

## Supplementary Material

This supplementary material provides additional details and analyses for the RLVG approach. It is organized as follows:

- **Section A** presents extended quantitative and qualitative results, including visualizations of SVGs generated from both image and text inputs. These results demonstrate how RLVG outperforms supervised fine-tuning (SVG-SFT) and other strong baselines.
- **Section B** details the experimental setup, including datasets, evaluation metrics, and implementation specifics.
- **Section C** provides further elaboration on the RLVG approach, covering architectural choices, training objectives, implementation techniques, and limitations.
- **Section D** offers a comprehensive overview of related work in SVG generation and reinforcement learning for structured code output.

### Table of Contents

<b>A</b>	<b>Additional Experiments and Results</b>	<b>16</b>
A.1	Qualitative Results . . . . .	16
A.2	Generalization across SVG Benchmarks . . . . .	17
A.3	Ablation Study: Impact of Rewards . . . . .	18
A.4	Cases of Reward Hacking . . . . .	19
<b>B</b>	<b>Experimental Setup</b>	<b>20</b>
B.1	Metrics . . . . .	20
B.2	Datasets . . . . .	22
<b>C</b>	<b>RLVG Method</b>	<b>23</b>
C.1	Multimodal Architecture . . . . .	23
C.2	Reward Details . . . . .	24
C.3	Engineering Tricks for Stable Training of RLVG . . . . .	25
C.4	Limitations . . . . .	25
<b>D</b>	<b>Extended Related Work</b>	<b>25</b>
D.1	SVG Generation . . . . .	25
D.2	Vision-Language Models (VLMs) . . . . .	26
D.3	Reinforcement Learning Post-Training . . . . .	26



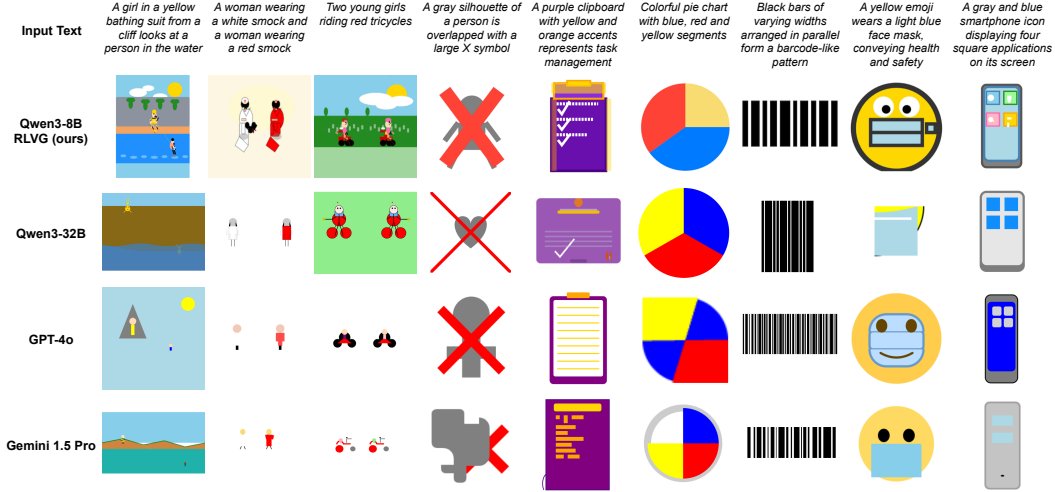


Figure 8: **Text2SVG Results Visualization.** We present qualitative comparisons of our RLVG outputs on test samples from the Text2SVG benchmark, alongside generations from Qwen3-32B, GPT-4o, and Gemini 1.5 Pro. Our method effectively aligns Qwen3-8B to produce high-quality, semantically rich, and visually coherent SVGs.



Figure 9: **Apple Example.** This challenging sample highlights the effectiveness of RLVG. The image contains complex shapes, lighting effects, and semantic cues that require multiple SVG primitives: rectangles for the background, paths for the main structure, and gradients for shadows and highlights. Such examples are not present in the SVG-Stack dataset, which primarily contains simpler icons and logos. While SFT alone fails to reproduce this image, RLVG enables the model to generalize and successfully generate high-fidelity SVGs for out-of-distribution samples like this.

## 558 A.2 Generalization across SVG Benchmarks

559 We tested the generalization capabilities of our trained RLVG models and found that the reinforcement  
560 learning stage equips them with strong out-of-distribution robustness. RLVG models can handle new  
561 visual domains and vector styles that were never seen during training.

562 Table 3 presents results on three held-out datasets (SVG-Emoji, SVG-Fonts, and SVG-Icons) which  
563 were entirely excluded from training. Rows marked “RLVG (ours)” correspond to models first  
564 instruction-tuned on SVG-Stack (1.7M samples), then refined with RLVG using a curated subset of  
565 16k images (SVG supervision is not used in this stage, only rendered rollouts are rewarded).

566 Across all perceptual fidelity metrics (lower MSE and LPIPS, higher SSIM and DINO) **RLVG**  
567 **delivers significant improvements over its own supervised baseline and over all other open-**  
568 **source or commercial VLMs.** For example, on *SVG-Fonts*, the 7B Qwen model improves from 26.5  
569 MSE and 63.9 SSIM to **4.7** MSE and **88.1** SSIM, while shortening the generated SVG by around  
570 1,400 tokens on average. These results show that reward signals derived from rendered images  
571 effectively transfer to unseen domains, even without exposure to those categories during optimization.

