped or cut off (missing elements
     at edges), evaluate this under the diagram_quality criterion.

Return ONLY valid JSON (no markdown, no explanation):
{"alignment": <1-5, half-point increments allowed>, "diagram_quality": <1-5, half-point
     increments allowed>}
```

C.3 HUMAN EVALUATION

Sampling. From the outputs generated under the original-format setting ($F = \{f_{\mathrm{src}}\}$), we randomly sampled 100 evaluation items in total: 17 each for SVG, TikZ, Graphviz, and Mermaid, and 16 each for draw.io and PlantUML. Each item consists of one reference image of a single diagram together with three generated images, one drawn from each of three model groups: proprietary models, open-weight models with fewer than 10B parameters, and open-weight models with 10B or more parameters. The evaluation therefore covers 300 generated images in total. The presentation order of the generated images was randomized, and model identities were blinded to the annotators.

Annotators and protocol. Annotators were recruited through crowdsourcing. Prior to the main evaluation, we conducted a screening round to filter out inattentive or low-effort workers. In this screening, we inserted trap items in which one of the three generated images was replaced with a reference image of an unrelated diagram. Because reference diagrams are visually clean and well-formed, DQ remains high while IA drops to the minimum, ensuring that the two criteria operate independently. Only workers who assigned the minimum score of 1.0 for IA to these trap items passed, yielding a pool of 52 qualified workers out of 117 participants.

The main evaluation was made available exclusively to these 52 workers, and 26 of them completed all 500 assignments, compensated at $2.25 per assignment. Each item was independently evaluated by five annotators.

Annotation interface. Figures 7–10 illustrate the interface presented to annotators. It consists of a task overview, rating criteria for the two metrics, rating guidelines, a worked-example block, and the rating task itself. In the worked-example block, the sliders are locked at reference benchmark scores assigned by the authors to one reference image and three generated images, allowing annotators to calibrate their judgment in advance. In the task section, the reference image and the three generated images are displayed side by side, and annotators adjust the IA and DQ sliders for each image. All sliders are initialized to 3.0, yielding six ratings per item.

Aggregation and inter-annotator agreement. The final human score for each item is the trimmed mean of the five ratings, calculated by removing the highest and lowest scores and averaging the middle three. Table 8 reports the inter-annotator agreement (Krippendorff's α for ordinal data).

Table 8: Inter-annotator agreement in the human evaluation. For image-level alignment (IA) and diagram quality (DQ), we report Krippendorff's α with the difference function for ordinal data. The evaluation covers 300 samples, each rated independently by five annotators. "Middle three" denotes the ratings after removing the highest and lowest scores among the five annotators.

	All five	Middle three
IA	0.701	**0.872**
DQ	0.570	**0.792**

Correlation with the judge models and model selection. Table 9 reports the correlation between the aggregated human ratings and four candidate judge models. Because the rating scale is ordinal and ties are frequent, we adopt Kendall's τ_b as our primary metric, alongside Spearman's ρ and Pearson's r. Gemini 3.1 Flash Lite achieves the highest τ_b on both IA and DQ, demonstrating the most stable and consistent alignment with human judgment across both criteria; we therefore select it as the evaluation judge for this study. GPT-5.6 Luna exhibits comparable performance on IA ($\tau_b = 0.765$) but is lower on DQ, while Gemini 3.5 Flash Lite reaches a similar level on DQ (0.7239 vs. 0.7235) but falls behind on IA. Claude Haiku 4.5 is substantially lower across both metrics.

Table 9: Correlation between the judge models used for automatic evaluation and the human scores. For image-level alignment (IA) and diagram quality (DQ), we report rank correlations (Kendall's τ_b and Spearman's ρ) and a linear correlation (Pearson's r) against the human score, taken as the mean of the middle three annotators. The evaluation covers 300 samples. The best value in each column is shown in bold.

Judge	IA			DQ		
	τ_b	Spearman	Pearson	τ_b	Spearman	Pearson
Gemini 3.1 Flash Lite	**0.765**	0.891	0.878	**0.724**	**0.843**	0.877
Gemini 3.5 Flash Lite	0.744	0.872	0.854	**0.724**	0.833	**0.879**
GPT-5.6 Luna	**0.765**	**0.892**	**0.889**	0.707	0.831	0.847
Claude Haiku 4.5	0.687	0.823	0.810	0.650	0.778	0.816

D Experimental Details

This section provides further implementation details on our experimental setup. Section D.1 details the sampling parameters, maximum generation lengths, and prompt templates used for inference; Section D.2 outlines the fine-tuning procedure and its hyperparameters.

D.1 Prompts and Sampling Parameters

The open-weight models were deployed locally using vLLM (version 0.19.1). The server was launched with `--generation-config auto`, ensuring that the official default generation settings distributed with each model were applied directly. During inference, the client passed only

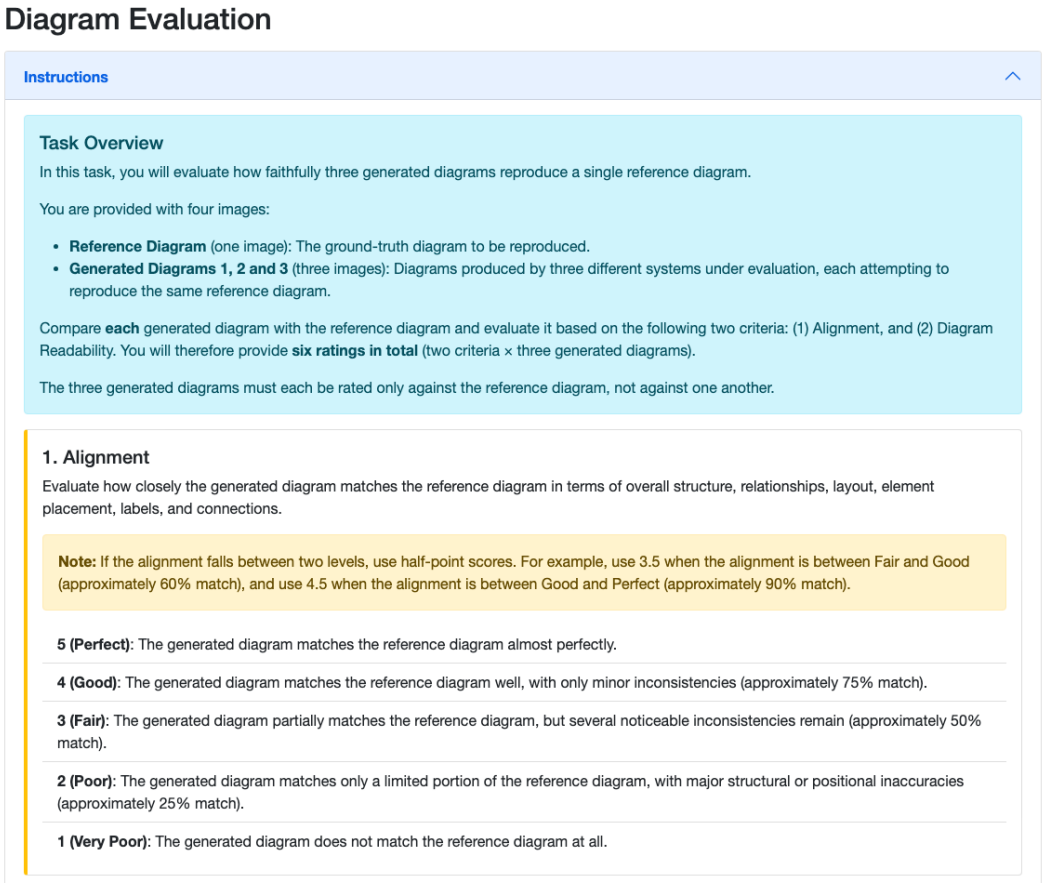


Figure 7: The crowdsourced human evaluation interface (1/4), showing the task overview and the rating criteria for image-level alignment (IA).

a random seed and the maximum sequence length; the resulting sampling parameters are summarized in Table 10. For models accessed via commercial provider APIs, we did not override decoding parameters and relied on the providers' default configurations.

The maximum output length was defined for each output format (Table 11). Because generating internal reasoning traces consumes a substantial token budget when thinking is enabled, we set the limit uniformly to 128,000 tokens for all thinking-enabled settings. An exception was made for GLM-4.5V, whose maximum context window is limited to 65,536 tokens; for this model, we allocated 60,000–65,000 tokens.

In the format-specification setting, we used a system prompt shared across all formats along with a format-specific user prompt.

System prompt (format-specification setting)

```
You are an expert assistant that analyzes a diagram image and generates source code capable
      of reproducing the diagram.

Return a valid JSON object with exactly one field:

{
  "code": "..."
}

Rules:
- The "code" field must contain only the target diagram source code.
- Do not output markdown fences.
- Do not output explanations.
```

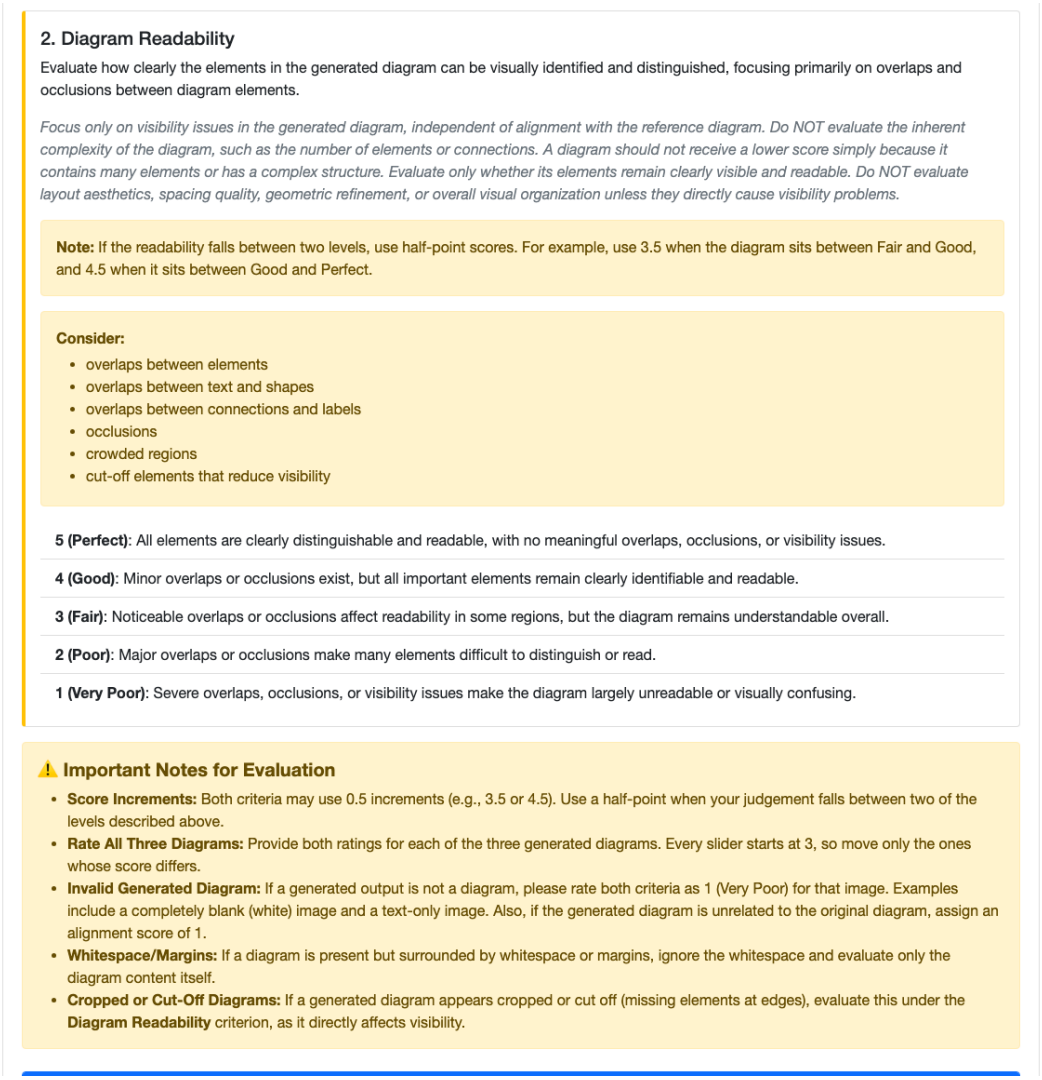


Figure 8: The crowdsourced human evaluation interface (2/4), showing the rating criteria for diagram quality (DQ) together with notes on how to rate.

