Render-in-the-Loop: Vector Graphics Generation via Visual Self-Feedback

Guotao Liang[1], Zhangcheng Wang[3], Juncheng Hu[1], Haitao Zhou[1], Ziteng Xue[1], Jing Zhang[1], Dong Xu[2], and Qian Yu[1⋆]

[1] School of Software, Beihang University
{liangguotao, hujuncheng, zhouhaitao, zt_xue, zhang_jing, qianyu}@buaa.edu.cn
[2] Department of Computer Science, The University of Hong Kong
dongxu@cs.hku.hk
[3] 4Paradigm
wzc1@mail.ustc.edu.cn

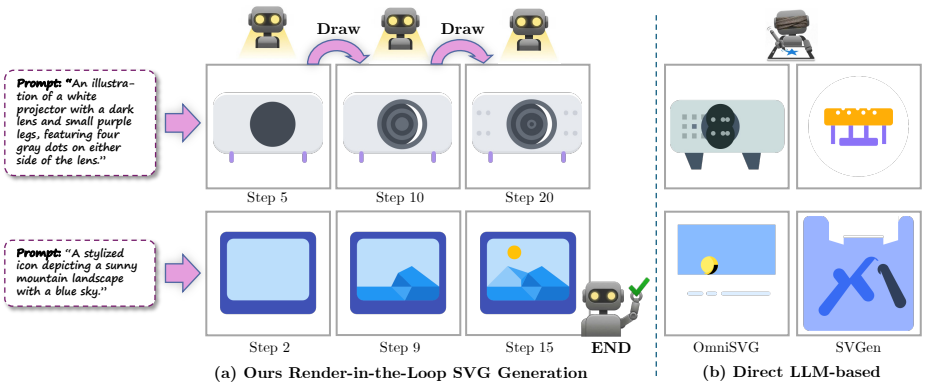


Fig. 1: Render-in-the-Loop Generation. (a) Our **Visual Self-Feedback** explicitly renders intermediate code into a canvas, feeding it back to provide continuous visual guidance. (b) Traditional open-loop approaches draw "blindly" using only textual history, often struggling with geometric accuracy and visual quality. By closing the loop, our method ensures structurally coherent and high-quality SVG synthesis.

Abstract. Multimodal Large Language Models (MLLMs) have shown promising capabilities in generating Scalable Vector Graphics (SVG) via direct code synthesis. However, existing paradigms typically adopt an open-loop "blind drawing" approach, where models generate symbolic code sequences without perceiving intermediate visual outcomes. This methodology severely underutilizes the powerful visual priors embedded in MLLMs' vision encoders, treating SVG generation as a disjointed textual sequence modeling task rather than an integrated visuo-spatial one. Consequently, models struggle to reason about partial canvas states and implicit occlusion relationships, which are visually explicit but textually ambiguous. To bridge this gap, we propose **Render-in-the-Loop**, a novel generation paradigm that reformulates SVG synthesis as a step-wise, visual-context-aware process. By rendering intermediate code states

⋆ Corresponding author.

into a cumulative canvas, the model explicitly observes the evolving visual context at each step, leveraging on-the-fly feedback to guide subsequent generation. However, we demonstrate that applying this visual loop naively to off-the-shelf models is suboptimal due to their inability to leverage incremental visual-code mappings. To address this, we first utilize fine-grained path decomposition to construct dense multi-step visual trajectories, and then introduce a Visual Self-Feedback (VSF) training strategy to condition the next primitive generation on intermediate visual states. Furthermore, a Render-and-Verify (RaV) inference mechanism is proposed to effectively filter degenerate and redundant primitives. Our framework, instantiated on a multimodal foundation model, outperforms strong open-weight baselines on the standard MMSVGBench. This result highlights the remarkable data efficiency and generalization capability of our Render-in-the-Loop paradigm for both Text-to-SVG and Image-to-SVG tasks.

Keywords: Scalable Vector Graphics · Multimodal Large Language Models · Visual Self-Feedback

1 Introduction

Scalable Vector Graphics (SVG) [6], characterized by resolution independence, high editability, and compact storage, have become an indispensable graphic format in modern UI/UX design, industrial typography, and front-end development [10, 18, 41, 55]. With the rapid progress of Multimodal Large Language Models (MLLMs) in code generation and visual understanding [1, 2], directly leveraging foundation models for *Text-to-SVG* and *Image-to-SVG* generation has recently emerged as a promising research direction [29, 46, 56].

Pioneering works such as StarVector [29] and OmniSVG [56] have demonstrated that MLLMs are capable of directly producing SVG XML code. However, existing generation paradigms typically adopt an open-loop "blind drawing" approach [5, 29, 46, 48, 56], where the model generates symbolic code sequences without perceiving the intermediate visual outcomes. This methodology severely underutilizes the powerful visual priors embedded in MLLMs' vision encoders, treating SVG generation as a disjointed textual sequence modeling task rather than an integrated visuo-spatial one [39, 44, 48]. As noted in recent studies [30], this disconnect between abstract code and its physical rendering often leads to overfitting on specific coordinate distributions, causing models to struggle with generalizing to diverse or out-of-distribution inputs. Without observing the canvas, models may hallucinate geometries that are syntactically valid but visually incoherent, and struggle to reason about partial canvas states and implicit occlusion relationships, which are visually explicit but textually ambiguous.

To mitigate instability in generation, recent efforts (*e.g.*, SVGen [39], Reason-SVG [51], and RLRF [30]) introduce Reinforcement Learning with Verifiable Rewards (RLVR), such as GRPO [11], to optimize large models. By utilizing renderability or visual alignment scores as feedback, these methods guide the

model toward better results. However, we argue that scalar rewards essentially *compress* rich visual information into a single number [57], thereby underutilizing the inherent capabilities of MLLMs. Modern MLLMs are equipped with powerful vision encoders designed to perceive dense visual details [1,2]. Instead of reducing visual feedback to a sparse reward signal, a more direct and information-rich approach is to leverage the vision encoder to explicitly *see* the evolving canvas as *Visual Context*.

Based on this insight, we propose **Render-in-the-Loop**, a novel generation paradigm (Fig. 1). The core idea is intuitive: decomposing the generation process into steps, rendering the partial SVG, and feeding it back to the MLLMs as a visual prompt for the next step. This paradigm is intrinsically aligned with the nature of vector graphics, where visual complexity arises from the sequential layering of primitives. By synchronizing the model's "drawing hand" with its "seeing eye", this loop allows the model to handle complex occlusion relationships and layer dependencies that are textually implicit but visually obvious.

However, **simply providing visual context to an off-the-shelf model is insufficient.** Our empirical observations reveal that naively applying this loop to pre-trained MLLMs (*e.g.*, Qwen3-VL [2], GPT-5 [4]) yields negligible improvement or even degradation across all evaluation metrics (see Tab. 2). This is because pre-trained models struggle to condition the generation of the next geometric primitive on intermediate visual states.

To bridge this gap, we develop a comprehensive framework encompassing **data construction, training, and inference verification**. Specifically, we utilize fine-grained path decomposition to construct dense multi-step visual trajectories for the *Visual Self-Feedback (VSF) training*. This explicitly aligns the model's visual perception with the incremental code generation process, enabling the vision encoder to effectively guide precise coordinate prediction based on the current canvas. During inference, leveraging the stepwise structure, we introduce a *Render-and-Verify (RaV)* strategy. Serving as a gatekeeper, RaV detects and filters out redundant or degenerate generation steps that result in negligible visual changes, effectively mitigating degenerate repetitions in generation.

To the best of our knowledge, the proposed Render-in-the-Loop mechanism represents a novel paradigm for LLM-based SVG generation. Importantly, it is highly orthogonal to existing code-domain improvements (*e.g.*, specialized semantic tokens) [40, 48, 56], curriculum learning strategies [39, 40], and optimization frameworks (*e.g.*, Reinforcement Learning) [30, 39, 51]. Therefore, Render-in-the-Loop can function both as a standalone enhancement and as a general underlying paradigm that integrates seamlessly with complementary optimization techniques, potentially unlocking stronger capabilities for vector generation.

Our main contributions are summarized as follows:

- **Paradigm Innovation.** We identify the limitations of "blind" autoregression and scalar-reward RL in SVG generation. We propose **Render-in-the-Loop**, which unlocks the visual potential of MLLMs by establishing a direct feedback loop between the rendered canvas and the generation process.