**Table 3: RLVG Generalization Results.** We test our RLVG models are evaluated on out-of-distribution datasets that were not used at any point during training. Despite not being exposed to these datasets, the models perform surprisingly well, demonstrating strong generalization capabilities. This is a notable improvement over previous SFT-based approaches, such as the StarVector-1B and 8B models, which only achieved good results when fine-tuned directly on each target dataset. Image processing methods also perform well in these settings, as they are specifically optimized for such domains. In contrast, other VLMs struggle to generalize beyond the SVG distributions seen during their pretraining, resulting in significantly lower performance on these out-of-distribution tasks.

Model	↓ MSE	↑ SSIM	↑ DINO	↓ LPIPS	Code Eff.	↓ MSE	↑ SSIM	↑ DINO	↓ LPIPS	Code Eff.	↓ MSE	↑ SSIM	↑ DINO	↓ LPIPS	Code Eff.
SVG-Emoji						SVG-Fonts					SVG-Icons				
VLMs (Open-Source)															
Qwen2.5VL-32B-Instruct	16.96	57.01	78.78	46.19	-1850	30.44	61.40	78.68	25.76	-1852	17.55	62.14	82.18	30.08	-3864
Qwen2.5VL-72B-Instruct	17.94	58.25	74.38	45.83	-2171	24.76	67.74	81.20	22.69	-1906	24.50	60.97	79.50	26.55	-3975
Llama4-Maverick (400B)	16.25	58.02	78.69	43.90	-2106	22.82	67.35	82.81	22.11	-1869	16.04	68.15	85.86	23.61	-3968
Llama4-Scout (109B)	15.76	58.86	78.69	44.71	-2188	23.75	65.85	80.25	23.25	-1938	13.74	67.76	83.62	25.95	-3948
VLMs (Closed-Source)															
Gemini-Flash-1.5	15.42	59.47	80.31	44.57	-1841	28.41	60.74	81.24	24.84	-1823	18.29	63.10	83.93	27.86	-3827
Gemini-Flash-2.0	15.31	60.95	76.31	44.41	-1866	23.31	65.98	83.88	23.11	-1698	12.28	69.18	87.61	24.37	-3662
Gemini-1.5-Pro	15.93	59.41	78.23	46.05	-1842	27.19	62.92	81.11	24.11	-1770	19.71	63.65	83.09	26.89	-3815
GPT4o-1120	13.44	63.32	81.99	39.62	-2122	20.73	69.13	86.39	21.22	-1806	9.52	74.24	89.82	20.66	-3884
Claude 3.7 Sonnet	11.43	64.95	89.10	35.86	-1828	16.60	73.88	89.77	18.62	-1695	7.83	76.79	93.30	17.57	-3707
Image Processing Methods															
PoTracer	6.70	78.00	88.20	26.70	-9700	0.20	98.80	96.70	0.90	-4200	0.40	97.30	97.20	2.30	-12000
VTracer	0.80	89.40	98.10	7.40	-15700	0.90	88.80	96.40	2.70	-4500	1.70	91.40	94.00	6.20	-20000
PyAutoTrace	1.10	90.20	97.50	7.70	-94000	0.60	96.80	95.40	2.50	-30800	1.40	93.70	94.60	5.30	-56700
DiffVG	3.40	77.60	81.40	24.20	-19700	0.70	95.90	82.10	5.10	-19700	1.50	95.60	95.20	5.60	-19800
LIVE	0.20	95.80	96.90	6.00	-18300	0.10	97.70	95.60	1.30	-18300	0.40	97.30	95.90	3.50	-18200
RLVG Results on SVG Base Models															
StarVector-1B	6.30	82.00	92.90	21.70	-4800	2.20	96.10	97.80	2.20	-2400	2.60	93.10	97.50	4.00	-3500
StarVector-8B	5.20	82.90	94.30	19.30	-6700	2.90	94.60	98.20	3.00	-3000	1.20	97.50	98.40	3.50	-2800
Qwen2.5VL-3B-Instruct	20.25	59.84	68.84	46.32	-2268	25.97	65.27	75.36	22.90	-1939	21.43	65.60	77.41	24.53	-4024
+RLVG (ours)	5.42	72.22	93.25	22.82	-861	5.04	87.42	94.02	7.75	-1574	6.13	82.59	91.20	11.64	-3387
Qwen2.5-VL-7B-Instruct	18.07	60.32	70.93	45.75	-2083	26.52	63.92	77.91	23.79	-1888	13.42	69.41	80.59	25.21	-3859
+RLVG (ours)	4.93	73.87	93.50	21.05	-483	4.73	88.11	93.73	7.36	-1550	7.38	81.64	90.04	12.05	-3111

In contrast, models like StarVector require explicit fine-tuning on each target dataset to achieve comparable scores, as shown by their task-specific tuning on *SVG-Fonts* and *SVG-Icons*.

**Image processing methods still excel at pixel error.** Vectorization pipelines like VTracer and LIVE achieve the lowest MSE values on inputs that are simple single-color glyphs with white backgrounds, where exact pixel reconstruction is relatively easy. However, these methods score lower on perceptual metrics like DINO, which emphasize sharpness and human-perceived fidelity. They also produce SVG code that is orders of magnitude longer and less efficient than any learning-based approach, lacking semantic alignment and practicality for editing or deployment.