```
- Do not output additional JSON fields.
- Do not truncate the output.
- Preserve syntactic correctness.
- Prioritize visual similarity to the original diagram.
```

User prompt (Mermaid)

```
Generate Mermaid diagram code that recreates the given diagram.

Requirements:
- The output must parse successfully in Mermaid.
- Use the most appropriate Mermaid diagram type.
- Use Mermaid-native syntax only.
- Preserve nodes, edges, labels, hierarchy, and flow direction.
- Preserve the overall layout as accurately as Mermaid allows.

Example output:

{
  "code": "flowchart TD\nA[Start] --> B[End]"
```

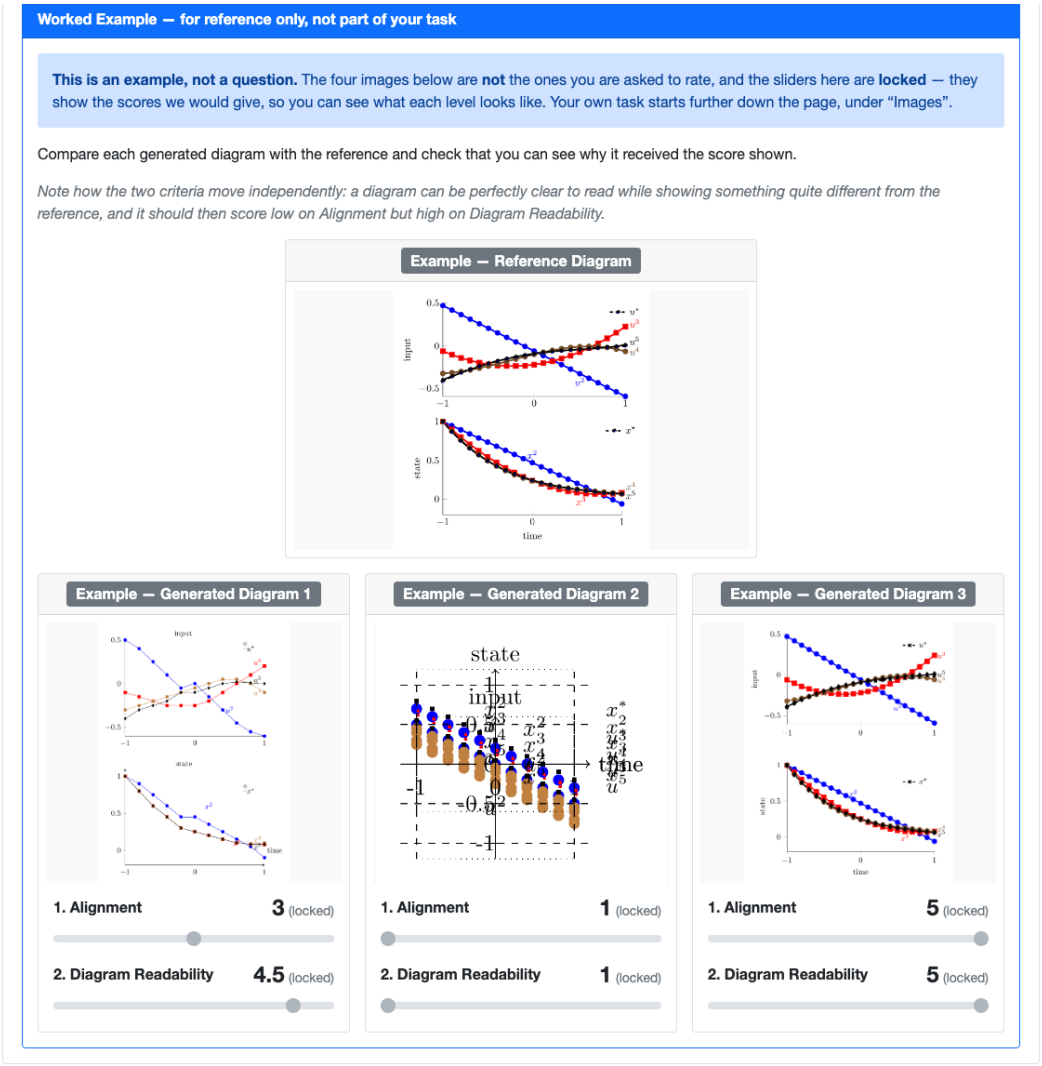


Figure 9: The crowdsourced human evaluation interface (3/4). This block illustrates the criteria with concrete examples; the sliders are fixed and display the reference scores.

```
}
```

User prompt (PlantUML)

```
Generate PlantUML code that recreates the given diagram.

Requirements:
- The output must compile successfully in PlantUML.
- Include both @startuml and @enduml.
- Use the most appropriate PlantUML diagram type.
- Use PlantUML-native syntax only.
- Preserve nodes, edges, labels, grouping, hierarchy, and flow direction.
- Preserve the overall layout as accurately as PlantUML allows.

Example output:

{
  "code": "@startuml\nA -> B\n@enduml"
}
```

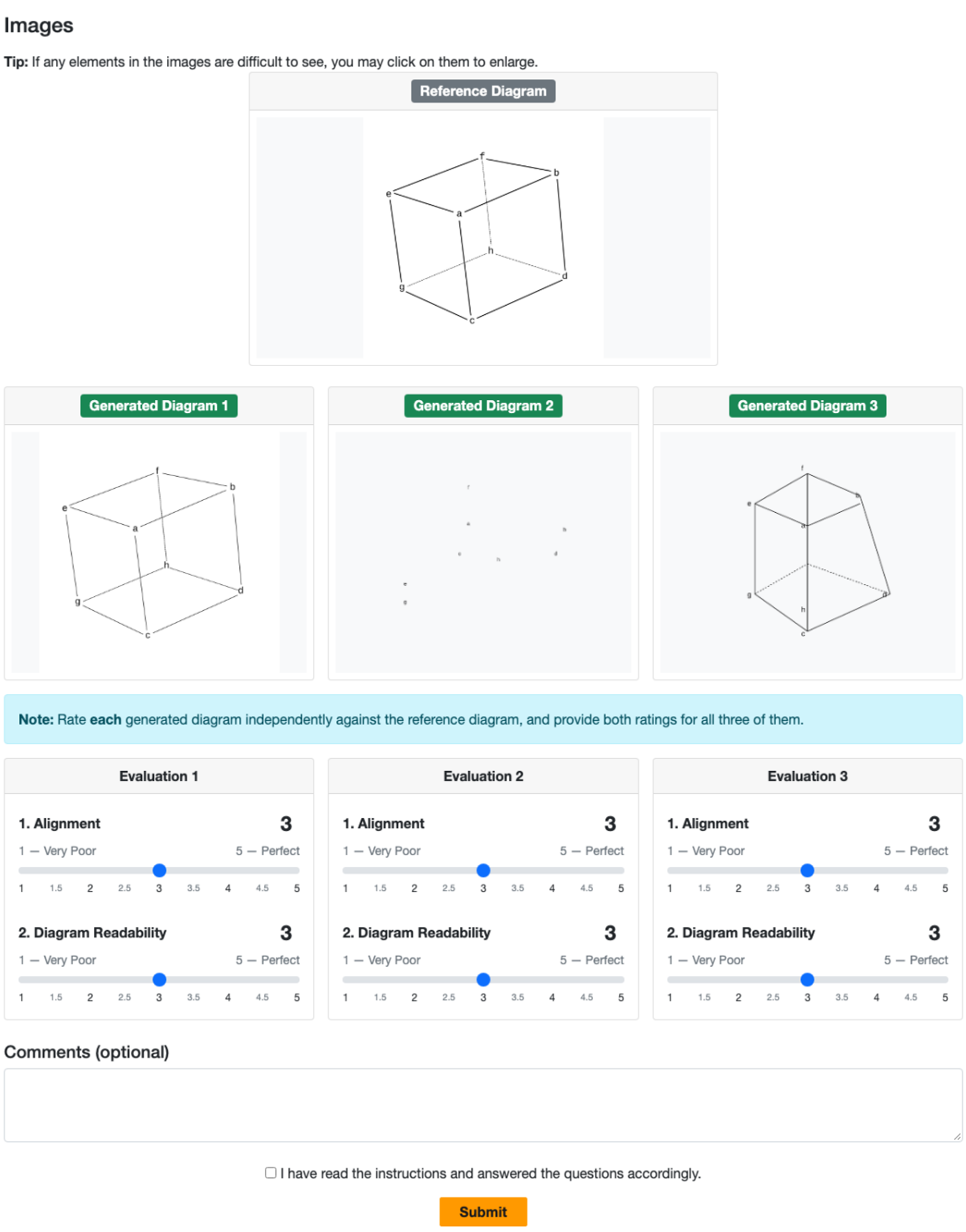


Figure 10: The crowdsourced human evaluation interface (4/4), showing the actual annotation task. For a reference image and three generated images, the annotator operates the IA and DQ sliders. The model identity of each generated image is hidden from the annotator.

User prompt (Graphviz)

```
Generate Graphviz DOT code that recreates the given diagram.

Requirements:
- The output must compile successfully with Graphviz.
- Use valid DOT syntax.
- Use either digraph or graph, whichever best matches the diagram.
- Use Graphviz-native syntax only.
```

Table 10: Sampling parameters used for inference. For models served locally with vLLM, we report the values actually applied; for models accessed through a provider API, we follow the provider defaults. The fine-tuned models use the same settings as their base model, Qwen3-VL-8B-Instruct.

Model	Temperature	Top-p	Top-k	Rep. penalty
Served locally with vLLM				
Qwen3-VL-4B-Instruct	0.7	0.8	20	1.0
Qwen3-VL-4B-Thinking	1.0	0.95	20	1.0
Qwen3-VL-8B-Instruct	0.7	0.8	20	1.0
Qwen3-VL-8B-Thinking	1.0	0.95	20	1.0
Qwen3-VL-30B-A3B-Instruct	0.7	0.8	20	1.0
Qwen3-VL-30B-A3B-Thinking	0.8	0.95	20	1.0
Qwen3.5-4B	1.0	1.0	0	1.0
Qwen3.5-9B	1.0	1.0	0	1.0
Qwen3.5-35B-A3B	1.0	0.95	20	1.0
Gemma-4-E4B	1.0	0.95	64	1.0
Gemma-4-31B	1.0	0.95	64	1.0
GLM-4.5V	1.0	0.0001	1	1.0
GLM-4.6V	0.8	0.6	2	1.0
Fine-tuned model (Ours)				
Qwen3-VL-8B	0.7	0.8	20	1.0
Served through a provider API				
Qwen3.5-397B-A17B	provider defaults			
GPT-5.4	provider defaults			
Claude Opus 4.7	provider defaults			
Gemini 3.1 Pro	provider defaults			

Table 11: Maximum number of output tokens for each setting and format. For both format specification and format selection, we give the limit with thinking enabled and disabled.

Setting	Thinking	Graphviz	Mermaid	PlantUML	draw.io	TikZ	SVG
Format specification	Off	16,000	1,000	2,000	32,000	4,000	32,000
	On			128,000			
Format selection	Off			32,000			
	On			128,000			

```
- Preserve nodes, edges, labels, grouping, hierarchy, and flow direction.
- Preserve the overall layout as accurately as Graphviz allows.