- **Synergistic Training and Inference.** We demonstrate that naive visual feedback is ineffective for foundational models. To address this, we introduce the *Visual Self-Feedback (VSF) training* to condition the next primitive generation on intermediate visual states, complemented by an inference-time Render-and-Verify (RaV) mechanism to suppress generation redundancy.
- **Strong Performance and Exceptional Data Efficiency.** Built upon the Qwen architectures [2], our approach achieves highly competitive performance on the standard MMSVGBench [56]. Notably, by learning from merely a 0.85M subset of the 2M open-source corpus provided by OmniSVG [56], our model surpasses the performance of both the OmniSVG baseline itself and the recent InternSVG [40] trained on 16M samples. This empirical evidence (see Tab. 1) demonstrates that explicit visual feedback is a more critical factor than raw data scaling for synthesizing high-quality vector graphics.

2 Related Works

Optimization and Autoregressive SVG Generation. Traditional vector graphics generation [8, 9, 14, 17, 18, 22, 23, 28, 33, 34, 37, 41, 42, 45, 49, 59, 61–63] is often formulated as an optimization problem. Enabled by differentiable rasterizers [20], early approaches [24, 35, 36] focus on image-to-SVG reconstruction. Subsequent works combine text-to-image diffusion models [15, 31] with Score Distillation Sampling (SDS) [26] to achieve text-to-SVG generation [19, 50, 53]. While visually appealing, these optimization-based methods [16, 19, 21, 38, 52] suffer from excessive inference latency and often produce chaotic paths that sacrifice topological editability. To bypass optimization bottlenecks, recent works treat SVG generation as a direct sequence modeling task. Leveraging large language models (LLMs) [1, 2], these methods synthesize SVGs by autoregressively predicting XML tags and path coordinates [3, 12, 29, 40, 46, 48, 56, 58]. However, these autoregressive models typically operate in an open-loop manner, blindly decoding abstract coordinates without observing the intermediate rendering results, which frequently leads to geometric hallucinations and structural inaccuracies.

Feedback-Driven SVG Generation. To mitigate the limitations of open-loop autoregressive generation, feedback mechanisms have been introduced. In broader text and code generation tasks, reinforcement learning (RL) techniques [11, 32] leverage outcome-based reward models [54] to align outputs with human preferences or logical constraints. In the specific context of vector graphics, recent methods such as SVGen [39], Reason-SVG [51], and rendering-aware RL approaches [30] explore the use of verifiable reinforcement learning. These methods typically employ rendering validity and visual-semantic alignment (*e.g.*, CLIP [27] scores) as scalar rewards to optimize the generated SVG codes. While RL introduces delayed guidance, mapping complex visual structures into unidimensional scalar rewards inherently compresses away critical spatial information. In this paper, we explore a more direct dense feedback mechanism. Instead of relying on abstract scalar rewards, our proposed Render-in-the-Loop paradigm

relies on constructing Visual Self-Feedback (VSF) training sequences. By continuously rendering and injecting intermediate canvas states back into the vision pathway of Multimodal Large Language Models (MLLMs), we provide precise step-by-step visual guidance, achieving highly competitive vector synthesis while avoiding the spatial information loss typical of reward-based scalar feedback.

3 Methodology

In this section, we present the proposed Render-in-the-Loop paradigm for vector graphic generation. We first introduce the Visual Self-Feedback (VSF) training framework in Sec. 3.1. We then describe the data preprocessing strategy tailored for constructing the training sequences in Sec. 3.2. Finally, we introduce a Render-and-Verify (RaV) decoding strategy for real-time error mitigation at inference time in Sec. 3.3.

3.1 Visual Self-Feedback Training Framework

We reformulate SVG generation not as a one-shot translation task (as seen in prior works [29]), but as a **step-wise drawing process** grounded in visual feedback. In this framework, the model functions as both the "hand" (generating XML code) and the "eye" (perceiving the canvas), learning to paint incrementally based on the evolving visual state. This sequential modeling aligns seamlessly with the intrinsic nature of vector rendering, where correct visual composition relies on the **precise ordering of layers**. By observing the intermediate canvas, the model effectively learns to resolve **complex occlusion relationships**.

Step-wise Drawing Formulation Formally, a complete SVG drawing session is modeled as a unified multimodal sequence containing interleaved user conditions, geometric code segments, and rasterized canvas states. Let P denote the user's prompt, which can be either a textual description (for text-to-SVG) or a reference image (for image-to-SVG). The generation process is decomposed into N steps, where each step t produces a code fragment C_t (*e.g.*, a path). The sequence is structured as:

$$\mathcal{S} = [P, C_1, I_1, C_2, I_2, \ldots, C_N, I_N, \langle \text{END} \rangle], \tag{1}$$

where **(1)** C_t is the SVG code segment generated at step t; **(2)** $I_t = R(C_1 \oplus \cdots \oplus C_t)$ is the rasterized image of the *cumulative* canvas after applying all code segments up to step t. This efficiently captures the evolving occlusion and spatial relationships. The rasterization function R is realized via standard vector graphics rendering pipelines; and **(3)** $\langle \text{END} \rangle$ is a special token indicating drawing completion. By comparing the intermediate canvas I_t against the target prompt P, the model explicitly learns *when to stop* to prevent redundant over-drawing.

Unlike standard pure code generation scenarios [39,56], our model conditions its prediction on the *full drawing history*. At any step t, to generate the next

code segment C_t, the model attends to the entire preceding context:

$$X_t = [P, C_1, I_1, \ldots, C_{t-1}, I_{t-1}]. \tag{2}$$

Input images I_k are processed by the vision encoder of Multimodal LLMs [2] and injected as visual tokens into the sequence. This ensures that the model explicitly perceives the valid visual outcome of its previous generated code before drawing the next stroke.

Training Objective We employ standard **Multiturn Visual Instruction Tuning** to train the model. The training data consists of constructed step-wise drawing sequences (details in Sec. 3.2). We optimize the model using the standard autoregressive language modeling loss over the entire sequence.

Let $\mathbf{y}$ represent the flattened sequence of tokens for the entire session $\mathcal{S}$. We define a binary mask M where $M_i = 1$ if the token y_i belongs to the model's output (*i.e.*, tokens within any C_t or the $\langle\text{END}\rangle$ token), and $M_i = 0$ otherwise (*i.e.*, for P and all inserted image tokens I_t). The VSF loss objective is:

$$\mathcal{L}_{\text{VSF}} = -\frac{1}{\sum_{j=1}^{|\mathbf{y}|} M_j} \sum_{i=1}^{|\mathbf{y}|} M_i \cdot \log P_\theta(y_i \mid y_{<i}), \tag{3}$$

where $y_{<i}$ denotes all preceding tokens in the sequence. Key characteristics of this objective include: **(1) Learning "How to Draw":** By observing the visual difference between the current canvas I_{t-1} and the goal P, the model learns the physical mapping to deduce the correct next stroke C_t. **(2) Learning "When to Stop":** The loss penalizes missing or premature $\langle\text{END}\rangle$ tokens, ensuring the model halts exactly when the intermediate canvas aligns with the target. **(3) Retaining Historical Context:** Rather than training on isolated state-to-action pairs, jointly optimizing the entire sequence allows the model to remember its past drawings. This ensures the model not only reacts to the current canvas but also follows a coherent, continuous drawing logic to complete the graphic.

3.2 Data Preprocessing for Step-wise Drawing

To enable the MLLM to *observe* intermediate rendering results during generation, we transform conventional static *Text/Image-to-Full-Code* datasets into interleaved multi-step image-text trajectories.

We adopt the open-source dataset subset from OmniSVG [56] as our primary data source. While the original work reports training on 2 million samples, only a portion was released (0.9M icons and 0.25M illustrations). To ensure data quality, we perform strict deduplication by filtering out samples with identical code strings, resulting in a refined set of 0.65M icons and 0.2M illustrations. Despite using this smaller, cleaner subset, our method demonstrates superior performance (see Sec. 4), highlighting the effectiveness of our visual feedback mechanism over raw data scale.

Fig. 2: Illustration of Fine-grained Path Decomposition. We compare the intermediate visual states without (a) and with (b) our path decomposition. Without splitting, a complex original path containing multiple visually disjoint elements (*e.g.*, projector body, lens, beams) is rendered abruptly in a single step, yielding sparse visual states. In contrast, our pipeline geometrically decomposes it into atomic subpaths, ensuring each distinct element is drawn sequentially. This significantly increases semantic feedback density, allowing the model to progressively learn spatial details.