**RLVG offers the best balance.** While image processing methods remain strong on raw pixel metrics, RLVG is the only approach that consistently improves pixel fidelity, structural similarity, semantic alignment, and code compactness. By combining reconstruction-based and efficiency-based rewards, RLVG learns to generate SVGs that are visually accurate, semantically meaningful, and compact, making it a strong candidate for real-world deployment where models must generalize to new domains and produce editable, efficient code.

### A.3 Ablation Study: Impact of Rewards

We perform an ablation study on the different reward formulations introduced in Section 3.2. We begin with the Qwen2.5VL-3B model after completing the SVG-SFT stage. At this point, the model is already proficient at SVG generation, producing valid outputs that resemble the input image in simple cases and in some moderately complex ones. However, it still lacks rendering awareness, which is necessary for achieving better pixel-level and semantic alignment. We evaluate several reward configurations, including standard L2 loss, L2-Canny (which focuses on edges only), and their combination. We also test DreamSim alone, DreamSim-Edges, and their combination with Canny, along with the Length reward and other composite configurations that combine multiple signals.

Table 4: **Text2SVG Performance Across Diverse Datasets.** This table shows that RLVG improves Text2SVG performance over the base model (Qwen3-8B-Instruct before RL) across multiple datasets. Although the gains are less pronounced than in Im2SVG, this is partly due to the limitations of standard metrics like CLIP and Aesthetic, which are biased toward natural images and misaligned with SVG-like outputs (see Figure 3 for visual examples). Our proposed Accurate metric, which uses LLMs as judges, more clearly captures the improvements. We also report scores from OmniSVG [Yang et al., 2025], the current state-of-the-art on MM-Icon and MM-Illustration. RLVG models consistently rank second, outperforming all other baselines.

Model	Flickr30k			MM-Icon			MM-Illustration		
	↑ CLIP	↑ Accurate	↑ Aesthetic	↑ CLIP	↑ Accurate	↑ Aesthetic	↑ CLIP	↑ Accurate	↑ Aesthetic
<i>VLMs (Open-Source)</i>									
Qwen2.5VL-32B-Instruct	22.10	2.08	2.07	30.26	3.57	3.21	28.54	3.19	2.80
Qwen2.5VL-72B-Instruct	22.27	1.78	2.13	30.26	3.48	3.22	29.01	3.13	2.85
Llama4-Scout (109B)	21.94	1.92	2.42	30.31	3.56	3.27	29.00	3.30	2.99
Llama4-Maverick (400B)	<b>23.26</b>	<b>2.36</b>	<b>2.51</b>	<b>31.17</b>	<b>4.01</b>	<b>3.54</b>	<b>30.18</b>	<b>3.76</b>	<b>3.25</b>
<i>VLMs (Closed-Source)</i>									
Gemini-Flash-1.5	22.09	1.59	2.16	30.28	3.37	3.23	29.70	3.16	3.05
Gemini-1.5-Pro	23.61	2.24	2.30	30.65	3.37	3.41	29.52	3.49	3.14
Gemini-Flash-2.0	20.89	0.87	0.99	0.30	3.93	3.48	29.50	3.30	2.92
GPT-4o-1120	25.00	2.43	2.48	31.92	4.10	3.57	31.53	3.92	3.32
Claude-3.7-sonnet	<b>27.40</b>	<b>3.37</b>	<b>2.88</b>	<b>32.73</b>	<b>4.62</b>	<b>3.82</b>	<b>32.75</b>	<b>4.30</b>	<b>3.61</b>
<i>Text2SVG Models</i>									
Vectorfusion	-	-	-	27.98	-	-	26.39	-	-
SVGDreamer	-	-	-	29.23	-	-	27.95	-	-
Chat2SVG	-	-	-	30.29	-	-	28.91	-	-
IconShop	-	-	-	23.59	-	-	21.98	-	-
OmniSVG	-	-	-	<b>32.78</b>	-	-	<b>31.64</b>	-	-
<i>RLVG Models</i>									
Qwen3-8B-Instruct	22.50	2.74	2.53	30.09	3.77	3.35	29.05	3.63	3.15
+RLVG(flickr) (ours)	<b>24.42</b>	<b>3.65</b>	<b>2.95</b>	30.20	3.88	3.44	29.06	<b>3.95</b>	<b>3.47</b>
+RLVG(icons) (ours)	22.64	3.12	2.75	<b>30.28</b>	<b>4.13</b>	<b>3.65</b>	<b>29.28</b>	3.95	3.45

Figure 10 presents the progression curves for our reward ablations, while Table 5 summarizes the final scores numerically. All results are evaluated on the SVG-Stack-Hard test set using the metrics DINO Score, MSE, LPIPS, and SSIM. **We observe that DreamSim-based rewards saturate earlier and achieve lower overall scores.** In contrast, *all reward setups that include L2 lead to better performance*, especially in MSE, LPIPS, and SSIM. For DINO Score, the gap is less pronounced, but combinations involving DreamSim yield slightly better results, suggesting that **DreamSim encourages more perceptual or aesthetic alignment rather than precise pixel-level matching.**

Among all configurations, *the combination of all rewards leads to the best overall results.* This setup not only improves scores across metrics but also accelerates convergence. Using L2 alone consistently produces strong MSE performance, as expected from a reward focused on pixel-level accuracy.

#### A.4 Cases of Reward Hacking

We observe several instances where the model learns to exploit the reward function without actually improving output quality, a behavior known as reward hacking. Below, we highlight one of the most notable cases and the corresponding mitigation strategy.