Example output:

{
  "code": "digraph G {\nA -> B;\n}"
}
```

User prompt (TikZ)

```
Generate LaTeX TikZ code that recreates the given diagram.

Requirements:
- The output must compile successfully as a standalone LaTeX document.
- Include all required LaTeX preamble, packages, document environment, and tikzpicture
    environment.
- Use TikZ-native syntax only.
- Preserve nodes, edges, labels, geometry, grouping, hierarchy, and relative positioning.
- Preserve the overall layout as accurately as TikZ allows.

Example output:

{
```

```
  "code": "\\documentclass{standalone}\n\\usepackage{tikz}\n\\begin{document}\n\\begin{
      tikzpicture}\n\\draw (0,0) -- (1,1);\n\\end{tikzpicture}\n\\end{document}"
}
```

User prompt (SVG)

```
Generate SVG markup that recreates the given diagram.

Requirements:
- The output must be valid SVG XML.
- Include a complete <svg>...</svg> document.
- Include the SVG namespace.
- Use SVG-native elements only.
- Preserve nodes, edges, labels, geometry, colors, grouping, hierarchy, and relative
    positioning.
- Preserve the overall layout as accurately as SVG allows.

Example output:

{
  "code": "<svg xmlns=\"http://www.w3.org/2000/svg\"></svg>"
}
```

User prompt (draw.io)

```
Generate diagrams.net/draw.io XML that recreates the given diagram.

Requirements:
- The output must be directly importable into diagrams.net.
- Generate a complete <mxfile>...</mxfile> document.
- Use valid mxGraph XML syntax.
- Preserve nodes, edges, labels, geometry, colors, and styles.
- Preserve the overall layout and hierarchy as accurately as possible.

Example output:

{
  "code": "<mxfile><diagram><mxGraphModel><root><mxCell id=\"0\"/><mxCell id=\"1\" parent
      =\"0\"/></root></mxGraphModel></diagram></mxfile>"
}
```

In the format-selection setting, we provided the model with a list of candidate formats, descriptions of their characteristics, and general selection guidelines. To prevent presentation-order bias, the candidate formats were randomly permuted for each sample. The prompt below illustrates one representative instance; across actual evaluation runs, the candidate formats were shuffled for each sample.

System prompt (format-selection setting)

```
You are an expert assistant that analyzes a diagram image, selects the most appropriate
    representation format, and generates source code that recreates the diagram.
```

User prompt (format-selection setting)

```
Available formats:
- tikz
- plantuml
- mermaid
- drawio
- svg
- graphviz

Selection guidelines:
- Choose the format that best preserves the structure, semantics, and visual layout of the
    diagram.
- Prefer semantic diagram languages when the diagram has clear logical structure.
- Prefer graphical/vector formats when precise positioning or visual fidelity is important.
```

```
- When multiple formats are suitable, prefer the format that produces shorter, clearer, and
     more maintainable code.
- Do not choose a shorter format if it significantly reduces semantic fidelity or visual
    fidelity.

Format characteristics:
- tikz: LaTeX TikZ format for mathematical, academic, geometric, or publication-quality
    technical diagrams.
- plantuml: PlantUML format for UML-style diagrams such as sequence, class, component,
    deployment, activity, and use-case diagrams.
- mermaid: Mermaid format for simple structured diagrams such as flowcharts, sequence
    diagrams, state diagrams, ER diagrams, and mindmaps.
- drawio: diagrams.net (draw.io XML) format for general-purpose diagrams with complex
    layouts, containers, mixed shapes, and arbitrary positioning.
- svg: SVG format for visually complex diagrams where exact positioning and appearance
    matter more than semantic structure.
- graphviz: Graphviz DOT format for graph/network structures, dependency diagrams, trees,
    and directed relationships.

Return a valid JSON object with exactly two fields:

{
  "format": "...",
  "code": "..."
}

Rules:
- The "format" field must be one of:
  "tikz",
  "plantuml",
  "mermaid",
  "drawio",
  "svg",
  "graphviz".

- The "code" field must contain only source code for the selected format.
- Do not output markdown fences.
- Do not output explanations.
- Do not output additional JSON fields.
- Do not truncate the output.
- Do not use placeholder text such as "...".
- Preserve syntactic correctness.
- Prioritize semantic fidelity while maintaining visual fidelity.

Format-specific requirements:
- tikz:
  Generate a complete standalone LaTeX document using TikZ.
  Include all required preamble, packages, document environment, and tikzpicture
      environment.

- plantuml:
  Include both @startuml and @enduml.

- mermaid:
  Generate Mermaid code that parses successfully.

- drawio:
  Generate a complete <mxfile>...</mxfile> document using valid mxGraph XML.

- svg:
  Generate a complete <svg>...</svg> document including the SVG namespace.

- graphviz:
  Generate valid DOT code using either digraph or graph.

Minimal valid examples:
Use these examples only as syntax references.
Do not copy their content unless it matches the input diagram.

- tikz:
{
  "format": "tikz",
  "code": "\\documentclass{standalone}\n\\usepackage{tikz}\n\\begin{document}\n\\begin{
      tikzpicture}\n\\draw (0,0) -- (1,1);\n\\end{tikzpicture}\n\\end{document}"
}

- plantuml:
{
  "format": "plantuml",
```

```
  "code": "@startuml\nA -> B\n@enduml"
}

- mermaid:
{
  "format": "mermaid",
  "code": "flowchart TD\nA[Start] --> B[End]"
}

- drawio:
{
  "format": "drawio",
  "code": "<mxfile><diagram><mxGraphModel><root><mxCell id=\"0\"/><mxCell id=\"1\" parent
      =\"0\"/></root></mxGraphModel></diagram></mxfile>"
}

- svg:
{
  "format": "svg",
  "code": "<svg xmlns=\"http://www.w3.org/2000/svg\"></svg>"
}

- graphviz:
{
  "format": "graphviz",
  "code": "digraph G {\nA -> B;\n}"
}
```

D.2 Fine-tuning Details

For the `Single` setting, we trained the model on all collected rendered-image/code pairs together with 155,635 sketch-style augmented samples derived from them. To create these samples, we converted each code sample to SVG and redrew its graphical elements using Rough.js-style hand-drawn strokes. We then rendered the SVGs as raster images and applied additional perturbations, including background noise and line jitter. Because these perturbations reduced text legibility, we augmented only samples whose original renderings had text above a minimum size threshold. Training was conducted for one epoch on four NVIDIA H100 SXM5 GPUs, each with 94 GB of HBM2e memory. For the `Multi` setting, we initialized the model from the `Single`-trained checkpoint and continued training for an additional 250 steps.

We used `Gemma-4-31B-it` to simplify the SVG training data prior to fine-tuning. Specifically, we prompted the model to reduce code complexity and redundancy while preserving the rendered appearance and essential SVG attributes. The system and user prompts were as follows.

System prompt

```
You are an expert assistant that analyzes a diagram image and generates source code capable
      of reproducing the diagram.

Return only the source code of the target diagram. Do not wrap it in a JSON object or any
     other structure.

Rules:
- Output the diagram source code only.
- Do not output Markdown fences.
- Do not output explanations.
- Do not output JSON or any wrapping structure.
- Do not truncate the output.
- Preserve syntactic correctness.
- Prioritize visual similarity to the original diagram.
```

User prompt

```
Rewrite the given SVG to be as simple as possible.

Guidelines:
- Eliminate unnecessary repetition wherever possible.
- Prefer basic shape elements such as rect, circle, and line over complex path data.
```

```
- Keep the visual appearance as close to the original as possible.

Additional requirements:
- The output must be valid SVG XML.
- Include a complete <svg>...</svg> document.
- Include the SVG namespace (xmlns).
- Preserve the original viewBox, width, height, colors, and layout.