The original SVG files are often optimized for storage, containing an average of only about 4 `<path>` elements per file. Simply splitting generation by these original elements would yield sparse visual states. Furthermore, generating a single complex path with numerous coordinates in one go challenges autoregressive models. We therefore perform fine-grained decomposition of lengthy paths to increase the density of visual feedback while maintaining rendering fidelity.

Fine-grained Path Decomposition Naively truncating the `d` attribute of a long `<path>` into multiple shorter paths often breaks SVG rendering semantics (*e.g.*, fill rules, subpath closure). To address this, we design a geometric decomposition pipeline. Fig. 2 illustrates an example of the decomposed results. The pipeline consists of three steps: **(1) Subpath Extraction:** We parse the drawing commands in the `d` attribute and decompose segments starting from non-contiguous commands (*e.g.*, `M`/`MoveTo`) into independent subpaths. **(2) Topology Analysis:** We treat each subpath as a 2D polygon and construct a dependency graph. An edge is added if subpaths have a containment relationship (often defining graphical holes via fill rules) or if they intersect spatially with non-unity opacity (to avoid color blending artifacts upon overlapping). **(3) Connected Component Merging:** We merge subpaths within the same connected component of the dependency graph back into a single `<path>` element. This ensures visually coupled elements remain atomic while independent elements are separated to provide more frequent visual feedback steps.

Curriculum Construction After fine-grained decomposition, the average number of `<path>` elements per SVG increases from 4 to 6. This preprocessing significantly improves the *density of visual intermediate states*. It also introduces an implicit curriculum: instead of drawing complex composite shapes in a single attempt, the model learns to sequentially compose simpler geometric primitives.

Finally, we convert each decomposed SVG into the standard interleaved format described in Sec. 3.1. The sequence $\mathcal{S}$ is constructed by rendering the cumulative canvas after each step t to obtain I_t, which is then inserted as the visual context for the subsequent step C_{t+1}.

3.3 Render-and-Verify

Standard autoregressive models often lack explicit mechanisms to correct errors during generation, leading to issues such as *repetitive loops*, *off-canvas drawing*, and *redundant over-drawing*. In our framework, since the model is conditioned on the rendered history, we introduce a lightweight **Render-and-Verify (RaV)** strategy at inference time to detect and mitigate these degenerate behaviors without additional training. At inference step t, the model generates a candidate code segment $\hat{C}_t$. Before accepting it, we render a hypothetical future canvas state $\hat{I}_t = R(C_{1:t-1} \oplus \hat{C}_t)$ and apply two heuristic checks. **Visual Difference Check:** We compare $\hat{I}_t$ with the previous state I_{t-1}. If the pixel-wise difference is below a threshold ϵ (indicating no visual contribution), $\hat{C}_t$ is rejected. **Repetition Check:** If the string similarity between $\hat{C}_t$ and C_{t-1} exceeds a threshold τ_{sim}, $\hat{C}_t$ is also rejected to prevent repetitive loops.

If the candidate $\hat{C}_t$ fails to pass either of these two heuristic checks, it indicates that the proposed code segment is either visually stagnant or structurally redundant. In such cases, we discard $\hat{C}_t$ and trigger an adaptive resampling mechanism. Specifically, we adjust the generation hyperparameters (*e.g.*, slightly increasing the sampling temperature) to encourage the model to escape local repetitive distributions and explore alternative, valid drawing sequences. This resampling is continuously repeated until a verified step is found. However, if the model consistently fails to produce a valid segment after a predefined maximum number of retries ($K_{\max}$), we reasonably assume that the visual composition is inherently complete and no further additions are necessary. Consequently, we force the current step to output the ⟨END⟩ token to explicitly terminate the drawing session. Overall, this Verify-then-Accept strategy functions as a lightweight yet highly effective inference-time guardrail. It dynamically filters out invalid generation steps on the fly, striving to ensure that every executed drawing action provides a concrete and meaningful visual contribution to the final graphic.

4 Experiments

To comprehensively evaluate the proposed Render-in-the-Loop paradigm, our evaluation focuses on three key questions: **(Q1)** Does the explicit visual feedback mechanism enable our model to outperform existing strong open-weight baselines, including larger-scale models (*e.g.*, OmniSVG and InternSVG) and RL-enhanced methods (*e.g.*, SVGen)? **(Q2)** How much does the Visual Self-Feedback (VSF) training strategy contribute to the model's generation quality compared to standard Supervised Fine-Tuning (SFT)? **(Q3)** Can the inference-time Render-and-Verify (RaV) strategy effectively mitigate degenerate repetitions in the generation process?

4.1 Experimental Setup

Datasets and Benchmarks. Following the protocol of OmniSVG [56], we utilize the **MMSVG** dataset for training and evaluation. **Training Set:** While the

original OmniSVG model was trained on the full dataset comprising 2M samples, only a subset has been publicly released. We train our model strictly on this available subset, which contains approximately **0.65 million icons and 0.2 million illustrations**. Despite using less than 50% of the baseline's data, our method demonstrates superior data efficiency. We apply the **Fine-grained Path Decomposition** (Sec. 3.2) to split complex paths, which are then organized into in-the-loop visual trajectories. **Test Set:** We evaluate our method on the comprehensive MMSVGBench [56], which was officially open-sourced by the authors of OmniSVG to establish a standardized evaluation protocol.

Baselines. We compare our method against three categories of strong baselines: (1) **Optimization-based Methods:** DiffVG [20], LIVE [24], VectorFusion [19] and SVGDreamer [53]; (2) **LLM/VLM-based Methods:** StarVector [29] (8B), IconShop [46], GPT-5 [4], OmniSVG [56] (8B, NeurIPS 2025), and InternSVG [40] (8B, ICLR 2026). Note that InternSVG is trained on a massive dataset of 16M samples (SAgoge). We ensure that the primary open-source baselines are of comparable model size to ours (8B); (3) **RL-based Methods:** SVGen [39] (7B). As the only open-sourced method currently applying Reinforcement Learning (specifically GRPO) to SVG generation, we select it as the representative baseline for this category.

Evaluation Metrics. We employ a comprehensive set of metrics to assess both visual quality and semantic alignment. For **Text-to-SVG**, we report FID (Fréchet Inception Distance) [13] to measure distribution-level image quality and CLIP Score [27] to evaluate semantic consistency with the prompt. We also report Aesthetic Score and Human Preference Score [47]. For **Image-to-SVG**, we measure reconstruction fidelity using MSE, LPIPS [60], SSIM [43], and DINO Score [25].

Implementation Details. Our model is initialized from Qwen3-VL-8B-Instruct [2]. We fine-tune the model for 3 epochs on 8 NVIDIA H100 GPUs using the AdamW optimizer with a learning rate of $1e-5$ and a cosine decay schedule. For the VSF training, all intermediate SVGs are rendered to a resolution of 224×224. In the Qwen3-VL architecture, the rendered image is encoded by the Vision Transformer [7] into exactly 49 visual tokens. This highly compressed visual representation ensures that interleaving multiple intermediate canvases does not overwhelm the LLM's context window. We set the maximum sequence length to 10240 to easily accommodate these multi-step trajectories. During inference, we employ the RaV strategy with a pixel difference threshold $\epsilon = 0.001$ and similarity threshold $\tau_{\text{sim}} = 0.98$.

4.2 Comparison

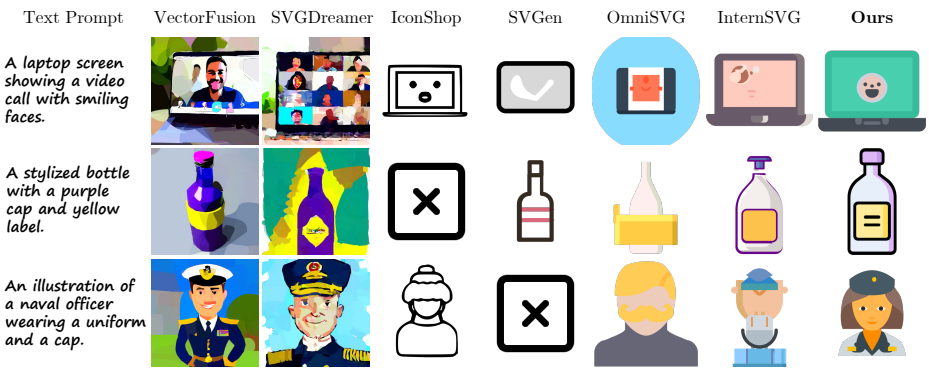
Quantitative Comparison We present the quantitative results on both Text-to-SVG and Image-to-SVG tasks in Tab. 1. Our Render-in-the-Loop model achieves highly competitive performance on both Icon and Illustration subsets.