**Small ViewBox Hack.** The model learns to produce SVGs with extremely small viewboxes, for example: `<svg ... viewBox="0 0 1 1">`. This causes the renderer to generate an extremely low-resolution image, and the input image is similarly resized for comparison. As a result, most information is lost, and the image-based reward becomes artificially high.

This issue arises because we originally used the predicted SVG’s viewBox to guide rendering resolution. To fix it, we enforce rendering using the reference image size and aspect ratio derived from the input image, ensuring a fair and stable reward computation.

Table 5: **Ablation Study on Rewards.** Higher DINO and SSIM, and lower MSE and LPIPS indicate better reconstructions. This table shows the effect of different reward formulations. Using L2 achieves a strong MSE score, and L2+Canny further improves it. DreamSim and DreamSim-Edges alone result in poor performance, but their combination with Canny edges yields better outcomes. The best overall performance is achieved when combining all rewards using a weighted sum. We clearly observe a substantial improvement from the baseline (Qwen2.5VL-3B after SVG-SFT, before RL) to the final model trained with RLVG.

Reward(s)	↓ MSE	↑ SSIM	↑ DINO	↓ LPIPS
Baseline (no RL)	7.48	78.40	92.6	17.44
L2	4.77	88.25	95.85	10.90
L2 Canny	4.60	88.30	95.60	10.95
L2 + L2 Canny	4.50	88.25	95.65	10.95
DreamSim	4.90	87.90	95.90	11.30
DreamSim Canny	4.85	87.90	96.00	11.35
DreamSim Canny + L2 Canny + Length	4.75	88.30	96.00	10.80
All rewards (weighted sum)	<b>4.55</b>	<b>88.45</b>	<b>96.00</b>	<b>10.75</b>

**SVG Length Collapse.** In some cases, we observe that the model progressively generates shorter and shorter SVGs until it reaches a collapse point, after which generation diverges entirely and becomes unusable. This behavior is driven by the length deviation reward defined in Equation 6, which continues to incentivize shorter outputs up to half the ground truth length.

The issue arises because lengths below  $\frac{1}{2}L_{\text{gt}}$  still receive increasingly higher rewards, peaking at 1 when exactly half the length is reached. However, this unintentionally encourages the model to keep shrinking the SVG below that threshold.

To address this, we explored assigning a fixed reward of  $-1$  when the predicted length falls below half the ground truth. In practice, we found a more stable solution by reducing the weight of the length reward to 0.1 in early training, then gradually increasing it as RLVG progresses. Once the model has undergone sufficient RL training, it is no longer biased toward producing overly short sequences.

**Text-in-Image Hack.** In the Text2SVG setup, where the model receives a textual instruction and generates corresponding SVG code, we observed a common reward hacking behavior. The model learns to exploit the reward signal by using the `<text>` SVG primitive to render the exact prompt string directly onto the image. This artificially boosts similarity scores, especially when using CLIP-based rewards, since the rendered image containing the input text aligns closely with its textual embedding.

To mitigate this, we preprocess the SVG before rendering and strip out any `<text>` elements or related primitives that visually encode the input prompt. We also strengthen reward robustness by incorporating a Qwen2.5-72B-based evaluator as part of the final reward mix, which helps reduce reliance on shallow visual-textual shortcuts.

## B Experimental Setup

### B.1 Metrics

In addition to the metrics described in Section 4.2, we provide further details specific to the Text2SVG task. We first employ the CLIP Score, following prior work [Wu et al., 2023, Rodriguez et al., 2025b, Yang et al., 2025], along with CLIP Aesthetics. These metrics capture general image-text alignment. However, we observed that they do not correlate well with the visual characteristics of the images generated by our models when RLVG is applied to general-purpose instruction-tuned models such as Qwen2.5VLM.

To address this, we introduce a dedicated Text2SVG Accuracy Score (“Accurate”), which uses a vision-language model (specifically, Qwen2.5VL-70B) as a judge. The VLM is prompted to rate the