Input SVG:
```xml
{code}
```
```

Table 12 summarizes the LoRA fine-tuning hyperparameters shared by both the single-format and multi-format models.

Table 12: Hyperparameters for fine-tuning with LoRA. Both the model trained with the original format specified and the model trained without specifying a format are fine-tuned from Qwen3-VL-8B-Instruct and share the same hyperparameter values.

Hyperparameter	Value
Base model	Qwen3-VL-8B-Instruct
Tuning method	LoRA
LoRA rank / alpha / dropout	128 / 256 / 0.05
Target modules	all-linear
Learning rate	1×10^{-4}
LR scheduler / warmup ratio	cosine / 0.05
Epochs	1
Batch size $\times$ grad. accumulation	1×16
Max sequence length	32,768
Precision	bfloat16
Vision encoder	frozen
Vision–language aligner	LoRA

E Full Results on Rendered Images

Table 13 presents performance for the models and settings omitted from Table 3 in the main text. Bold values indicate the best performance within this table, rather than across the models in the main text. The overall trends remain consistent with the main text: RFR is highest for PlantUML and draw.io, whereas SVG achieves a near-zero RFR while maintaining lower IA. Enabling thinking improves IA for the Qwen3.5 family, but does not yield noticeable improvements for the Gemma-4 or GLM-V families.

Table 14 reports aggregated performance across output-format settings for the models not covered in Table 4 of the main text. The aggregation methodology follows that of the main table. Models for which cross-format inference was not conducted are omitted. Consistent with the findings in the main text, targeting SVG yields higher IA than the original format while reducing RFR to near zero, and the format-selection setting effectively reduces code length (CL).

Tables 15 and 16 provide a per-input-format breakdown of the format-selection setting, which is summarized under the `Multi` column in Table 4. The former reports task performance (IA, DQ, and RFR), while the latter reports the code length (CL) against the original-format baseline; both tables cover all models. The code length reduction varies substantially across input formats: the ratio falls well below 1.0 for draw.io and SVG inputs, whereas for Mermaid and PlantUML inputs, several models generate longer code instead.

Figures 11–14 present cross-format performance heatmaps when the output format is explicitly specified, grouped by model family. These figures additionally include RFR, which was omitted from the main text due to space constraints. Models without cross-format inference evaluations do not appear in these figures. Taking RFR into account confirms the findings in the main text: targeting SVG maintains a near-zero RFR across all input formats, whereas targeting draw.io incurs the highest failure rate.

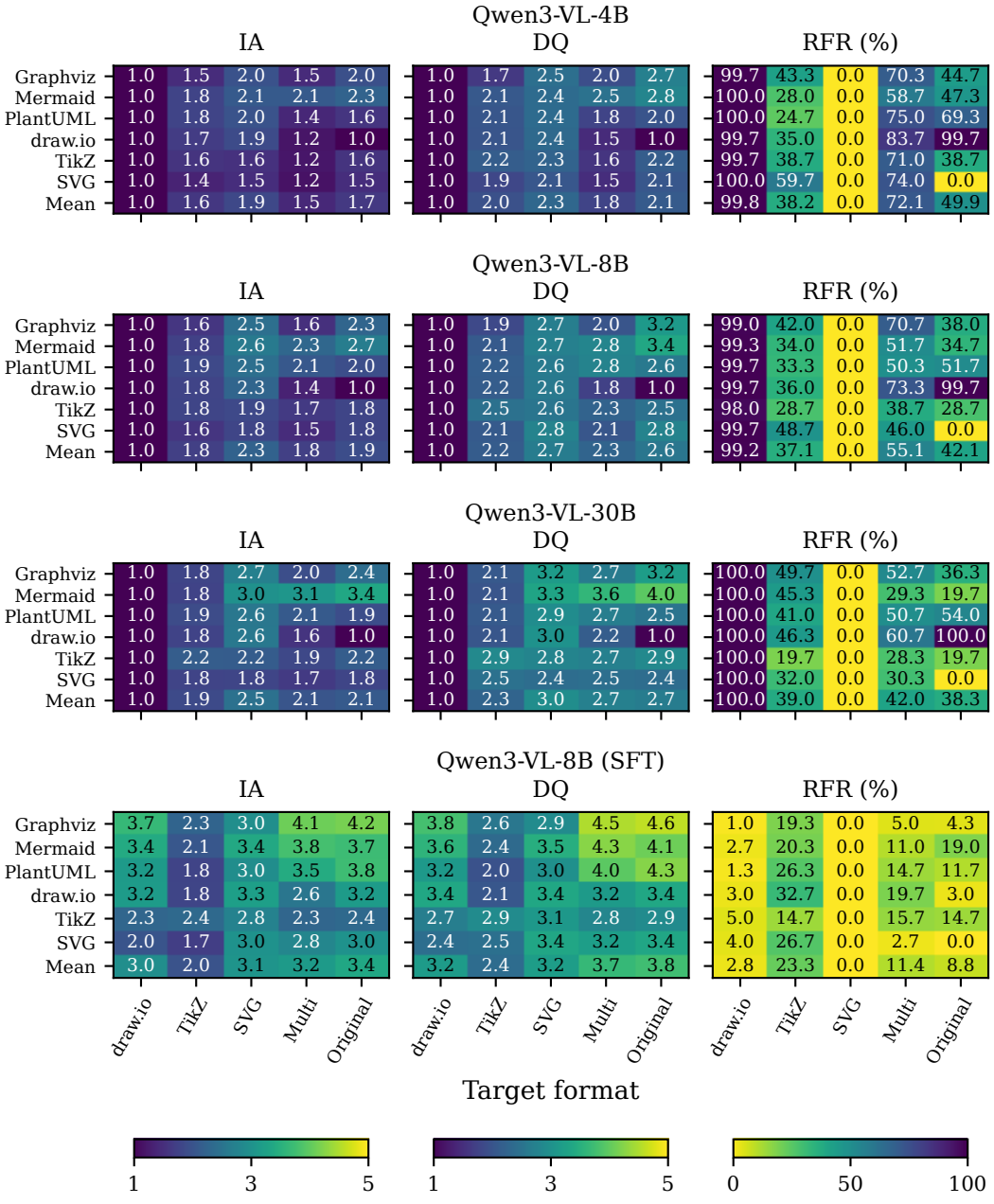


Figure 11: Performance when the output format is specified, for the Qwen3-VL family and the fine-tuned model. From left to right, the columns of panels show image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Within each panel, rows represent the original format of the input diagram and columns represent the specified output format; `Multi` denotes the format-selection setting and Original denotes the original format. The `Mean` row averages over the six input formats. In the Qwen3-VL-8B (SFT) panels, the `Multi` column comes from the model trained without specifying a format and the other columns from the model trained with the original format specified; both were trained from Qwen3-VL-8B-Instruct.

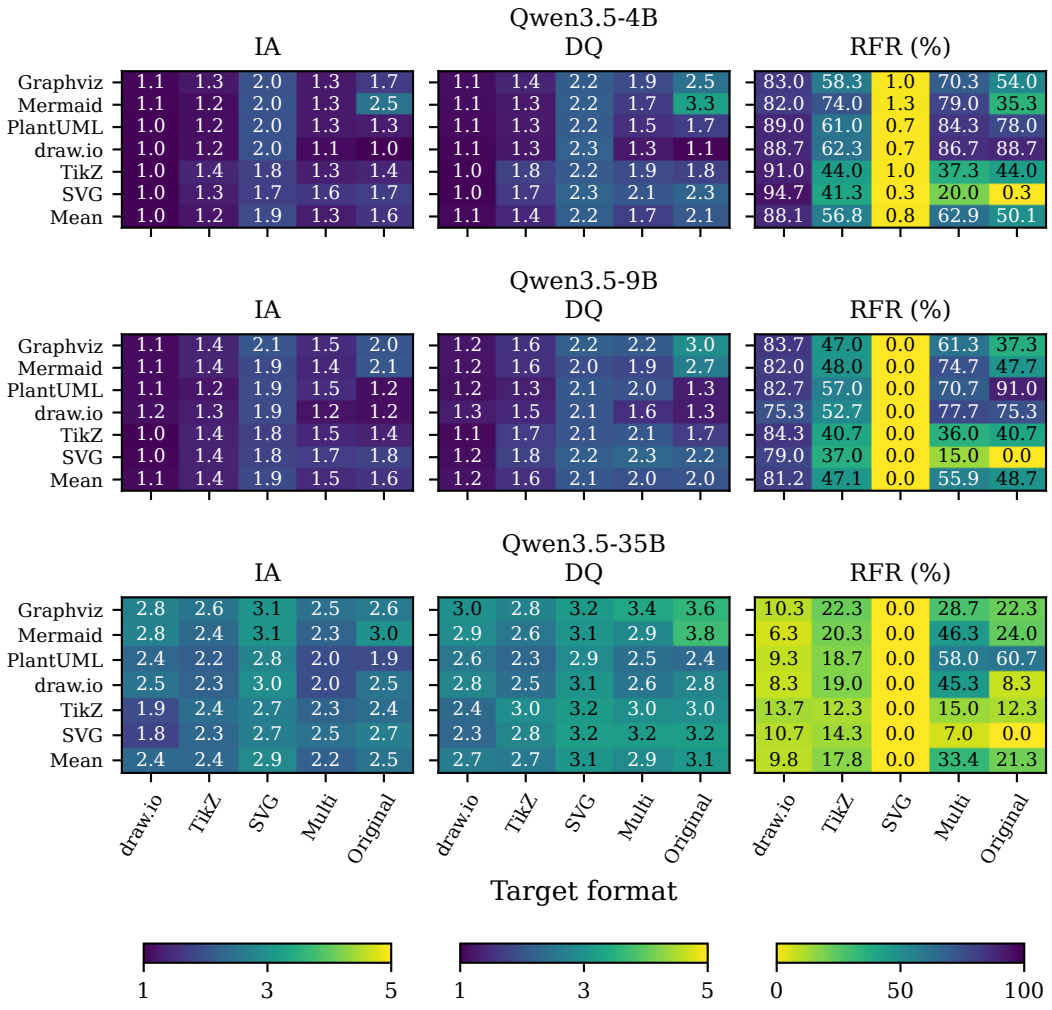


Figure 12: Performance when the output format is specified, for the Qwen3.5 family. From left to right, the columns of panels show image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Within each panel, rows represent the original format of the input diagram and columns represent the specified output format; `Multi` denotes the format-selection setting and Original denotes the original format. The `Mean` row averages over the six input formats.

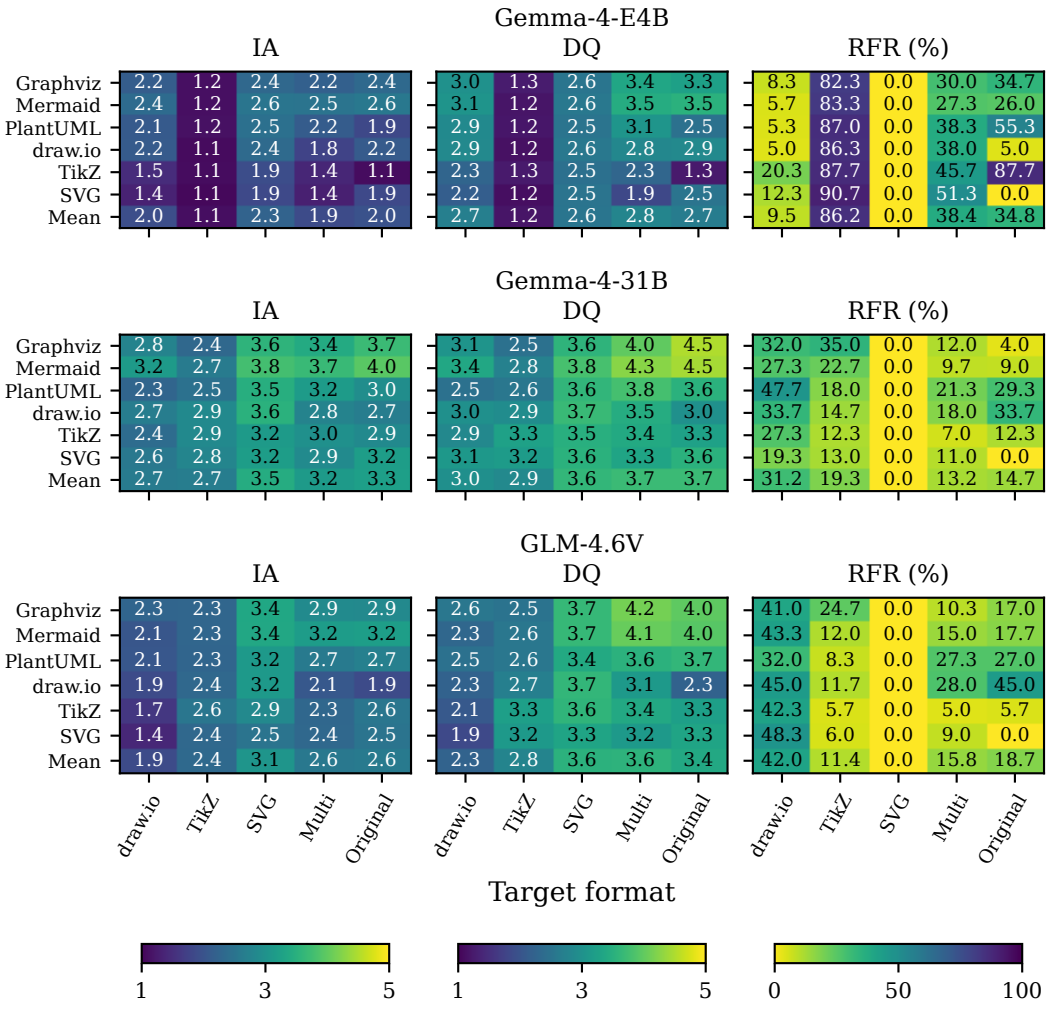


Figure 13: Performance when the output format is specified, for the Gemma-4 and GLM-V families. From left to right, the columns of panels show image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Within each panel, rows represent the original format of the input diagram and columns represent the specified output format; `Multi` denotes the format-selection setting and Original denotes the original format. The `Mean` row averages over the six input formats.