Table 1: Quantitative results between our method and current strong text-to-SVG and image-to-SVG baselines on MMSVG benchmarks. Our model demonstrates superior SVG generation performance. Best results are in **bold**, and second best are underlined.

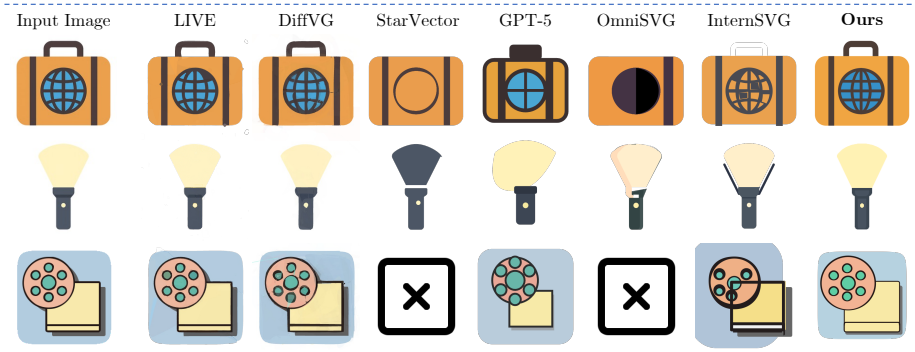
Dataset	Methods	Text-to-SVG				Image-to-SVG			
		FID↓	CLIP↑	Aes↑	HPS↑	DINO↑	SSIM↑	LPIPS↓	MSE↓
MMSVG-Icon	*Optimization-based Methods*								
	VectorFusion	**250.77**	**0.240**	**4.76**	**0.237**	-	-	-	-
	SVGDreamer	308.94	0.207	4.26	0.221	-	-	-	-
	LIVE	-	-	-	-	0.932	0.943	0.106	0.011
	DiffVG	-	-	-	-	**0.940**	**0.954**	**0.066**	**0.002**
	Autoregressive Methods								
	StarVector(8B)	-	-	-	-	0.895	0.881	0.231	0.059
	GPT-5	-	-	-	-	0.902	0.825	0.387	0.113
	IconShop	213.28	0.288	4.55	0.244	-	-	-	-
	SVGen(RL-based)	129.23	0.281	4.78	0.244	-	-	-	-
	OmniSVG(8B)	130.56	0.276	4.60	0.242	0.922	0.893	0.235	0.040
	InternSVG(8B)	128.80	0.291	4.75	0.246	0.926	0.901	0.182	0.034
	Qwen3-VL (SFT only)	148.72	0.262	4.41	0.232	0.889	0.842	0.312	0.094
	Ours (VSF)	127.98	**0.294**	4.80	0.249	0.928	0.908	0.188	0.031
	Ours (VSF+RaV)	**127.64**	0.293	**4.86**	**0.251**	**0.931**	**0.914**	**0.172**	**0.027**
MMSVG-Illustration	*Optimization-based Methods*								
	VectorFusion	**253.94**	0.185	**4.94**	**0.226**	-	-	-	-
	SVGDreamer	419.70	**0.201**	4.37	0.221	-	-	-	-
	LIVE	-	-	-	-	0.935	0.950	0.111	0.008
	DiffVG	-	-	-	-	**0.945**	**0.955**	**0.065**	**0.001**
	Autoregressive Methods								
	StarVector(8B)	-	-	-	-	0.877	0.900	0.238	0.046
	GPT-5	-	-	-	-	0.894	0.876	0.345	0.064
	IconShop	137.93	0.233	4.46	0.224	-	-	-	-
	SVGen(RL-based)	139.53	0.220	4.61	0.230	-	-	-	-
	OmniSVG(8B)	138.42	0.231	4.51	0.232	0.905	0.907	0.231	0.031
	InternSVG(8B)	138.10	0.229	4.58	0.234	0.912	0.915	0.205	0.026
	Qwen3-VL (SFT only)	155.84	0.221	4.33	0.221	0.882	0.861	0.318	0.082
	Ours (VSF)	137.86	0.235	4.69	0.241	0.913	0.921	0.193	0.024
	Ours (VSF+RaV)	**137.79**	**0.237**	**4.73**	**0.242**	**0.918**	**0.928**	**0.178**	**0.021**

Text-to-SVG Analysis. As shown in Tab. 1 (Left), our method significantly outperforms the optimization-based baselines (VectorFusion, SVGDreamer) in terms of editability and generation quality. More importantly, compared to **OmniSVG** (trained on 2M data), our method (trained on 0.85M) achieves a lower FID and higher CLIP Score. Remarkably, our method even surpasses the recent open-weight leader **InternSVG** (trained on a massive **16M** dataset (approx. 20× larger than ours)). For instance, on the Icon subset, we achieve a lower FID (127.64 vs. 128.80) and higher CLIP Score (0.293 vs. 0.291). This verifies our core hypothesis: *seeing* the clear rendering process and explicit layer-wise dependencies is far more effective than *blindly* learning from massive text-code pairs. Furthermore, our method outperforms the RL-based **SVGen**, suggesting that dense visual feedback provides richness that scalar rewards cannot match.

Image-to-SVG Analysis. Tab. 1 (Right) summarizes the reconstruction results. Our method demonstrates superior reconstruction capability, particularly in the



(a) **Text-to-SVG**



(b) **Image-to-SVG**

Fig. 3: Qualitative comparison on MMSVG benchmarks. (a) In Text-to-SVG, our method generates clear semantic details (*e.g.*, facial expressions, uniform details) where baselines like OmniSVG often fail or produce noise. (b) In Image-to-SVG, our model effectively reconstructs internal structures (*e.g.*, flashlight lens, briefcase patterns), demonstrating superior fidelity to the input raster image.

complex Illustration subset. The high DINO score indicates that our model captures the semantic structure of the input image accurately. By conditioning on the intermediate rendering results, our model maintains better structural consistency with the target image throughout the generation process, resulting in significantly lower LPIPS scores compared to the open-loop OmniSVG.

Qualitative Comparison Fig. 3 presents a visual comparison between our Render-in-the-Loop and representative baselines, including optimization-based (*e.g.*, VectorFusion [19], DiffVG [20]) and autoregressive (*e.g.*, OmniSVG [56], InternSVG [40]) approaches.

In the Text-to-SVG task (Fig. 3a), optimization-based methods like VectorFusion and SVGDreamer suffer from excessively long inference times and tend to generate messy, overlapping paths. This results in poor editability and disorganized layering, as seen in the chaotic background of the "naval officer". In contrast, our method generates compact, topologically clean primitives, main-

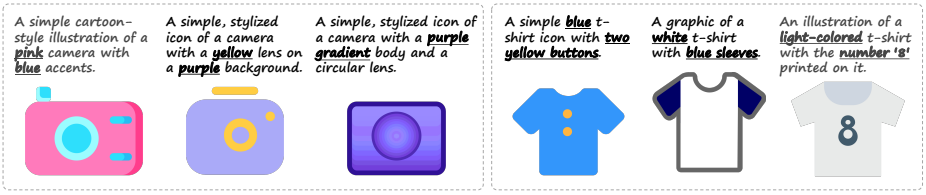


Fig. 4: Instruction-following examples. Our model accurately applies specific attributes (*e.g.*, colors and numbers) to their corresponding objects based on complex prompts.

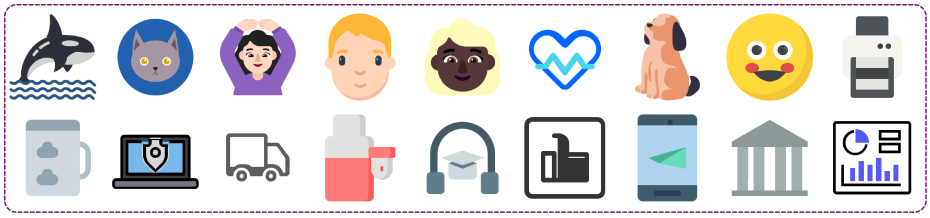


Fig. 5: A gallery of diverse vector graphics generated by our Render-in-the-Loop model. Our approach consistently produces aesthetically pleasing and semantically coherent SVGs across a wide variety of subjects, maintaining topologically clean path structures.

taining high visual quality and standard SVG structure. Compared to autoregressive baselines like OmniSVG and InternSVG, our model demonstrates superior semantic alignment. For example, in the "laptop screen", baselines often produce distorted faces or glitches, whereas our method generates clear details.