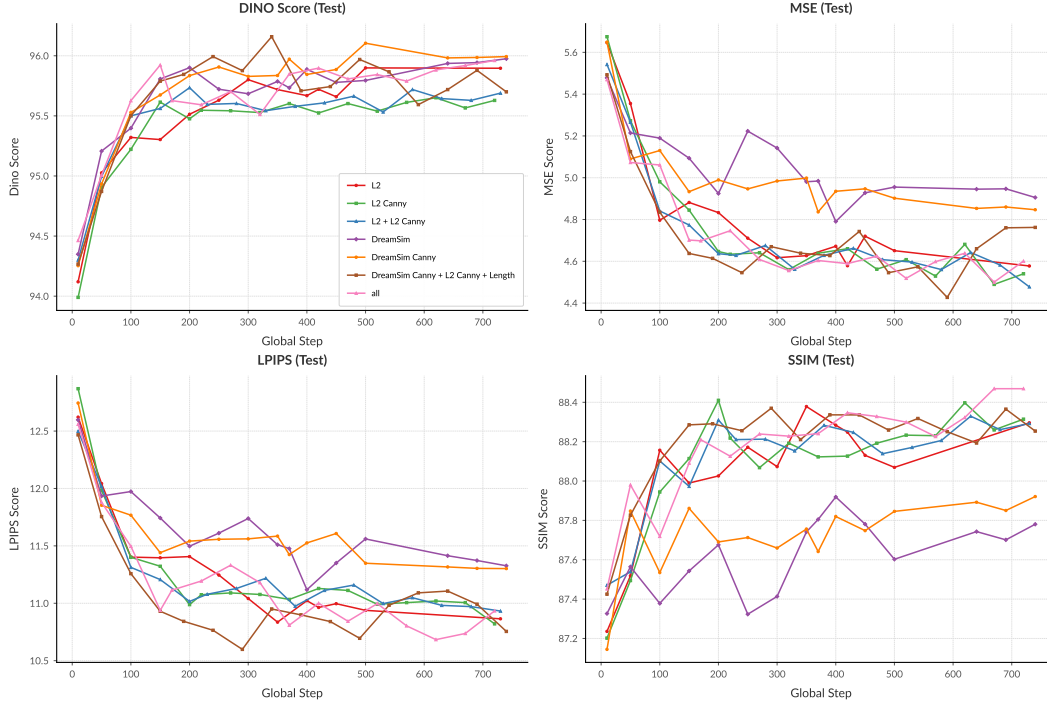


Figure 10: **Ablation on the Impact of Rewards.** We show the evolution of test metrics during RLVG training under different reward configurations. L2 alone achieves strong reconstruction performance, especially in MSE. DreamSim alone performs poorly, but configurations combining DreamSim with other signals tend to perform better. The best overall results come from combining all rewards, which leads to more consistent improvement and a stronger learning signal, while balancing pixel-level accuracy and perceptual alignment.

649 generation along several axes. The prompt used is shown in Prompt 1 and was carefully designed and  
 650 tuned using a held-out validation set to ensure that model-based scores align with human preferences.

**Prompt 1: Used for VLM as a Judge Score (Accurate)**

```
<|im_start|>user<|vision_start|><|image_pad|><|vision_end|> You are an
impartial evaluator of SVG/icon renderings.
----- RUBRIC -----
Alignment Score (0-5) - "Does the image depict what the text describes?"
0 - Completely unrelated: no shared objects, themes, or context.
1 - Very weak match: one minor element overlaps, but overall scene/concept
is different.
2 - Weak match: a few elements overlap, yet key objects or the main action
differ.
3 - Partial match: primary objects/actions align, but notable details or
context differ.
4 - Strong match: image reflects the description with only small,
non-critical discrepancies.
5 - Perfect match: image fully and accurately depicts every essential
detail of the description.
Aesthetics Score (0-5) - "Overall visual quality: clarity of meaning +
first-impression appeal."
0 - Unusable: broken or illegible; no clear subject; chaotic or noisy.
1 - Very poor: subject partly recognizable but ugly-obvious errors,
off-proportion shapes, harsh or clashing colors.
2 - Poor: conveys the subject but feels rough; unbalanced layout, dull/flat
styling, sparse detail.
```

651



Figure 11: **RLVG Improves SVG Generation.** Given an input image (left), we show five SVG generations produced by the model after applying RLVG. By rendering its own predictions and receiving rewards for accuracy and compactness, the model improves over time.

```

3 - Fair:  subject clear at first glance; proportions mostly correct;
acceptable composition and palette with minor flaws.
4 - Good:  rich detail, harmonious colors, balanced negative space; polished
with only subtle imperfections.
5 - Excellent:  instantly communicates its subject; perfect proportions and
composition; refined details, beautiful color harmony-production-ready.
----- TASK -----
Rate the image on both scales above.  Return only this JSON
object-nothing else:
“json { "alignment_score":  <integer 0-5>, "alignment_reason":
"<=100-word justification>", "aesthetics_score":  <integer 0-5>,
"aesthetics_reason":  "<=100-word justification>" } Description:
<|im_end|><|im_start|>assistant

```

652

## 653 B.2 Datasets

654 **Curating SVG Stack** To train our model, we require large-scale data to capture the fundamental  
655 structures of SVG. For this, we leverage the SVG-Stack dataset Rodriguez et al. [2025b], which  
656 consists of SVG files scraped from GitHub. We rasterize these files to obtain paired image-SVG  
657 examples.

658 The original SVG-Stack contains 2.1M samples. We preprocess this data by rounding decimals to two  
659 significant figures, removing XML headers, and filtering out samples with excessively long URLs or  
660 embedded base64 images, which could lead the model to memorize irrelevant content. After cleaning,  
661 we retain 1.7M high-quality examples, which we use to train the model in the SVG-SFT stage, where  
662 it learns to generate SVGs from images as faithfully as possible. The SVGs are rendered using  
663 CairoSVG [Kozee, 2023], and image augmentations follow the protocol introduced by LLaVA Liu  
664 et al. [2023].

665 **SVG-Stack-Hard Test Set** We construct the SVG-Stack-Hard benchmark to address several  
666 limitations of the original SVG-Stack evaluation set, which contains noisy, overly simple, and short  
667 samples. The full test set is also relatively large (3k samples), making evaluation slow, and includes  
668 some broken or empty (white) SVGs.

669 To improve quality and difficulty, we first filter out broken SVGs, white-background SVGs, and  
670 samples with low color entropy. We then retain only SVGs with at least 500 tokens to ensure  
671 complexity. Next, we cluster the remaining samples using DINO image features and perform  
672 stratified sampling to select 600 examples. Finally, we manually verify and curate this set to ensure it  
673 includes visually intricate and moderately challenging samples.