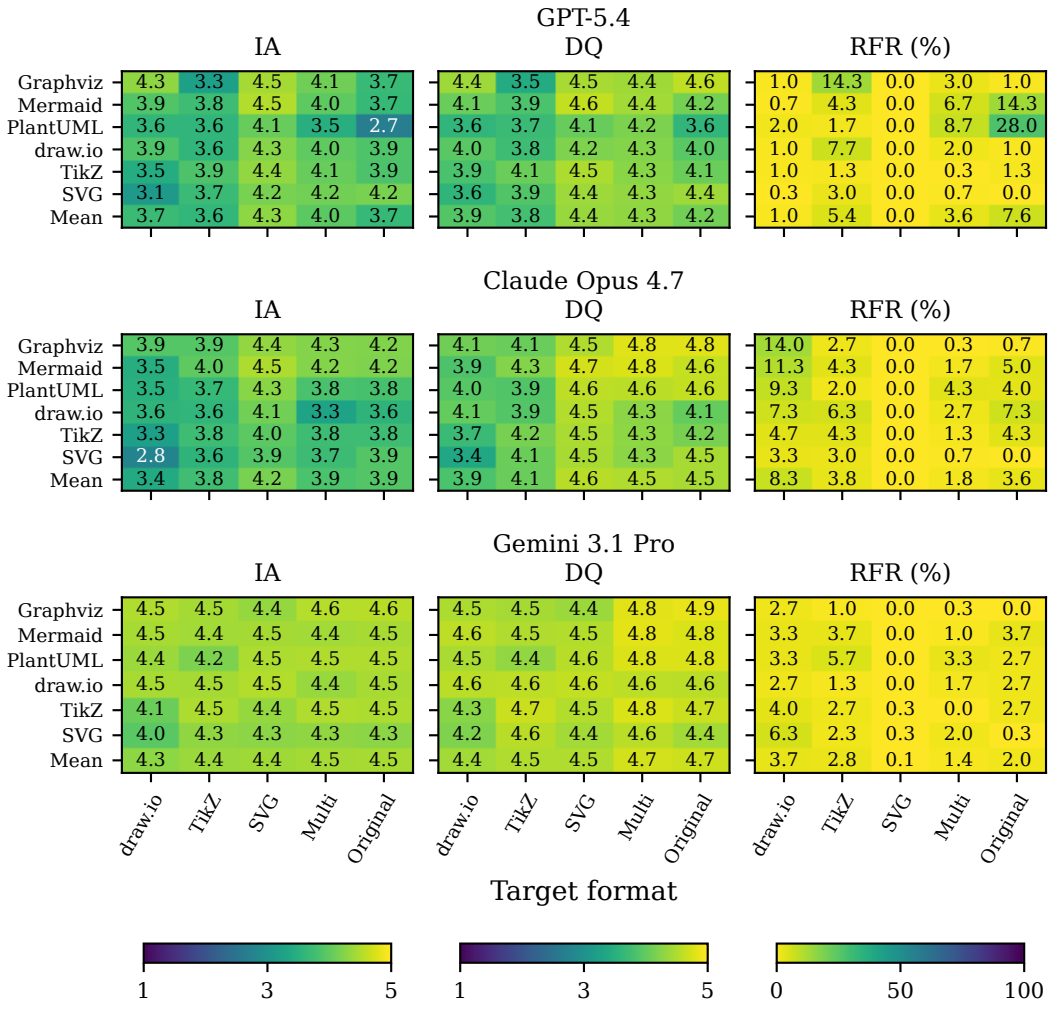


Figure 14: Performance when the output format is specified, for the proprietary models. From left to right, the columns of panels show image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Within each panel, rows represent the original format of the input diagram and columns represent the specified output format; `Multi` denotes the format-selection setting and Original denotes the original format. The `Mean` row averages over the six input formats.

Table 13: Performance with rendered images as input and the original format specified, supplementing Table 3 in the main text. We report image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). This table covers the models and settings not included in the main text. (T) denotes the setting with thinking enabled. The best value in each column is shown in bold.

Model	Graphviz			Mermaid			PlantUML			draw.io			TikZ			SVG		
	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR
Qwen3.5-4B	1.7	2.5	54.0	2.5	3.3	35.3	1.3	1.7	78.0	1.0	1.1	88.7	1.4	1.8	44.0	1.7	2.3	0.3
Qwen3.5-9B	2.0	3.0	37.3	2.1	2.7	47.7	1.2	1.3	91.0	1.2	1.3	75.3	1.4	1.7	40.7	1.8	2.2	**0.0**
Qwen3.5-35B	2.6	3.6	22.3	3.0	3.8	24.0	1.9	2.4	60.7	2.5	2.8	8.3	2.4	3.0	12.3	2.7	3.2	**0.0**
Qwen3.5-397B	4.0	**4.7**	**0.7**	4.1	4.5	7.3	2.4	3.0	43.7	3.0	3.2	20.3	3.1	3.6	7.7	3.4	4.0	0.7
GLM-4.6V	2.9	4.0	17.0	3.2	4.0	17.7	2.7	3.7	27.0	1.9	2.3	45.0	2.6	3.3	5.7	2.5	3.3	**0.0**
Qwen3-VL-4B (T)	2.5	3.7	24.7	2.4	2.9	46.7	1.8	2.6	55.3	1.0	1.0	100.0	1.7	2.3	23.0	1.8	2.4	**0.0**
Qwen3-VL-8B (T)	3.0	4.3	8.0	2.7	3.3	34.0	2.4	3.4	34.7	1.0	1.0	91.3	2.0	2.7	13.7	1.9	2.6	**0.0**
Qwen3-VL-30B (T)	3.2	4.3	8.0	2.9	3.4	35.7	2.1	2.9	46.3	1.4	1.7	57.7	2.5	3.2	5.0	2.3	3.0	**0.0**
Qwen3.5-4B (T)	1.7	2.1	69.0	2.4	2.7	52.0	1.2	1.3	92.0	1.1	1.2	83.0	1.4	1.6	52.0	1.9	2.2	0.3
Qwen3.5-9B (T)	2.8	3.6	25.7	2.7	3.1	41.7	1.4	1.7	79.3	1.5	1.7	51.0	1.9	2.2	34.7	2.2	2.6	**0.0**
Qwen3.5-35B (T)	3.4	4.3	9.0	3.2	3.6	30.7	2.0	2.5	59.0	2.7	3.0	12.0	3.1	3.5	9.7	3.1	3.6	**0.0**
Qwen3.5-397B (T)	4.0	4.5	3.0	3.9	4.3	13.0	2.7	3.4	34.3	**3.7**	**3.9**	5.3	**3.7**	**4.1**	**4.3**	**3.7**	4.1	1.3
Gemma-4-E4B	2.4	3.3	34.7	2.6	3.5	26.0	1.9	2.5	55.3	2.2	2.9	**5.0**	1.1	1.3	87.7	1.9	2.5	**0.0**
Gemma-4-E4B (T)	2.5	3.5	27.3	2.5	3.4	30.0	1.9	2.5	53.0	2.1	2.6	16.7	1.5	1.9	38.3	1.9	2.5	**0.0**
Gemma-4-31B (T)	**4.1**	4.6	1.3	**4.2**	**4.6**	**5.7**	**3.8**	**4.4**	**8.7**	3.5	**3.9**	11.7	3.4	3.9	7.7	**3.7**	**4.2**	**0.0**
GLM-4.5V	3.1	4.2	11.7	2.8	3.4	30.7	2.5	3.4	32.7	1.2	1.3	84.3	2.4	3.1	6.7	2.4	3.4	**0.0**
GLM-4.5V (T)	2.9	4.0	18.3	2.6	3.2	38.3	2.1	2.8	48.3	1.5	1.7	60.0	2.3	2.9	20.3	2.5	3.3	7.7
GLM-4.6V (T)	3.0	4.1	16.3	3.0	3.5	30.7	2.4	3.2	37.0	2.4	2.8	24.3	2.5	2.9	17.3	2.6	3.4	4.3

Table 14: Performance for each output-format setting, with rendered images as input, supplementing Table 4 in the main text. This table covers the models not included in the main text. Each cell averages over the combined test sets of the six input formats, 1,800 examples in total. In addition to image-level alignment (IA), diagram quality (DQ), and rendering failure rate (RFR, %), we report the code length (CL) relative to Original, normalized to 1.0. The best value in each column is shown in bold.

Model	Single															Multi			
	Original			draw.io				TikZ				SVG							
	IA	DQ	RFR	IA	DQ	RFR	CL	IA	DQ	RFR	CL	IA	DQ	RFR	CL	IA	DQ	RFR	CL
Qwen3.5-4B	1.6	2.1	50.1	1.0	1.1	88.1	4.2	1.2	1.4	56.8	0.8	1.9	2.2	0.8	1.9	1.3	1.7	62.9	1.2
Qwen3.5-9B	1.6	2.0	48.7	1.1	1.2	81.2	2.7	1.4	1.6	47.1	0.7	1.9	2.1	**0.0**	1.9	1.5	2.0	55.9	0.7
Qwen3.5-35B	2.5	3.1	21.3	**2.4**	**2.7**	9.8	2.5	**2.4**	2.7	17.8	0.7	2.9	3.1	**0.0**	1.5	2.2	2.9	33.4	0.6
Gemma-4-E4B	2.0	2.7	34.8	2.0	**2.7**	**9.5**	3.7	1.1	1.2	86.2	0.7	2.3	2.6	**0.0**	2.2	1.9	2.8	38.4	0.6
GLM-4.6V	**2.6**	**3.4**	**18.7**	1.9	2.3	42.0	3.5	**2.4**	**2.8**	**11.4**	0.8	**3.1**	**3.6**	**0.0**	2.0	**2.6**	**3.6**	**15.8**	0.6

Figures 15–18 show the distribution of chosen output formats under the format-selection setting, grouped by model family. Across most models, the selected output format rarely coincides with the original input format; this tendency is especially pronounced for SVG and draw.io inputs.

F Full Results on Hand-drawn Sketches

This section presents detailed evaluation results with hand-drawn sketches as input. Because Tables 5 and 6 in the main text already cover all models evaluated on sketch inputs, this section presents figures only, with no additional supplementary tables. The corresponding figures for rendered-image inputs are Figures 11–18.