In the Image-to-SVG reconstruction (Fig. 3b), optimization methods like DiffVG and LIVE, while capturing pixel-level details, often resort to thousands of tiny paths to overfit the raster image, sacrificing the semantic editability of vector graphics. Blind autoregressive models (*e.g.*, OmniSVG) struggle to capture internal structures (*e.g.*, the lens details in the "flashlight" example). Our model strikes a balance, faithfully reconstructing both geometry and highlighting effects while maintaining a clean, editable rendering suitable for design workflows.

Fine-grained Instruction Following. Fig. 4 illustrates our model's fine-grained instruction-following ability. For prompts such as "a pink camera with blue accents" and "a blue t-shirt with two yellow buttons", the generated SVGs preserve the requested attribute–object bindings. In other words, the model places each attribute on the intended part and keeps this assignment stable throughout the drawing process. Additionally, Fig. 5 showcases a diverse gallery of SVG generations, demonstrating our model's versatility across a wide variety of subjects.

4.3 Ablation Studies

To investigate the effectiveness of each component in our framework, we conduct ablation studies by comparing our full model (VSF + RaV) with two variants: the base model fine-tuned using standard SFT (Qwen3-VL SFT only) and the model trained with VSF but without inference-time verification (Ours w/o RaV). Detailed results are presented in Tab. 1.

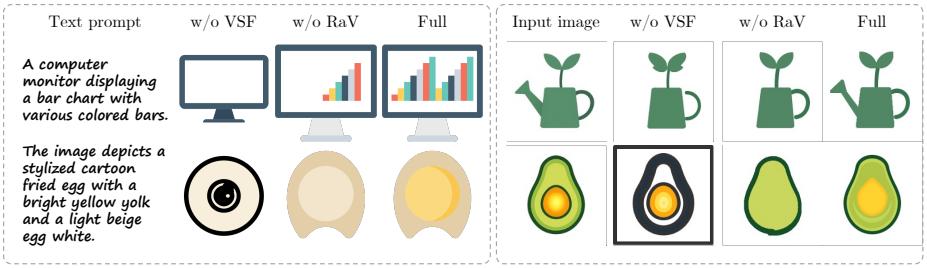


Fig. 6: Qualitative ablation study. **w/o VSF** fails to capture semantic details (*e.g.*, missing bar chart, hallucinating an eye for an egg). **w/o RaV** suffers from degenerate repetition (*e.g.*, re-drawing the same leaf), leading to incomplete shapes. Our full model generates correct and complete graphics.

Table 2: Degradation of "Naive VSF". Directly feeding intermediate visual canvases into standard models (without VSF training) uniformly degrades performance across both Text-to-SVG and Image-to-SVG tasks.

Dataset	Setting	Text-to-SVG				Image-to-SVG			
		CLIP↑	Aes↑	HPS↑	FID↓	DINO↑	SSIM↑	LPIPS↓	MSE↓
MMSVG-Icon	One-shot (Open-loop)	**0.311**	**5.026**	**0.219**	-	**0.902**	**0.825**	**0.387**	**0.113**
	Naive VSF (In-the-loop)	0.294	4.925	0.185	-	0.891	0.814	0.405	0.121
MMSVG-Illust.	One-shot (Open-loop)	**0.293**	**5.195**	**0.210**	-	**0.894**	**0.876**	**0.345**	**0.064**
	Naive VSF (In-the-loop)	0.281	4.807	0.167	-	0.883	0.852	0.388	0.075

Necessity of VSF Training. As argued in our introduction, simply providing visual context (an intermediate canvas) to an off-the-shelf model without specialized training is insufficient. To empirically validate this, we design a "Naive VSF" baseline: we prompt standard foundational models (Qwen3-VL for Text-to-SVG and GPT-5 for Image-to-SVG) to perform in-the-loop generation by observing intermediate rendering results, without any VSF training. As reported in Tab. 2, applying the Naive VSF (In-the-loop) formulation paradoxically leads to severe performance degradation across all metrics compared to standard open-loop generation. For instance, GPT-5's LPIPS notably increases from 0.345 to 0.388 on the Illustration subset. This confirms that pre-trained models intuitively struggle to align intermediate visual states with the incremental geometry code space. Therefore, our VSF data trajectory construction and training strategy are absolutely necessary to unlock the capabilities of the Render-in-the-Loop paradigm.

Effect of Visual Self-Feedback (VSF). As illustrated in Fig. 6 (Left), standard SFT models (w/o VSF) suffer from severe hallucinations when handling complex prompts. For the "monitor with bar chart" prompt, the blind model fails to draw the bars entirely, outputting a generic monitor shape. Similarly, for the "fried egg" prompt, it generates an eye-like structure, misinterpreting the geometric composition. By incorporating VSF, the model perceives the evolving canvas, enabling it to correct these semantic deviations and generate identifiable objects. Quantitatively, this corresponds to the significant FID reduction (155.84 $\rightarrow$ 137.86) observed in Tab. 1.

Effect of Render-and-Verify (RaV). Fig. 6 (Right) highlights the necessity of the RaV strategy. Without RaV, the model tends to fall into degenerate repetition loops, continuously generating redundant strokes that do not contribute to the visual progress (*e.g.*, re-drawing the same leaf contour in the "plant" example). This leads to incomplete SVGs. RaV acts as a gatekeeper: by rejecting paths that yield negligible visual differences (Pixel Difference $< \epsilon$), it forces the model to break out of these loops and proceed to draw the next meaningful component (*e.g.*, the watering can body). This ensures both the completeness and conciseness of the final output.

Data Efficiency: Doing More with Less. One of our most significant findings is the data efficiency of the Render-in-the-Loop paradigm. Despite being trained on only **0.85M** samples, our model matches or exceeds the performance of baselines trained on significantly larger datasets, including OmniSVG (2M) and InternSVG (16M). This suggests that the bottleneck in current SVG generation is not solely the scale of data, but also the lack of *effective usage* of visual priors. By incorporating visual self-feedback, we achieve highly competitive results with less than 6% of the training data used by the leading large-scale baseline.

Robustness to Occlusion. Vector graphics often involve complex layer stacking (*e.g.*, drawing a background first, then a foreground object). Open-loop models often fail to correctly order these layers, leading to "hidden" geometries that waste tokens without contributing to the image. Qualitative analysis shows that our model, by observing the canvas, naturally learns to draw from back to front, resolving occlusion ambiguities that text descriptions alone cannot capture.

5 Conclusion

In this paper, we propose *Render-in-the-Loop*, a novel generation paradigm that reformulates SVG generation from an open-loop textual process into an integrated, context-aware visual sequence. By explicitly conditioning the MLLM on its own incremental intermediate rendering results, our framework effectively unlocks the model's visual priors, better handling structural inaccuracies and complex occlusion relationships. Equipped with Visual Self-Feedback training and the Render-and-Verify inference mechanism, our model achieves highly competitive capabilities and exceptional data efficiency on standard benchmarks.

Limitations and Future Work. A current limitation of our approach is the inference overhead. Since our method requires repeatedly rendering intermediate SVG paths and interleaving the resulting images into the context window, the actual inference time is slightly increased compared to single-pass open-loop generation. Furthermore, the intermediate canvases are fixed at 224×224; the trade-off between perceiving finer details at higher resolutions and computational cost remains unexplored. For future work, we plan to further train the model to develop stronger capabilities of self-correction, reasoning, and planning

on top of the Render-in-the-Loop foundation, potentially incorporating dynamic multi-scale visual feedback. Beyond SVGs, we believe the core idea of closing the rendering feedback loop can seamlessly generalize to broader inverse rendering code generation tasks, including HTML/CSS for web development, LaTeX/TikZ for scientific visualization, 3D rendering programs, and complex CAD modeling systems. we truly hope this exciting line of work can serve as a highly versatile and general framework for advancing structured, code-driven visual synthesis.

Overview

In this supplementary material, we provide additional details related to our work on **Render-in-the-loop**. Specifically, this document covers the following aspects:

- **Implementation Details** (Sec. A): Provides extended details on our Render-and-Verify filtering strategy during inference, the system prompts for our models, and concrete examples of the visual self-feedback training format.
- **More Details of the Baselines** (Sec. B): Provides expanded overviews of the various baseline models included in our comparative study.
- **Comparison with LIVE and DiffVG** (Sec. C): Offers a more granular visual critique of the stroke-stacking problem prevalent in optimization-based methods compared to our structured generation.
- **MMSVGBench** (Sec. D): Details the structure and metrics of the multi-modal benchmark employed in our capability evaluations.