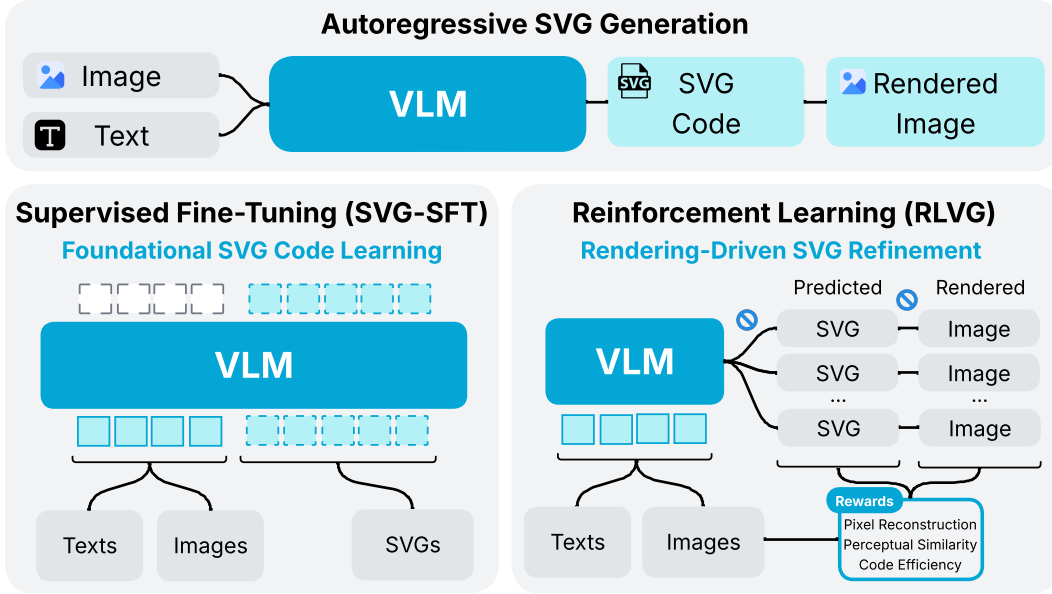


Figure 12: **Overview of Autoregressive SVG Generation and RLVG.** (Above) SVG generation pipeline. An image or a prompt is given to a VLM to produce SVG code, which is then rendered into an image. (Left) Supervised Fine-Tuning. The VLM is trained with teacher forcing to generate the SVG code from the training data. (Right) Reinforcement Learning. Given the same input, the model generates several SVG codes, which are then rendered and a reward is used to update the model.

**Implementation Details** We use the LLaMA-Factory codebase [Zheng et al., 2024] to conduct our supervised fine-tuning (SFT) experiments. For reinforcement learning, including our GRPO-based approach, we build on EasyR1 [Zheng et al., 2025] and VERL [Sheng et al., 2024]. We leverage vLLM [Kwon et al., 2023] for sampling during rollout generation, as it enables highly optimized decoding with high throughput and low latency. This is particularly important for SVG generation, which involves long context sequences.

## C RLVG Method

### C.1 Multimodal Architecture

To perform SVG generation using the autoregressive approach (see Figure 12), we adopt a vision-language model (VLM) [Alayrac et al., 2022, Liu et al., 2023] composed of a decoder-only language model [Brown et al., 2020] that generates SVG code tokens one at a time, and an image encoder that processes visual inputs. In tasks that do not require an image, the image encoder can be omitted, and the model reduces to a text-only LLM. For Im2SVG, the model receives an image as input and generates SVG code as output, treating the code as a plain text sequence in standard SVG format. In contrast, Text2SVG is a purely textual task, where a strong language model alone may be sufficient to generate valid SVGs from text prompts.

**Vision-Language Model Design** VLMs combine an image encoder with a decoder-only language model to process and generate multimodal content. The image encoder, often a Vision Transformer (ViT) [Dosovitskiy et al., 2020] or a convolutional network [Krizhevsky et al., 2012], converts the input image into a sequence of high-dimensional vectors called visual tokens. These are not discrete tokens like in language, but continuous embeddings representing image regions.

To make these embeddings compatible with the language model, they are passed through linear projection layers [Liu et al., 2023]. This allows the language model to attend to the visual context during generation. Since large images can produce many visual tokens, some models add modules to reduce their number and retain only the most relevant ones. For example, Q-Former [Li et al., 2023a] uses learnable queries to extract a smaller set of tokens, while Perceiver Resampler [Alayrac et al.,

2022] and AlignVLM [Masry et al., 2025] use resampling and sparse attention to compress the visual information efficiently.

Once the visual tokens are projected, they are combined with any textual prompt and fed into a decoder-only language model. This model generates output tokens one at a time, using both the image and text context. In our case, the output is SVG code. The model learns to translate visual inputs into structured sequences of SVG instructions that can be rendered to reconstruct the original image. This setup allows the model to reason about both the appearance and structure of visual content and generate precise vector representations.

**Model Choice: Qwen2.5-VL** We perform our experiments using the Qwen2.5-VL [Bai et al., 2025] family of models, which are general-purpose VLMs designed for diverse multimodal tasks. Qwen2.5-VL uses a dynamic-resolution vision transformer (ViT) as the image encoder and a high-capacity language model decoder based on the Qwen2.5 series. This combination allows it to scale to long contexts, adapt to varied image resolutions, and perform complex vision-language reasoning. The vision tokens extracted by the ViT are prepended to the text tokens and jointly processed in the decoder.