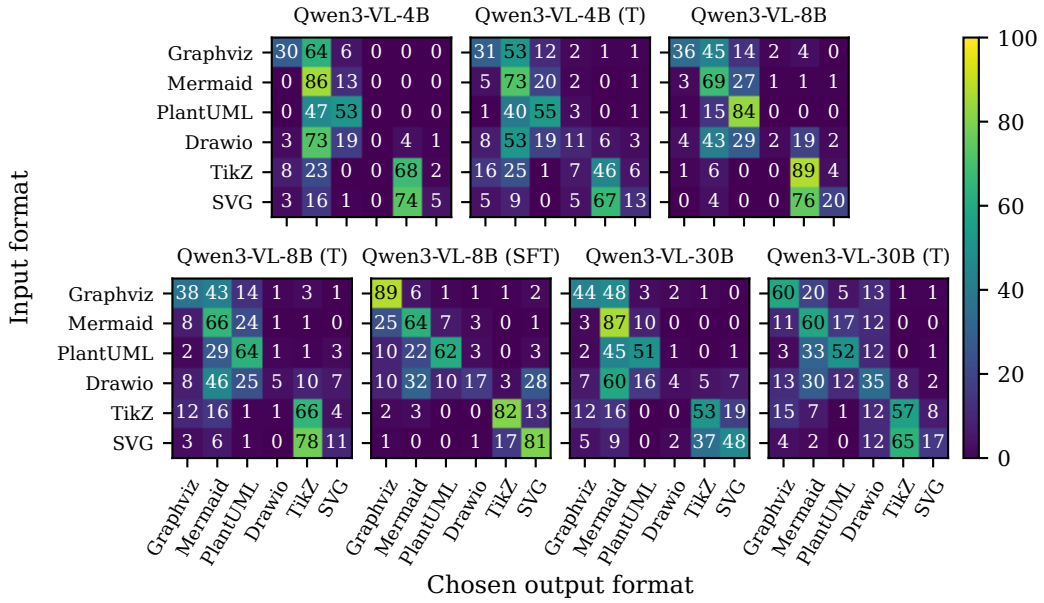


Figure 15: Distribution of the output format chosen under the format-selection setting, for the Qwen3-VL family and the fine-tuned model. Within each panel, rows represent the original format of the input diagram, and columns represent the output format chosen by the model. Each cell indicates the percentage among parsable outputs (%). `(T)` denotes the setting with thinking enabled. Qwen3-VL-8B (SFT) was fine-tuned from Qwen3-VL-8B-Instruct without specifying a format.

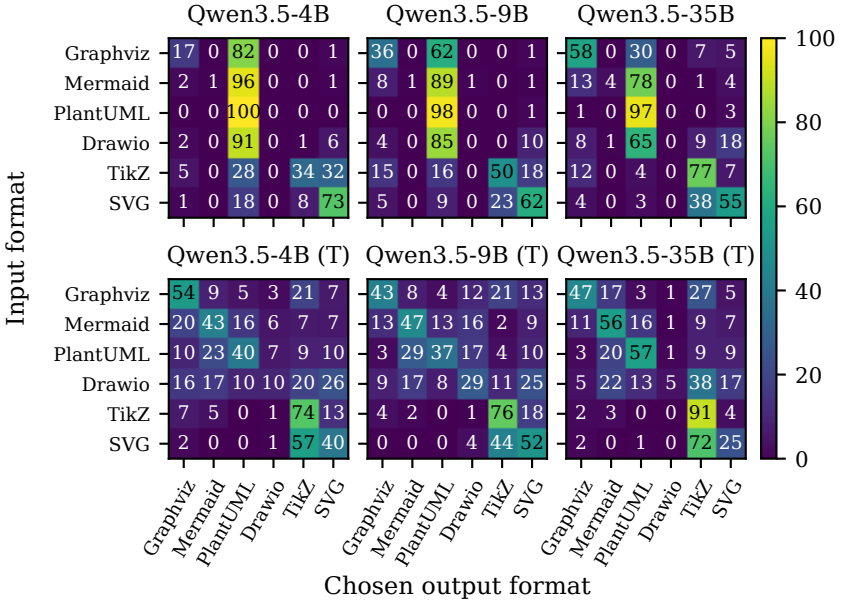


Figure 16: Distribution of the output format chosen under the format-selection setting, for the Qwen3.5 family. Within each panel, rows represent the original format of the input diagram, and columns represent the output format chosen by the model. Each cell indicates the percentage among parsable outputs (%). Panels in the same column are the same model, with thinking disabled in the top row and enabled in the bottom row. `(T)` denotes the setting with thinking enabled.

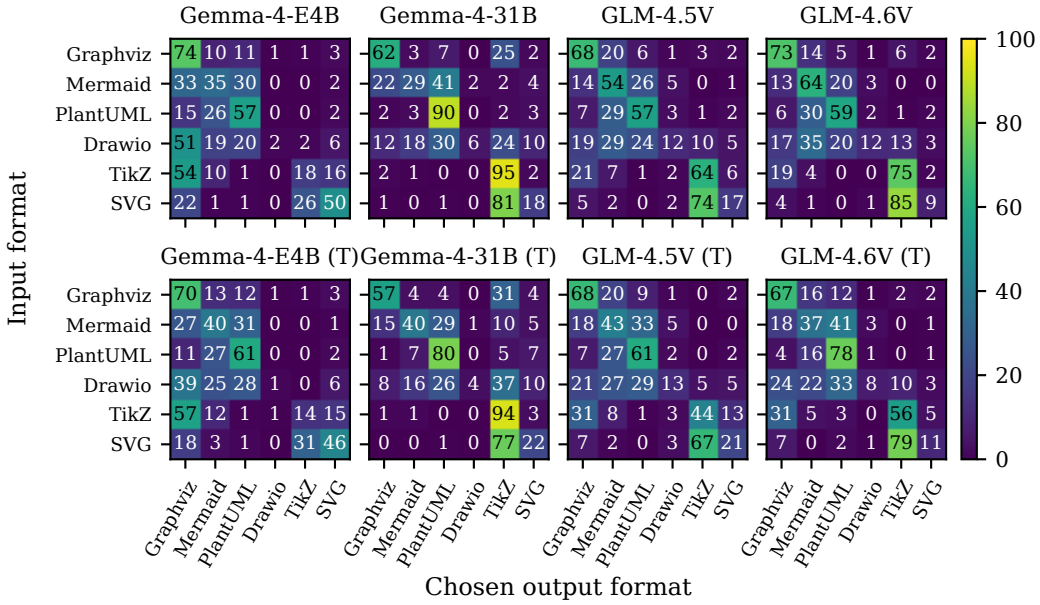


Figure 17: Distribution of the output format chosen under the format-selection setting, for the Gemma-4 and GLM-V families. Within each panel, rows represent the original format of the input diagram, and columns represent the output format chosen by the model. Each cell indicates the percentage among parsable outputs (%). Panels in the same column are the same model, with thinking disabled in the top row and enabled in the bottom row. `(T)` denotes the setting with thinking enabled.

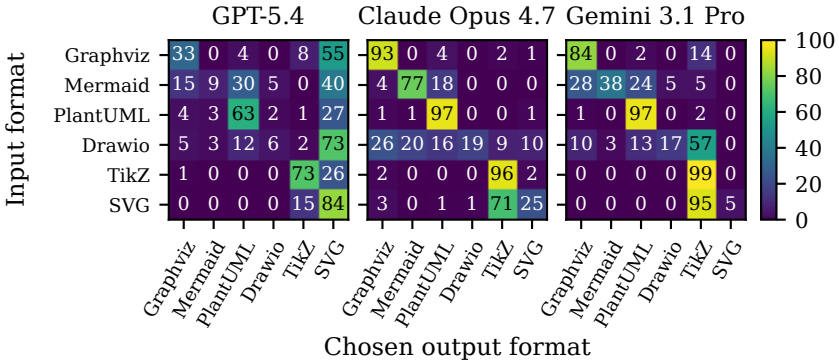


Figure 18: Distribution of the output format chosen under the format-selection setting, for the proprietary models. Within each panel, rows represent the original format of the input diagram, and columns represent the output format chosen by the model. Each cell indicates the percentage among parsable outputs (%).

Table 15: Performance under the format-selection setting, with rendered images as input, broken down by input format. We report image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Columns represent the original format of the input diagram; the output format is selected by the model. Each cell aggregates all 300 samples of that input format, regardless of which output format the model selected. `(T)` denotes the setting with thinking enabled. The best value in each column is shown in bold.

Model	Graphviz			Mermaid			PlantUML			draw.io			TikZ			SVG		
	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR	IA	DQ	RFR
Open-weight models																		
Qwen3-VL-4B	1.5	2.0	70.3	2.1	2.5	58.7	1.4	1.8	75.0	1.2	1.5	83.7	1.2	1.6	71.0	1.2	1.5	74.0
Qwen3-VL-4B (T)	2.0	3.0	42.7	2.2	2.7	54.3	2.1	3.0	41.3	1.5	2.0	64.0	1.4	2.2	43.3	1.6	2.3	32.7
Qwen3-VL-8B	1.6	2.0	70.7	2.3	2.8	51.7	2.1	2.8	50.3	1.4	1.8	73.3	1.7	2.3	38.7	1.5	2.1	46.0
Qwen3-VL-8B (T)	2.4	3.4	31.0	3.0	3.8	24.7	2.5	3.5	27.7	1.8	2.7	39.7	1.9	2.9	16.0	1.9	3.0	9.0
Qwen3-VL-30B	2.0	2.7	52.7	3.1	3.6	29.3	2.1	2.7	50.7	1.6	2.2	60.7	1.9	2.7	28.3	1.7	2.5	30.3
Qwen3-VL-30B (T)	2.5	3.5	26.3	2.8	3.3	34.3	2.3	3.1	36.3	1.9	2.7	35.0	2.1	3.0	17.0	2.1	3.0	13.7
Qwen3.5-4B	1.3	1.9	70.3	1.3	1.7	79.0	1.3	1.5	84.3	1.1	1.3	86.7	1.3	1.9	37.3	1.6	2.1	20.0
Qwen3.5-4B (T)	2.0	2.4	46.0	2.4	2.7	43.7	1.7	2.1	55.0	1.6	2.0	40.7	1.5	1.9	36.0	1.5	1.9	24.0
Qwen3.5-9B	1.5	2.2	61.3	1.4	1.9	74.7	1.5	2.0	70.7	1.2	1.6	77.7	1.5	2.1	36.0	1.7	2.3	15.0
Qwen3.5-9B (T)	2.5	2.9	30.0	2.8	3.2	32.0	2.0	2.6	44.0	2.0	2.4	32.7	2.0	2.4	20.0	2.0	2.4	14.3
Qwen3.5-35B	2.5	3.4	28.7	2.3	2.9	46.3	2.0	2.5	58.0	2.0	2.6	45.3	2.3	3.0	15.0	2.5	3.2	7.0
Qwen3.5-35B (T)	3.3	3.7	18.0	3.3	3.7	23.0	2.6	3.2	33.7	2.5	3.0	24.0	3.0	3.4	10.3	2.9	3.4	7.7
Qwen3.5-397B	3.7	4.3	7.7	3.7	4.2	14.0	3.0	3.8	22.7	2.9	3.5	19.0	3.3	3.8	9.3	3.1	3.7	8.3
Qwen3.5-397B (T)	3.9	4.4	5.0	3.8	4.3	12.0	3.0	3.7	26.3	2.9	3.5	21.3	3.5	3.9	6.0	3.2	3.6	13.7
Gemma-4-E4B	2.2	3.4	30.0	2.5	3.5	27.3	2.2	3.1	38.3	1.8	2.8	38.0	1.4	2.3	45.7	1.4	1.9	51.3
Gemma-4-E4B (T)	2.1	3.3	30.7	2.5	3.4	29.3	2.1	2.9	42.7	1.8	2.7	41.3	1.4	2.1	53.0	1.3	1.8	59.7
Gemma-4-31B	3.4	4.0	12.0	3.7	4.3	9.7	3.2	3.8	21.3	2.8	3.5	18.0	3.0	3.4	7.0	2.9	3.3	11.0
Gemma-4-31B (T)	3.8	4.2	9.7	4.0	4.4	8.3	3.7	4.4	8.0	3.2	3.8	10.0	3.4	3.9	6.0	3.2	3.8	6.7
GLM-4.5V	2.8	4.0	15.0	2.8	3.6	26.3	2.3	3.2	36.3	2.0	2.9	31.0	2.2	3.2	9.0	2.3	3.2	10.0
GLM-4.5V (T)	2.8	4.0	13.7	2.8	3.7	23.7	2.3	3.2	37.7	2.0	3.1	32.3	2.1	3.2	12.7	2.2	3.1	13.7
GLM-4.6V	2.9	4.2	10.3	3.2	4.1	15.0	2.7	3.6	27.3	2.1	3.1	28.0	2.3	3.4	5.0	2.4	3.2	9.0
GLM-4.6V (T)	2.9	4.0	16.3	3.2	4.1	14.7	2.8	3.9	21.0	2.1	3.3	25.3	2.2	3.5	7.0	2.3	3.2	11.7