A Implementation Details

Render and Verify. The Render-and-Verify (RaV) strategy is implemented as a lightweight inference-time verification module that filters degenerate generation steps. While the main paper describes the conceptual framework, here we provide additional implementation details for reproducibility.

At inference step t, the model first produces a candidate SVG code fragment $\hat{C}_t$. Before accepting this fragment, we render a hypothetical future canvas state

$$\hat{I}_t = R(C_{1:t-1} \oplus \hat{C}_t),$$

where $R(\cdot)$ denotes the SVG rasterization function and $C_{1:t-1}$ represents all previously accepted primitives.

Two heuristic checks are then applied.

Visual Difference Check. We compute the pixel-wise difference between the rendered candidate canvas $\hat{I}_t$ and the previous canvas state I_{t-1}:

$$\Delta(I_{t-1}, \hat{I}_t) = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} |I_{t-1}(i,j) - \hat{I}_t(i,j)|.$$

If $\Delta(I_{t-1}, \hat{I}_t) < \epsilon$, the candidate fragment is considered to provide negligible visual contribution and is rejected. In practice, this mechanism effectively filters redundant operations such as repeatedly drawing the same primitive, generating elements that are fully occluded, or placing shapes outside the visible canvas.

Repetition Check. To prevent autoregressive repetition loops, we compute the string similarity between the candidate fragment $\hat{C}_t$ and the previously accepted fragment C_{t-1}. The similarity is measured using a normalized sequence matching score. If the similarity exceeds a threshold τ_{sim}, the candidate fragment is rejected to avoid structurally redundant SVG primitives.

Adaptive Resampling. If the candidate $\hat{C}_t$ fails either verification check, the fragment is discarded and the model resamples a new candidate. During resampling, the sampling hyperparameters (e.g., temperature and nucleus sampling range) are slightly increased to encourage the model to escape local repetitive distributions and explore alternative drawing actions. This process is repeated until a verified fragment is obtained.

Termination Criterion. If the model fails to produce a valid fragment after a predefined maximum number of retries ($K_{\max}$), the system assumes that the drawing is already complete and forces the generation of the ⟨END⟩ token to terminate the drawing process.

In our implementation, all intermediate canvases are rasterized at a resolution of 224×224. The SVG rasterization is implemented using the CairoSVG renderer, which faithfully follows the W3C SVG specification [6]. During inference, we set the visual difference threshold to $\epsilon = 0.001$ and the repetition similarity threshold to $\tau_{\text{sim}} = 0.98$.

System Prompt. We present the system prompts used for Text-to-SVG and Image-to-SVG tasks in Fig. S1 and Fig. S2, respectively. These prompts guide the MLLM to follow the iterative drawing protocol and leverage visual feedback.

Dataset Construction Examples. To illustrate the Visual Self-Feedback (VSF) training format, we provide a representative training sample in Fig. S3. The sequence demonstrates how the model learns to map instructions to incremental SVG paths and consistently conditions its next output on the rendered visual feedback.

B More Details of the Baselines

In this section, we provide additional details of the baseline methods used in our experiments. These methods span three categories: optimization-based approaches, autoregressive LLM/VLM-based models, and reinforcement learning based methods.

Optimization-based Methods. **VectorFusion** [19] generates SVG graphics from text prompts by distilling knowledge from pretrained diffusion models into vector representations. It optimizes vector parameters using score distillation sampling

Text-to-SVG System Prompt

You are an expert SVG artist and coder. Your task is to generate Scalable Vector Graphics (SVG) code incrementally based on a visual feedback loop.

Workflow Protocol
1. **Initialization**: The user will provide a text description.
2. **Incremental Drawing**: Do NOT output the entire SVG at once. Output small, logical fragments (e.g., one shape or path) in each turn wrapped in ‘‘‘`svg` code blocks.
3. **Visual Self-Correction**:
- After your first turn, the user will **STOP** providing text instructions.
- The user will **ONLY** input a rasterized `image` of the current canvas state.
- IMPORTANT: Each `image` is the PNG render of the SVG canvas produced by cumulatively applying all SVG fragments you have output so far. Treat it as the current accumulated drawing state.
- You must act as the “eye” and the “hand”: Look at the image, and output the *next* SVG code fragment to complete the drawing.
4. **Termination**:
- When you see the image and determine the drawing is fully complete and matches the goal, you must output a special termination signal inside an svg block:

‘‘‘`svg`
`<END>`
’’’

Technical Constraints
- Canvas Size: 224x224.
- Coordinate System: Maintain strict spatial awareness within the 0-224 range.
- Style: Concise, geometric, and aesthetically pleasing vector art.

Fig. S1: System Prompt for Text-to-SVG Task.

(SDS) together with a differentiable vector graphics renderer. While the method can produce diverse vector graphics without large SVG training datasets, the optimization process is computationally expensive and often requires long inference time.

SVGDreamer [53] proposes a semantic-driven image vectorization (SIVE) framework for text-to-SVG generation. It separates foreground objects from background regions and optimizes vector primitives using a vectorized particle-based score distillation strategy. Although this approach improves editability and semantic structure, the iterative optimization pipeline remains computationally intensive.

LIVE [24] (Layer-wise Image Vectorization) reconstructs SVG graphics by progressively adding and optimizing vector paths to match a raster target image. The method employs a differentiable renderer to optimize path parameters and introduces component-wise path initialization to improve topology extraction.

Image-to-SVG System Prompt

You are an expert SVG artist and visual reverse-engineering model. Your task is to reconstruct Scalable Vector Graphics (SVG) code incrementally based on a visual feedback loop.

Task Definition
You are given an image that represents the final target appearance of an SVG drawing. Your goal is to reproduce this image using SVG code, generated step by step.

Workflow Protocol
1. **Initialization**
- The user's first input will be an image.
- Observe the image and infer the intended visual result.
2. **Incremental Drawing**
- Do NOT output the entire SVG at once.
- Output small, logical SVG fragments wrapped in ```` ```svg ```` code blocks.
- All SVG fragments are cumulatively applied to the same canvas.
3. **Visual Feedback Loop**
- After your first SVG output, the user will stop providing instructions.
- The user will ONLY provide images as input.
- Each image represents the current canvas state rendered from all SVG fragments so far.
- Compare the image with your intended result and output the next SVG fragment needed.
4. **Termination**
- When the drawing is complete and matches the target image, output: `<END>`

Technical Constraints
- Canvas size: 224 × 224.
- Coordinate system: 0–224.
- Style: Concise, geometric, and visually faithful.

Fig. S2: System Prompt for Image-to-SVG Task.

DiffVG [20] is a differentiable vector graphics rasterization framework that enables gradient-based optimization over vector primitives. By computing gradients through the rasterization pipeline, DiffVG directly optimizes vector parameters to match raster images. Despite its effectiveness in image vectorization tasks, the forward–backward rasterization process introduces significant computational overhead.

Autoregressive LLM/VLM-based Methods. **StarVector** [29] generates SVG code directly from images using a multimodal transformer architecture. It integrates an image encoder with a language model to autoregressively predict SVG primitives. While the model demonstrates strong performance on icon-level vectoriza-

Visual Self-Feedback Training (Sample)

[User]: Two interlocking hearts, one with a single drop of water or liquid at the bottom.

[Assistant (Step 1)]: `<path d="M 173 153 C 175 143 176 133 182 124 C 197 100 220 77 211 46 C 206 27 190 18 175 18 ..."/>`
→ *Canvas rendered as* I_1

[Visual Self-Feedback]: `<image>` (Current canvas state I_1)

[Assistant (Step 2)]: `<path d="M 160 203 C 160 210 166 215 174 215 C 182 215 188 210 188 203 C 188 188 177 191 ..."/>`
→ *Canvas updated to* I_2

[Visual Self-Feedback]: `<image>` (Current canvas state I_2)

[Assistant (Step 3)]: `<path d="M 98 72 C 98 72 98 72 98 72 C 98 72 98 72 98 72"/>`
→ *Canvas updated to* I_3

[Visual Self-Feedback]: `<image>` (Canvas state I_3)

[Assistant (Step 4)]: `<END>`

Fig. S3: A sample from our fine-grained decomposition dataset. The Assistant generates SVG segments incrementally, interleaved with rendered visual context.

tion tasks, its limited context length restricts the generation of highly complex SVG structures.

IconShop [46] adopts a transformer-based autoregressive model to generate SVG path sequences. By tokenizing SVG commands and coordinates, the model learns to synthesize vector graphics from textual descriptions. IconShop achieves competitive performance for icon synthesis but is mainly designed for relatively simple and monochrome SVG structures.