**Comparison to StarVector** Compared to StarVector [Rodriguez et al., 2025b], which is specifically designed for SVG generation, Qwen2.5-VL is a more generalist model not specialized for code or vector formats. StarVector employs a CLIP-based image encoder and a code-generation-optimized decoder (e.g., StarCoder) with an adapter to bridge modalities. It is trained directly on SVG-Stack, a large dataset of real-world SVGs, making it more attuned to the syntax, structure, and compositional semantics of SVG code. In contrast, Qwen2.5-VL benefits from broader multimodal training but lacks SVG-specific pretraining, which we address through supervised fine-tuning and reinforcement learning.

**Design Tradeoffs** The generalist nature of Qwen2.5-VL enables strong transfer learning capabilities and flexibility across tasks but makes it initially less precise at SVG generation. Our method, RLVG, fine-tunes Qwen2.5-VL to align its output with visual fidelity and code compactness through supervised fine-tuning (SVG-SFT) reinforcement learning (RLVG). This specialization helps close the gap with task-specific models like StarVector while maintaining broader instruction-following abilities inherent to Qwen2.5-VL.

**Context Length** Context length plays a crucial role in autoregressive SVG generation. Complex SVGs with fine-grained structure and intricate visual elements often require sequences that span tens or even hundreds of thousands of tokens. The Qwen2.5-VL model supports a context length of up to 128k tokens, which allows it to handle long and highly detailed SVG code sequences.

With the growing availability of models that support very long contexts, including recent work pushing towards hundreds of thousands or even millions of tokens [Liu et al., 2025, Meta, 2025], we expect context length to become less of a limitation in the near future. In principle, this trend makes large-scale vector representation increasingly feasible.

In our experiments, we cap the context length at 32k tokens to fit within GPU memory constraints. This is sufficient for all benchmark samples we evaluate, including a wide range of complex SVGs. In comparison, the StarVector model was limited to an 8k token context, which restricts its ability to model large or deeply nested SVG structures.

## C.2 Reward Details

**Text2SVG Rewards.** In addition to the core rewards used for the Im2SVG task, we incorporate image-text alignment signals to better guide the model toward generating text-consistent SVGs. However, we found that CLIP-based rewards (including CLIP Score and CLIP Aesthetics) were not effective in this setting. These models provide weak supervision, as the generated SVGs fall far outside the distribution of natural images on which CLIP was trained. As seen in Figure 8, the SVG outputs follow a distinctive abstract style composed of primitives like rectangles, spirals, and curves, making them poorly suited for CLIP-based evaluation.

To overcome this, we use a vision-language model (VLM) to assess the quality of the generated outputs. Specifically, we prompt Qwen2.5VL-7B (a smaller variant chosen for memory efficiency

during rollout scoring) to judge whether the generated SVG accurately reflects the input text. This VLM-based reward is designed to be more aligned with human preferences and to provide a stronger signal than CLIP. The prompts used for this reward are detailed in Prompts 2 and 3, and include axes for assessing semantic accuracy, visual resemblance, and aesthetic quality.

Prompt 2: Used for VLM as a Judge Reward for Text2SVG (Easy)

```
<|im_start|>user<|vision_start|><|image_pad|><|vision_end|>Does the drawing  
resemble the description: "" [Yes/No]<|im_end|><|im_start|>assistant
```

Prompt 3: Used for VLM as a Judge Reward for Text2SVG (Hard)

```
<|im_start|>users<|vision_start|><|image_pad|><|vision_end|>Does the image  
match the description clearly, accurately, and aesthetically pleasing: ""  
[Yes/No]<|im_end|><|im_start|>assistant
```

### C.3 Engineering Tricks for Stable Training of RLVG

We describe practical strategies that enabled stable and efficient training of our RLVG method.

**Dynamic Max Length.** After the SVG-SFT stage, models often lack strong SVG code completion skills and can struggle with out-of-distribution or complex samples. This occasionally leads to failure cases where the model enters indefinite loops, repeating SVG commands until the maximum token limit is reached. These long and unproductive rollouts slow down RL training, as each rollout must be fully sampled and passed through multiple models.

To mitigate this, we implement a dynamic maximum length schedule. For each batch, we estimate the required output length using the ground truth SVGs and set the maximum length to the longest sample plus a small threshold  $t$ . This strategy significantly reduces rollout overhead early in training, encourages the model to generate shorter and cleaner sequences, and naturally fades in importance as training progresses and generation improves.

During inference, input images are resized using bilinear interpolation such that the shortest side is 512 pixels, preserving the original aspect ratio. We sample with temperature and top-p equal to 0.5 and 0.9 respectively, using the best-of- $n$  strategy: we generate five candidates ( $n = 5$ ) and select the one with the lowest mean-squared error (MSE) relative to the target image.

### C.4 Limitations

As with prior work, our method is constrained by the context length of current models. While this remains a challenge, we note that recent models with extended context windows are beginning to address it. A more specific limitation of our RLVG experiments is the tendency for the model to become increasingly specialized in SVG generation, which may diminish its general instruction-following capabilities. Although straightforward solutions exist, we leave their exploration to future work.

Another key limitation is the inefficiency of GRPO training, which is bottlenecked by rollout generation. This process introduces significant GPU idle time during the RL stage<sup>1</sup>. Mitigating this overhead is an important direction for future optimization.