Proprietary models																		
GPT-5.4	4.1	4.4	3.0	4.0	4.4	6.7	3.5	4.2	8.7	4.0	4.3	2.0	4.1	4.3	0.3	4.2	4.3	**0.7**
Claude Opus 4.7	4.3	**4.8**	**0.3**	4.2	**4.8**	1.7	3.8	4.6	4.3	3.3	4.3	2.7	3.8	4.3	1.3	3.7	4.3	**0.7**
Gemini 3.1 Pro	**4.6**	**4.8**	**0.3**	**4.5**	**4.8**	**1.0**	**4.5**	**4.8**	**3.3**	**4.4**	**4.6**	**1.7**	**4.5**	**4.8**	**0.0**	**4.3**	**4.6**	2.0
Fine-tuned model (Ours)																		
Qwen3-VL-8B (SFT)	4.1	4.5	5.0	3.8	4.3	11.0	3.5	4.0	14.7	2.6	3.2	19.7	2.3	2.8	15.7	2.8	3.2	2.7

Figures 19 and 20 show performance under the format-specified setting. The relative performance across output formats is consistent with that observed for rendered images: generating in SVG achieves the highest IA, whereas targeting draw.io substantially degrades performance.

Figure 21 shows the distribution of chosen output formats under the format-selection setting. Consistent with the results on rendered images, the chosen format does not necessarily coincide with the original input format.

G Qualitative Analysis

This section provides qualitative analysis of the generated outputs. Section G.1 presents examples of model outputs under the original-format-specified setting.

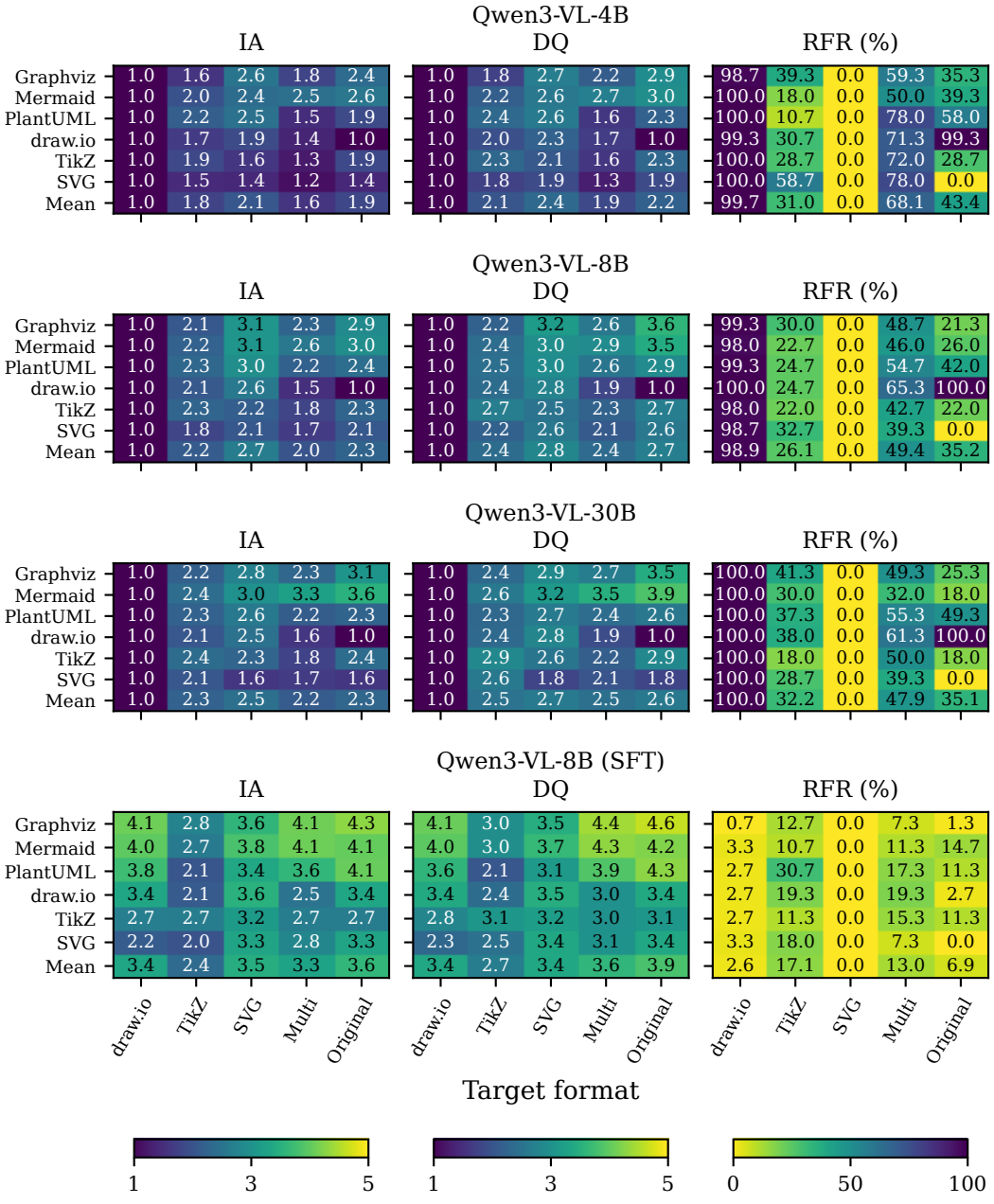


Figure 19: Performance with hand-drawn sketches as input and the output format specified, for the Qwen3-VL family and the fine-tuned model. From left to right, the columns of panels show image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Within each panel, rows represent the original format of the input diagram and columns represent the specified output format; `Multi` denotes the format-selection setting and Original denotes the original format. The `Mean` row averages over the six input formats. In the Qwen3-VL-8B (SFT) panels, the `Multi` column comes from the model trained without specifying a format and the other columns from the model trained with the original format specified; both were trained from Qwen3-VL-8B-Instruct.

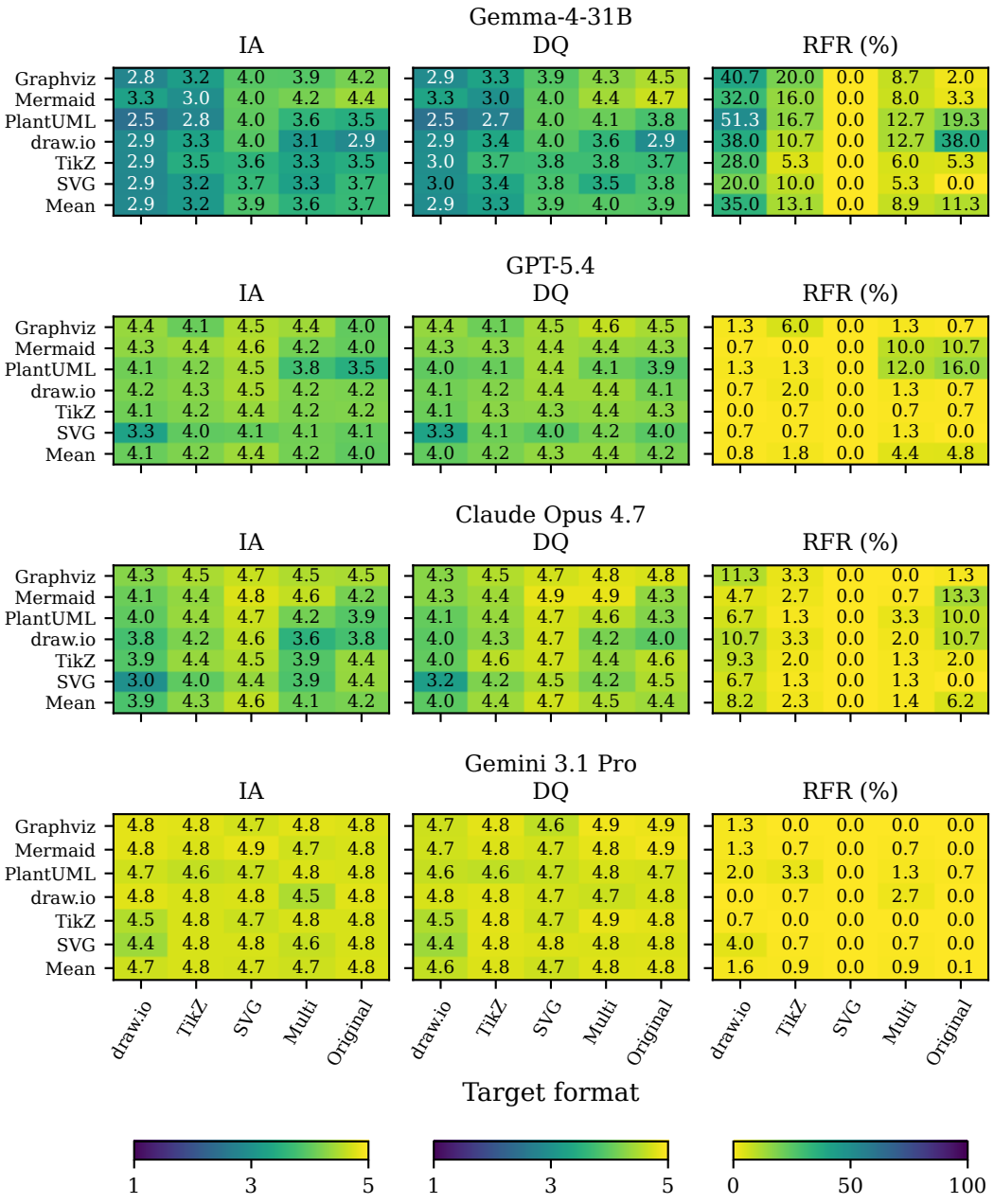


Figure 20: Performance with hand-drawn sketches as input and the output format specified, for Gemma-4-31B and the proprietary models. From left to right, the columns of panels show image-level alignment (IA), diagram quality (DQ), and the rendering failure rate (RFR, %). Within each panel, rows represent the original format of the input diagram and columns represent the specified output format; `Multi` denotes the format-selection setting and Original denotes the original format. The `Mean` row averages over the six input formats.

Table 16: Code length under the format-selection setting relative to the original format, with rendered images as input, broken down by input format. Each cell shows, for a given model, the mean number of output tokens without specifying a format divided by the mean with the original format specified. A value above the baseline of 1.0 means that letting the model select the format produces longer code. The ratio is computed only from samples that produced both valid structured output and a rendered diagram; cells where either setting has no such sample are shown as "—". `(T)` denotes the setting with thinking enabled.

Model	Graphviz	Mermaid	PlantUML	draw.io	TikZ	SVG
Open-weight models						
Qwen3-VL-4B	0.8	1.0	0.7	0.5	0.9	0.4
Qwen3-VL-4B (T)	0.8	1.0	1.1	—	1.0	0.5
Qwen3-VL-8B	0.6	0.8	0.9	—	1.1	0.4
Qwen3-VL-8B (T)	1.0	1.0	1.4	0.4	0.9	0.4
Qwen3-VL-30B	0.7	0.9	1.0	—	1.4	0.5
Qwen3-VL-30B (T)	1.3	2.1	2.6	0.4	1.2	0.5
Qwen3.5-4B	1.0	1.0	0.9	0.2	1.8	0.9
Qwen3.5-4B (T)	2.1	2.8	5.3	0.9	1.1	0.6
Qwen3.5-9B	0.6	0.9	1.9	0.4	1.1	0.7
Qwen3.5-9B (T)	1.8	2.5	4.8	0.5	1.1	0.7
Qwen3.5-35B	1.2	1.4	1.6	0.3	1.2	0.8
Qwen3.5-35B (T)	1.5	1.8	3.2	0.4	1.0	0.6
Qwen3.5-397B	1.1	1.7	1.5	0.3	1.1	0.5
Qwen3.5-397B (T)	1.1	1.4	1.4	0.4	1.0	0.5
Gemma-4-E4B	1.0	1.5	1.1	0.1	1.2	0.7
Gemma-4-E4B (T)	0.9	1.5	1.1	0.2	0.7	0.7
Gemma-4-31B	1.1	1.5	1.3	0.3	1.0	0.5
Gemma-4-31B (T)	1.3	1.6	1.8	0.3	0.9	0.6
GLM-4.5V	1.1	1.3	1.3	0.2	1.0	0.5
GLM-4.5V (T)	1.1	1.2	1.3	0.2	1.0	0.5
GLM-4.6V	1.0	1.4	1.3	0.2	1.1	0.5
GLM-4.6V (T)	0.6	1.0	1.2	0.2	0.9	0.4
Proprietary models						
GPT-5.4	1.6	3.7	3.3	0.6	1.1	1.1
Claude Opus 4.7	1.0	1.1	1.1	0.4	1.0	0.7
Gemini 3.1 Pro	1.1	2.2	1.1	0.4	0.9	0.5
Fine-tuned model (Ours)						
Qwen3-VL-8B (SFT)	1.0	2.1	2.2	0.5	1.3	0.9

G.1 Examples of Model Outputs

Figures 22–25 show examples of model outputs across four formats under the original-format-specified setting. These samples are drawn from the human evaluation subset. For each sample, we report both the human score (the trimmed mean across five annotators after removing the minimum and maximum ratings) and the automatic score predicted by the judge model (`gemini-3.1-flash-lite`).