GPT-5 [4] represents a large-scale multimodal language model capable of generating SVG code through instruction following. It demonstrates strong general reasoning and visual understanding capabilities but is not specifically optimized for structured vector graphics generation.

OmniSVG [56] is a unified multimodal SVG generation framework built upon a vision-language model backbone. By parameterizing SVG commands and coordinates into discrete tokens, OmniSVG enables autoregressive SVG generation conditioned on textual or visual inputs.

InternSVG [40] is a recent large-scale SVG generation model trained on the SAgoge dataset containing approximately 16M SVG samples. It leverages large multimodal transformers to generate complex vector graphics and demonstrates strong scalability with large training corpora.

Reinforcement Learning-based Methods. **SVGen** [39] introduces reinforcement learning for SVG generation by optimizing the generation policy using the Group Relative Policy Optimization (GRPO) [11] algorithm. The method aims to improve the alignment between generated SVG outputs and evaluation metrics through policy optimization. As one of the first RL-based approaches for SVG generation, it provides an important baseline for evaluating reinforcement learning strategies in vector graphics synthesis.

Unless otherwise specified, we use the official implementations of these baselines and follow the hyperparameters recommended in their original papers.

C Comparison with LIVE and DiffVG

While optimization-based vectorization methods such as LIVE [24] and DiffVG [20] can achieve high visual fidelity through gradient descent, their resulting SVG representations often lack human-interpretable structure and logical layering.

As demonstrated in Fig. S4, these methods typically generate a massive number of redundant paths and messy overlapping layers to merely fit the target image pixels. This "stroke-stacking" behavior leads to bloated SVG files that are difficult for human designers to edit, animate, or reuse. In contrast, our proposed **Render-in-the-Loop** framework leverages the semantic prior of Multimodal Large Language Models (MLLMs) to generate clean, geometrically logical, and highly editable vector graphics that align with human drawing habits.

D MMSVGBench

We evaluate our method on MMSVGBench [56], a benchmark proposed by OmniSVG [56] for evaluating multimodal SVG generation models. The benchmark contains 600 synthetic samples covering two tasks: text-to-SVG and image-to-SVG. Each task contains 300 evaluation samples, further divided into icon-level and illustration-level complexity.

To ensure fair evaluation and avoid training data leakage, all prompts and images in MMSVGBench are generated from scratch using GPT [1] models. Therefore, the benchmark samples are guaranteed to be unseen during model training.

Each benchmark sample contains the input modality (text or image), task type, semantic category (icon or illustration), and metadata such as the source URL. Following the protocol of OmniSVG, we report FID, CLIP score, Aesthetic score and HPS for text-to-SVG tasks, and DINO similarity, SSIM, LPIPS, and MSE for image-to-SVG tasks.

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)

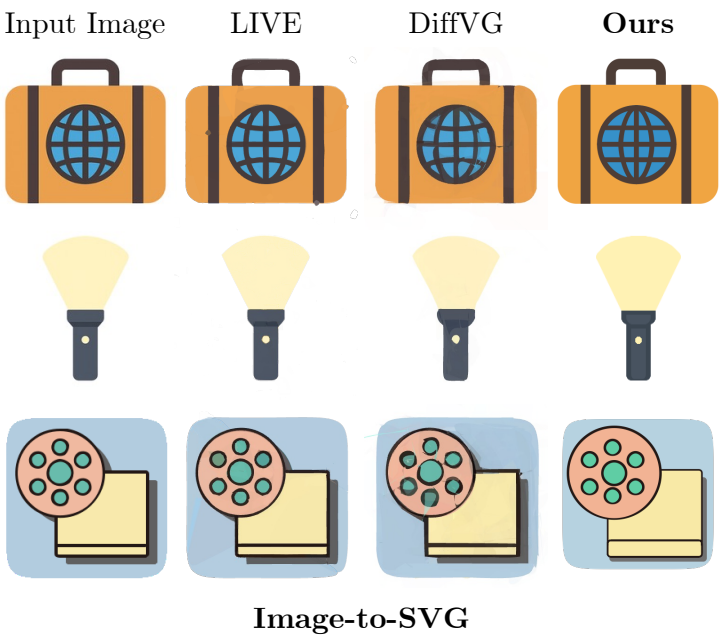


Fig. S4: A zoomed-in visualization of the SVG paths generated by optimization-based baselines (LIVE and DiffVG). To minimize the pixel-wise rendering loss, these methods tend to produce chaotic, densely overlapping Bézier curves and redundant shape layers. This results in an uneditable and semantically structureless vector representation, which starkly contrasts with the clean, logically constructed outputs produced by our approach.

2. Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., Ge, W., Guo, Z., Huang, Q., Huang, J., Huang, F., Hui, B., Jiang, S., Li, Z., Li, M., Li, M., Li, K., Lin, Z., Lin, J., Liu, X., Liu, J., Liu, C., Liu, Y., Liu, D., Liu, S., Lu, D., Luo, R., Lv, C., Men, R., Meng, L., Ren, X., Ren, X., Song, S., Sun, Y., Tang, J., Tu, J., Wan, J., Wang, P., Wang, P., Wang, Q., Wang, Y., Xie, T., Xu, Y., Xu, H., Xu, J., Yang, Z., Yang, M., Yang, J., Yang, A., Yu, B., Zhang, F., Zhang, H., Zhang, X., Zheng, B., Zhong, H., Zhou, J., Zhou, F., Zhou, J., Zhu, Y., Zhu, K.: Qwen3-vl technical report. arXiv preprint arXiv:2511.21631 (2025)
3. Carlier, A., Danelljan, M., Alahi, A., Timofte, R.: Deepsvg: A hierarchical generative network for vector graphics animation. Advances in Neural Information Processing Systems (NeurIPS) **33**, 16351–16361 (2020)
4. ChatGPT: Gpt-5 (2024), `https://chatgpt.com`
5. Chen, H., Zhao, Z., Chen, Y., Liang, Z., Ni, B.: Svgthinker: Instruction-aligned and reasoning-driven text-to-svg generation. In: Proceedings of the 33rd ACM International Conference on Multimedia. pp. 11004–11012 (2025)
6. Consortium, W.W.W.: Scalable vector graphics (svg) specification (1999), `https://www.w3.org/TR/1999/WD-SVG-19990211/`
7. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is

worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020)
8. Frans, K., Soros, L., Witkowski, O.: CLIPDraw: Exploring text-to-drawing synthesis through language-image encoders. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)
9. Gal, R., Vinker, Y., Alaluf, Y., Bermano, A., Cohen-Or, D., Shamir, A., Chechik, G.: Breathing life into sketches using text-to-video priors. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4325–4336 (2024)
10. Geroimenko, V.: Visualizing Information Using SVG and X3D: XML-based Technologies for the XML-based Web. Springer Science & Business Media (2005)
11. Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., et al.: Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948 (2025)
12. Ha, D., Eck, D.: A neural representation of sketch drawings. In: International Conference on Learning Representations (ICLR) (2018), https://openreview.net/forum?id=Hy6GHpkCW
13. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
14. Hirschorn, O., Jevnisek, A., Avidan, S.: Optimize & reduce: a top-down approach for image vectorization. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 38, pp. 2148–2156 (2024)
15. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. In: Advances in Neural Information Processing Systems (NeurIPS). vol. 33, pp. 6840–6851 (2020)
16. Hu, J., Xing, X., Zhang, J., Yu, Q.: Vectorpainter: Advanced stylized vector graphics synthesis using stroke-style priors. In: 2025 IEEE International Conference on Multimedia and Expo (ICME). pp. 1–6. IEEE (2025)
17. Hu, T., Yi, R., Qian, B., Zhang, J., Rosin, P.L., Lai, Y.K.: Supersvg: Superpixel-based scalable vector graphics synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 24892–24901 (2024)
18. Iluz, S., Vinker, Y., Hertz, A., Berio, D., Cohen-Or, D., Shamir, A.: Word-as-image for semantic typography. ACM Transactions on Graphics (TOG) **42**(4), 1–11 (2023)
19. Jain, A., Xie, A., Abbeel, P.: Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1911–1920 (2023)
20. Li, T.M., Lukáč, M., Gharbi, M., Ragan-Kelley, J.: Differentiable vector graphics rasterization for editing and learning. ACM Transactions on Graphics (TOG) **39**(6), 1–15 (2020)
21. Liang, G., Hu, J., Xing, X., Zhang, J., Yu, Q.: Multi-object sketch animation with grouping and motion trajectory priors. In: Proceedings of the 33rd ACM International Conference on Multimedia. pp. 9237–9246 (2025)
22. Liu, J., Xin, Z., Fu, Y., Zhao, R., Lan, B., Li, X.: Multi-object sketch animation by scene decomposition and motion planning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 11537–11546 (2025)
23. Lopes, R.G., Ha, D., Eck, D., Shlens, J.: A learned representation for scalable vector graphics. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7930–7939 (2019)