## D Extended Related Work

### D.1 SVG Generation

SVG generation methods are typically divided into three main categories: classical image processing, latent variable models, and large language model (LLM)-based approaches. Traditional methods, such as VTracer [Vision Cortex, 2023], Potrace [Selinger, Peter, 2024], and Autotrace [Weber,

<sup>1</sup><https://huggingface.co/blog/ServiceNow/pipelinert1>

Martin, 2024], convert raster images into vector graphics by tracing contours and clustering regions. While effective for shape extraction, these methods produce long, unstructured SVGs with raw path commands, resulting in verbose, difficult-to-edit code that lacks higher-level semantic structure. Latent variable models [Carlier et al., 2020, Cao et al., 2023, Wang and Lian, 2021, Ma et al., 2022, Li et al., 2020] focus on learning deep representations of vector graphics using techniques like VAEs [Kingma and Welling, 2013], diffusion models [Ho et al., 2020], and autoregressive decoders [Cao et al., 2023]. These models, when compared to their predecessors, offer better control and allow tasks like interpolation or style transfer but often work with simplified subsets of the SVG syntax, generating less compact or interpretable code.

More recent approaches treat SVG generation as a code generation task using LLMs. By tokenizing SVG code and generating it directly, LLM-based models bypass intermediate representations and learn the full SVG syntax. StarVector [Rodriguez et al., 2025b] was one of the first to explore in this direction, combining a code-centric LLM (StarCoder [Li et al., 2023b]) with a CLIP-based [Radford et al., 2021] image encoder, achieving impressive results on large-scale image-to-SVG benchmarks. Subsequent works, such as Beyond Pixels [Zhang et al., 2023], and follow-up studies [Cai et al., 2023, Nishina and Matsui, 2024, 2025], explored the potential for editing, reasoning, and structured generation with SVGs. OmniSVG [Yang et al., 2025] further advanced this field by building on the Qwen2.5-VL [Bai et al., 2025] foundation. However, these models face a significant limitation: they do not observe or evaluate the visual output of the SVG code they generate, often producing SVGs that are syntactically correct but inefficient and visually inaccurate.

## D.2 Vision-Language Models (VLMs)

Early VLMs such as Flamingo [Alayrac et al., 2022], BLIP-2 [Li et al., 2023a], LLaVA [Liu et al., 2023], and GPT-4V [OpenAI, 2023] introduced a powerful paradigm by adapting frozen vision encoders and connecting them to pretrained unimodal LLMs. This is typically done through learned projection layers that map visual features into token-like embeddings. These models follow an autoregressive generation strategy [Vaswani et al., 2017], enabling them to process both images and text within a unified token stream and perform instruction-following and multi-turn conversations grounded in visual inputs.

In parallel, the use of LLMs for code generation has grown rapidly recently. Code can be seen as a separate modality, with structure and syntax that differ significantly from natural language. Progress in this area has been driven by the availability of large scale code corpora [Kocetkov et al., 2022, Penedo et al., 2024] and the development of specialized coding models such as Codex [Chen et al., 2021], CodeGen [Nijkamp et al., 2022], and StarCoder [Li et al., 2023b].

Recent frontier models like GPT-4o [Hurst et al., 2024], Gemini [Georgiev et al., 2024], Claude [Anthropic, 2024], and Qwen2.5-VL [Bai et al., 2025] have expanded these capabilities further by incorporating larger and more diverse code datasets during pretraining, substantially improving performance on structured code tasks.

This convergence of multimodal architectures and code generation enables a new class of problems: inverse rendering code generation. In these tasks, the model generates code that compiles or renders into visual content. Examples include SVG, TikZ, and CAD, which are used for generating graphics, diagrams, and illustrations. Prior efforts in this area include diagram and layout generation [Rodriguez et al., 2023b,a, Belouadi et al., 2023, 2024, Rodriguez et al., 2025a, Belouadi et al., 2025] and text-to-CAD synthesis [Wang et al., 2025].

Although current models can learn valid code distributions and produce visually plausible outputs, they often suffer from hallucinations, limited long-range coherence, and poor generalization. A key limitation is the lack of feedback from the rendered environment. These models are never shown how their outputs actually look.

## D.3 Reinforcement Learning Post-Training

Reinforcement Learning (RL) has become essential for post-training fine-tuning of large language models (LLMs). Reinforcement Learning from Human Feedback (RLHF), using Proximal Policy Optimization (PPO) [Schulman et al., 2017], has become the standard for aligning LLM outputs with human preferences, improving tasks like summarization and instruction-following [Ziegler

et al., 2019, Stiennon et al., 2020, Ouyang et al., 2022]. An alternative approach, Group Relative Policy Optimization (GRPO) [Shao et al., 2024, Guo et al., 2025], stabilizes training by normalizing rewards across batches, removing the need for a separate value network and reducing variance. In code generation, frameworks like CodeRL employ an actor-critic approach where the critic predicts functional correctness to guide the actor [Le et al., 2022]. PPOCoder integrates PPO with execution feedback, using compiler results as rewards to fine-tune code generation models [Shojaee et al., 2023]. StepCoder introduces a curriculum of code completion subtasks, optimizing exploration by masking unexecuted code segments [Dou et al., 2024]. Additionally, Reinforcement Learning from Execution Feedback (RLEF) enables models to refine code iteratively based on execution outcomes, enhancing performance on complex tasks [Gehring et al., 2024]. In multimodal domains, approaches like ViCT use a visual critic to align generated UI code with input screenshots, improving fidelity in UI-to-code generation [Soselia et al., 2023]. RL has also been applied in architectural design, where agents learn to generate space layouts by optimizing spatial and functional constraints [Kakooee and Dillenburger, 2024]. Together, these methods demonstrate the power of RL in enhancing alignment, correctness, and efficiency across generative models.