Proprietary models reproduce the reference diagrams closely across all formats. While output quality degrades as model size decreases, the nature of this degradation varies across formats. In SVG, geometric shapes are lost while only the label text remains; in TikZ, the layout collapses, causing elements to overlap. In draw.io, raw HTML tags appear as literal text rather than being rendered, and in Mermaid, individual elements are preserved but their spatial arrangement and connectivity are altered. In all of these examples, the code renders successfully yet fails to preserve diagram structure, highlighting that RFR alone cannot capture such crucial differences in visual quality.

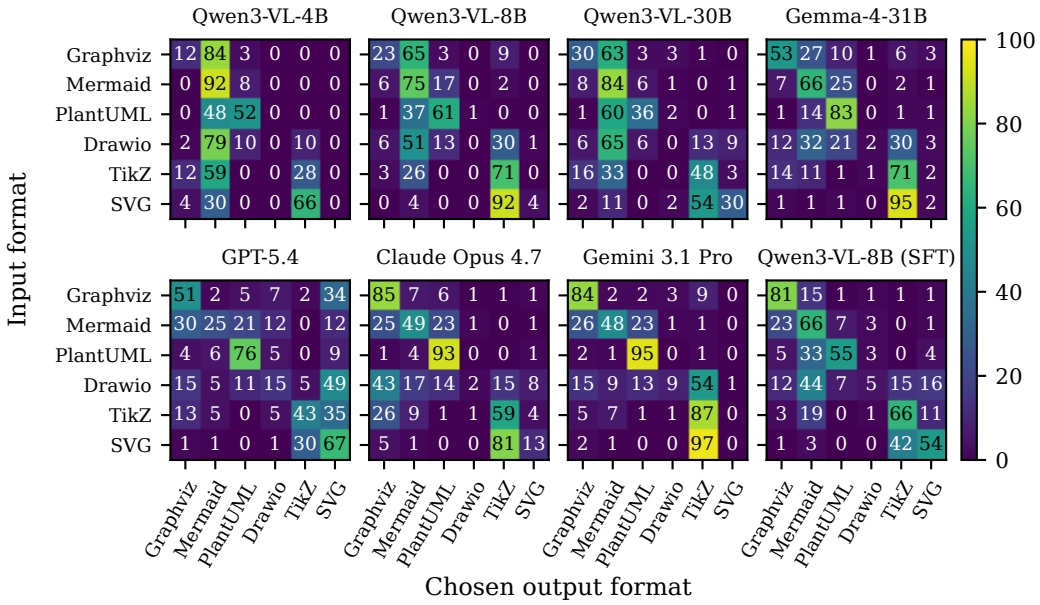
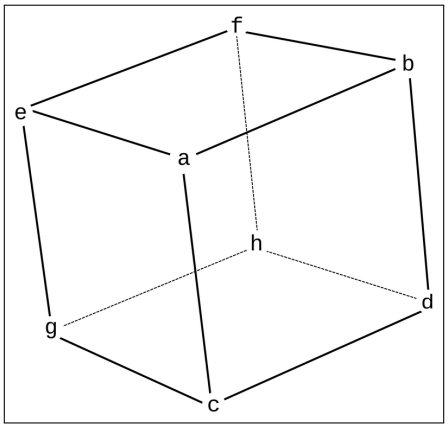
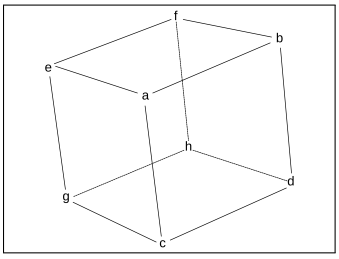


Figure 21: Distribution of the output format chosen under the format-selection setting, with hand-drawn sketches as input. Within each panel, rows represent the original format of the input diagram, and columns represent the output format chosen by the model. Each cell indicates the percentage among parsable outputs (%). This figure covers every model for which sketch results are available.

Reference (SVG)

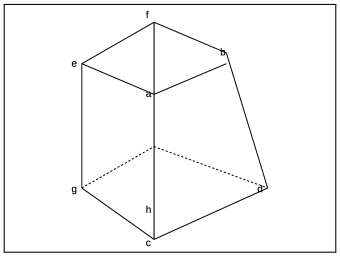


GPT-5.4



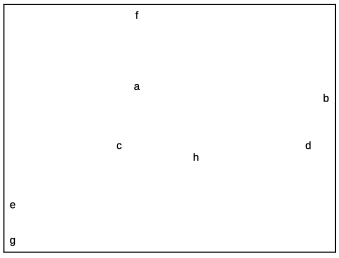
	IA	DQ
Human	5.0	5.0
Auto	5.0	5.0

Qwen3.5-35B-A3B



	IA	DQ
Human	2.8	3.2
Auto	3.0	3.0

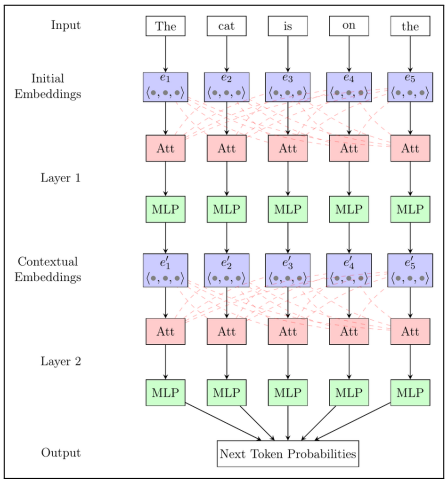
Qwen3.5-9B



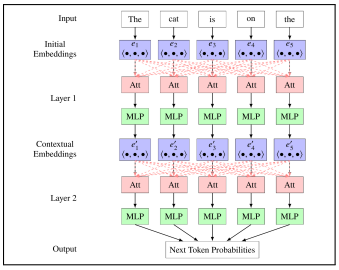
	IA	DQ
Human	1.2	1.5
Auto	1.0	1.0

Figure 22: Example model outputs with the original format specified (SVG). The reference image is a cube with its vertices labeled. The three panels below it show, from left to right, the output of a proprietary model, a larger open model, and a smaller open model. Beneath each panel, scores for image-level alignment (IA) and diagram quality (DQ) are reported from both human evaluation (Human) and the judge model (Auto).

Reference (TikZ)

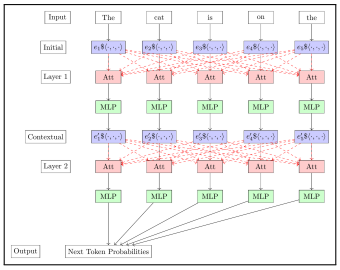


GPT-5.4



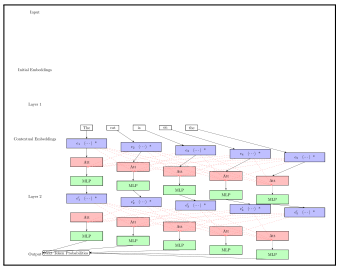
	IA	DQ
Human	4.7	4.7
Auto	4.5	4.5

GLM-4.5V



	IA	DQ
Human	3.7	3.8
Auto	4.0	4.0

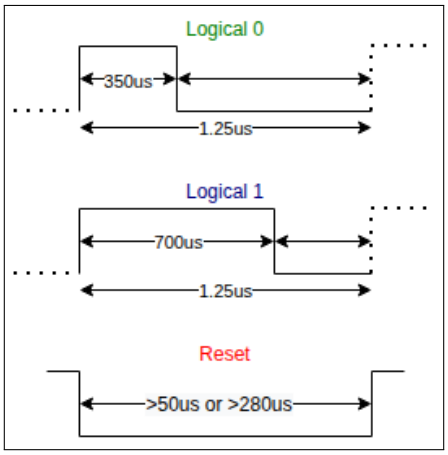
Qwen3.5-4B



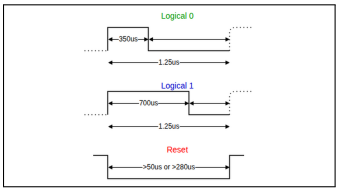
	IA	DQ
Human	2.3	2.3
Auto	2.5	2.5

Figure 23: Example model outputs with the original format specified (TikZ). The reference image illustrates the architecture of a Transformer. The three panels below it show, from left to right, the output of a proprietary model, a larger open model, and a smaller open model. Beneath each panel, scores for image-level alignment (IA) and diagram quality (DQ) are reported from both human evaluation (Human) and the judge model (Auto).

Reference (draw.io)

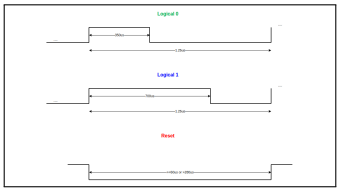


Gemini 3.1 Pro



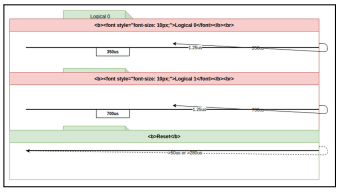
	IA	DQ
Human	4.8	4.8
Auto	5.0	5.0

Gemma-4-31B



	IA	DQ
Human	3.7	3.8
Auto	4.0	4.5

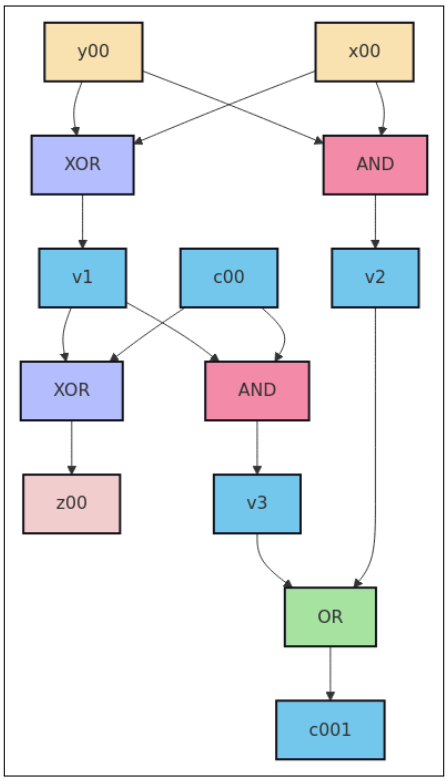
Gemma-4-E4B



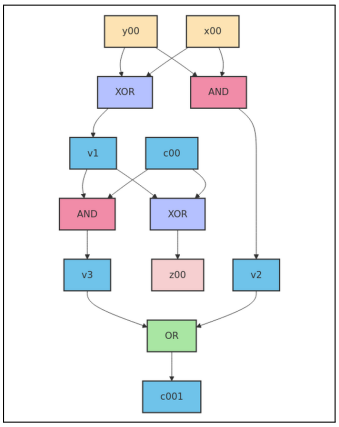
	IA	DQ
Human	1.3	1.3
Auto	1.5	2.0

Figure 24: Example model outputs with the original format specified (draw.io). The reference image is a signal timing diagram. The three panels below it show, from left to right, the output of a proprietary model, a larger open model, and a smaller open model. Beneath each panel, scores for image-level alignment (IA) and diagram quality (DQ) are reported from both human evaluation (Human) and the judge model (Auto). In the smaller open model, HTML tags that should have been rendered as part of the diagram appear verbatim as text.

Reference (Mermaid)

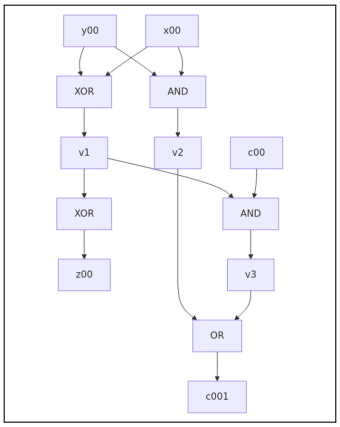


Gemini 3.1 Pro



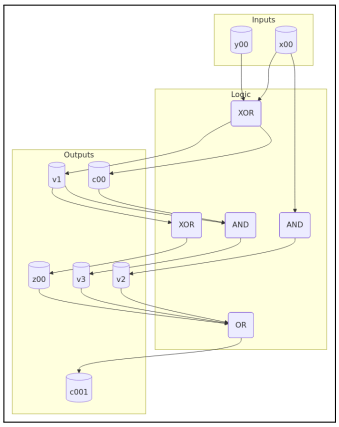
	IA	DQ
Human	4.8	5.0
Auto	2.5	5.0

GLM-4.5V



	IA	DQ
Human	3.7	4.5
Auto	4.0	5.0

Gemma-4-E4B



	IA	DQ
Human	2.0	3.2
Auto	1.5	3.0

Figure 25: Example model outputs with the original format specified (Mermaid). The reference image is a logic circuit. The three panels below it show, from left to right, the output of a proprietary model, a larger open model, and a smaller open model. Beneath each panel, scores for image-level alignment (IA) and diagram quality (DQ) are reported from both human evaluation (Human) and the judge model (Auto). In this example, the judge model rates the proprietary model's IA below that of the open model, reversing the order given by human evaluation.