24. Ma, X., Zhou, Y., Xu, X., Sun, B., Filev, V., Orlov, N., Fu, Y., Shi, H.: Towards layer-wise image vectorization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 16314–16323 (2022)
25. Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al.: Dinov2: Learning robust visual features without supervision. arXiv preprint arXiv:2304.07193 (2023)
26. Poole, B., Jain, A., Barron, J.T., Mildenhall, B.: Dreamfusion: Text-to-3d using 2d diffusion. In: The Eleventh International Conference on Learning Representations (ICLR) (2023)
27. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International Conference on Machine Learning (ICML). pp. 8748–8763. PMLR (2021)
28. Reddy, P., Gharbi, M., Lukac, M., Mitra, N.J.: Im2vec: Synthesizing vector graphics without vector supervision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7342–7351 (2021)
29. Rodriguez, J.A., Puri, A., Agarwal, S., Laradji, I.H., Rodriguez, P., Rajeswar, S., Vazquez, D., Pal, C., Pedersoli, M.: Starvector: Generating scalable vector graphics code from images and text. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 16175–16186 (2025)
30. Rodriguez, J.A., Zhang, H., Puri, A., Feizi, A., Pramanik, R., Wichmann, P., Mondal, A., Samsami, M.R., Awal, R., Taslakian, P., et al.: Rendering-aware reinforcement learning for vector graphics generation. arXiv preprint arXiv:2505.20793 (2025)
31. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10684–10695 (2022)
32. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
33. Tang, Z., Wu, C., Zhang, Z., Ni, M., Yin, S., Liu, Y., Yang, Z., Wang, L., Liu, Z., Li, J., et al.: Strokenuwa: Tokenizing strokes for vector graphic synthesis. In: International Conference on Machine Learning. pp. 47830–47845. PMLR (2024)
34. Thamizharasan, V., Liu, D., Fisher, M., Zhao, N., Kalogerakis, E., Lukac, M.: Nivel: Neural implicit vector layers for text-to-vector generation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4589–4597 (2024)
35. Vinker, Y., Alaluf, Y., Cohen-Or, D., Shamir, A.: Clipascene: Scene sketching with different types and levels of abstraction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4146–4156 (2023)
36. Vinker, Y., Pajouheshgar, E., Bo, J.Y., Bachmann, R.C., Bermano, A.H., Cohen-Or, D., Zamir, A., Shamir, A.: Clipasso: Semantically-aware object sketching. ACM Transactions on Graphics (TOG) **41**(4), 1–11 (2022)
37. Vinker, Y., Shaham, T.R., Zheng, K., Zhao, A., E Fan, J., Torralba, A.: Sketchagent: Language-driven sequential sketch generation. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 23355–23368 (2025)
38. Wang, C., Zhou, H., Luo, L., Yu, Q.: Viewcraft3d: High-fidelity and view-consistent 3d vector graphics synthesis. arXiv preprint arXiv:2505.19492 (2025)
39. Wang, F., Zhao, Z., Liu, Y., Zhang, D., Gao, J., Sun, H., Li, X.: Svgen: Interpretable vector graphics generation with large language models. In: Proceedings of the 33rd ACM International Conference on Multimedia. pp. 9608–9617 (2025)

40. Wang, H., Yin, J., Wei, Q., Zeng, W., Gu, L., Ye, S., Gao, Z., Wang, Y., Zhang, Y., Li, Y., et al.: Internsvg: Towards unified svg tasks with multimodal large language models. arXiv preprint arXiv:2510.11341 (2025)
41. Wang, Y., Lian, Z.: Deepvecfont: Synthesizing high-quality vector fonts via dual-modality learning. ACM Transactions on Graphics (TOG) **40**(6) (2021)
42. Wang, Z., Huang, J., Sun, Z., Gong, Y., Cohen-Or, D., Lu, M.: Layered image vectorization via semantic simplification. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 7728–7738 (2025)
43. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4), 600–612 (2004)
44. Wu, R., Su, W., Liao, J.: Chat2svg: Vector graphics generation with large language models and image diffusion models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 23690–23700 (2025)
45. Wu, R., Su, W., Liao, J.: Layerpeeler: Autoregressive peeling for layer-wise image vectorization. In: Proceedings of the SIGGRAPH Asia 2025 Conference Papers (2025)
46. Wu, R., Su, W., Ma, K., Liao, J.: Iconshop: Text-guided vector icon synthesis with autoregressive transformers. ACM Transactions on Graphics (TOG) **42**(6), 1–14 (2023)
47. Wu, X., Sun, K., Zhu, F., Zhao, R., Li, H.: Human preference score: Better aligning text-to-image models with human preference. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 2096–2105 (2023)
48. Xing, X., Hu, J., Liang, G., Zhang, J., Xu, D., Yu, Q.: Empowering llms to understand and generate complex vector graphics. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 19487–19497 (2025)
49. Xing, X., Hu, J., Zhang, J., Xu, D., Yu, Q.: Svgfusion: Scalable text-to-svg generation via vector space diffusion. arXiv preprint arXiv:2412.10437 (2024)
50. Xing, X., Wang, C., Zhou, H., Zhang, J., Yu, Q., Xu, D.: Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. Advances in Neural Information Processing Systems **36**, 15869–15889 (2023)
51. Xing, X., Xue, Z., Guan, Y., Zhang, J., Xu, D., Yu, Q.: Reason-svg: Enhancing structured reasoning for vector graphics generation with reinforcement learning. arXiv preprint arXiv:2505.24499 (2026)
52. Xing, X., Yu, Q., Wang, C., Zhou, H., Zhang, J., Xu, D.: Svgdreamer++: Advancing editability and diversity in text-guided svg generation. IEEE Transactions on Pattern Analysis and Machine Intelligence (2025)
53. Xing, X., Zhou, H., Wang, C., Zhang, J., Xu, D., Yu, Q.: Svgdreamer: Text guided svg generation with diffusion model. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4546–4555 (2024)
54. Xu, J., Liu, X., Wu, Y., Tong, Y., Li, Q., Ding, M., Tang, J., Dong, Y.: Imagereward: Learning and evaluating human preferences for text-to-image generation. Advances in Neural Information Processing Systems **36**, 15903–15935 (2023)
55. Yan, W.: Integrating web 2d and 3d technologies for architectural visualization: applications of svg and x3d/vrml in environmental behavior simulation. In: Proceedings of the eleventh international conference on 3D web technology. pp. 37–45 (2006)
56. Yang, Y., Cheng, W., Chen, S., Zeng, X., Yin, F., Zhang, J., Wang, L., Yu, G., Ma, X., Jiang, Y.G.: Omnisvg: A unified scalable vector graphics generation model. In: The Thirty-ninth Annual Conference on Neural Information Processing Systems (2025)

57. Yue, Y., Chen, Z., Lu, R., Zhao, A., Wang, Z., Song, S., Huang, G.: Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? arXiv preprint arXiv:2504.13837 (2025)
58. Zhang, P., Zhao, N., Fisher, M., Xu, Y., Liao, J., Liu, D.: Duetsvg: Unified multimodal svg generation with internal visual guidance. arXiv preprint arXiv:2512.10894 (2025)
59. Zhang, P., Zhao, N., Liao, J.: Text-to-vector generation with neural path representation. ACM Transactions on Graphics (TOG) **43**(4), 1–13 (2024)
60. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 586–595 (2018)
61. Zhao, K., Bao, L., Li, Y., Su, X., Zhang, K., Qiao, X.: Less is more: Efficient image vectorization with adaptive parameterization. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 18166–18175 (2025)
62. Zhou, H., Zhang, H., Wang, B.: Segmentation-guided layer-wise image vectorization with gradient fills. In: European Conference on Computer Vision. pp. 165–180. Springer (2024)
63. Zhu, H., Chong, J.I., Hu, T., Yi, R., Lai, Y.K., Rosin, P.L.: Samvg: A multi-stage image vectorization model with the segment-anything model. In: ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 4350–4354. IEEE (2024)