PLANNING BEFORE DRAWING: DUAL-EXPERT ON-POLICY DISTILLATION FOR TEXT-TO-SVG

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-SVG generation aims to translate textual descriptions into executable SVG code that produces visually faithful and editable vector graphics. A key challenge is to preserve visual fidelity while organizing low-level drawing commands into meaningful semantic components that better reflect the compositional structure of the depicted content. Existing methods improve SVG generation through large scale data, SVG specific representations, and reinforcement learning, yet typically learn semantic organization and geometric realization within a single model. Since planning what to draw and realizing how to draw involve distinct objectives, joint optimization can cause competing supervision and limit both capabilities. To address these, we propose DEFT, a dual-expert framework that separates planning from drawing and then integrates their complementary knowledge. We first construct SVGVerse, a semantically structured SVG dataset, and perform two stage supervised fine tuning to establish correspondences among textual descriptions, semantic components, and drawing commands. We then optimize a Planning Expert for semantic organization and a Drawing Expert for geometric realization, followed by Command Routed On-Policy Distillation that selectively transfers their expertise according to the semantic roles of SVG commands. Extensive experiments demonstrate that our method achieves superior performance over traditional SVG generation methods, zero-shot models, and specialized SVG-LLMs.

1 INTRODUCTION

Scalable Vector Graphics (SVG) (Peng, 2000; Quint, 2003) represent visual content as structured Extensible Markup Language (XML) code composed of parameterized geometric primitives, providing resolution independent rendering, compact storage, and fine-grained editability. These properties have made SVG a fundamental format for web design, user interfaces, scientific visualization, and computer aided design (Li et al., 2020; Wang et al., 2026c). Text-to-SVG generation aims to translate a natural language description into executable SVG code, allowing users to create editable vector graphics from textual intent without manually specifying geometric primitives or manipulating professional design tools. With the rapid development of large language models (LLMs) and vision language models (VLMs), this task has received increasing attention as an intersection of visual content generation and structured code synthesis (Xing et al., 2025b; Rodriguez et al., 2025).

Existing Text-to-SVG methods mainly follow two technical paradigms: optimization based approaches and autoregressive sequence based approaches. Optimization based methods treat geometric primitives or path parameters as optimizable variables and iteratively update them according to feedback from differentiable rendering, vision language models, or diffusion models (Li et al., 2020; Jain et al., 2023; Xing et al., 2024). While these methods can produce visually appealing vector graphics, they require costly iterative optimization and often result in redundant or poorly organized SVG code, limiting their scalability and editability. In contrast, autoregressive approaches formulate Text-to-SVG as a sequence generation problem and directly synthesize SVG code using Transformers, LLMs, or VLMs (Wu et al., 2023; Xing et al., 2025b; Rodriguez et al., 2025; Yang et al., 2025; Wang et al., 2026c;b;a). This formulation supports end to end learning from large SVG corpora, avoids per instance optimization, and better exploits the textual and programmatic nature of SVG, making autoregressive generation a promising direction for scalable Text-to-SVG synthesis.

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

Despite this progress, current autoregressive methods still underexploit the semantic structure inherent in SVG code. Many SVGs encode visually coherent objects through low level drawing commands without organizing them according to their semantic roles, causing the code structure to poorly reflect the compositional structure of the rendered vector graphic. Recent studies attempt to expose intermediate structure through chain-of-thought reasoning (Wang et al., 2025; Chen et al., 2025a; Xing et al., 2025a; Wang et al., 2026b) or explicit plan then generate pipelines (Li et al., 2026). These methods make the generation process more interpretable, and some further align intermediate planning steps with grouped SVG code. However, they do not treat semantic component organization and geometric realization as two specialized capabilities that require separate optimization and subsequent integration. This distinction is important because the two capabilities are governed by different objectives: planning determines which semantic components should appear and how they should be organized, whereas drawing determines how these components are realized through geometric primitives and attributes. We empirically find that jointly optimizing planning and drawing rewards leads to a compromise policy. Although it improves both objectives over the shared supervised initialization, it fails to retain the respective strengths of the specialized policies.

To address these challenges, we propose **DEFT** (**D**ual-**E**xpert **F**ine-grained Dis**T**illation), a Text-to-SVG generation framework that separates the acquisition of planning and drawing capabilities and then integrates their complementary knowledge according to the semantic roles of SVG commands. We first construct **SVGVerse**, a semantically structured SVG dataset in which visually coherent elements are organized into explicit semantic groups while preserving the rendered content. Based on this data, we perform two stage supervised fine tuning to establish correspondences among textual descriptions, semantic components, and drawing commands. Starting from the same supervised initialization, we then optimize a Planning Expert for semantic component selection and organization and a Drawing Expert for geometric realization and visual quality. Rather than combining their heterogeneous reward signals within a single policy, this design allows each expert to specialize in a distinct aspect of SVG generation. Finally, we introduce Command Routed On-Policy Distillation to integrate the two experts into a single model. Distillation supervision is selectively applied to SVG commands associated with each expert, and different supervision strategies are adopted according to the distinct characteristics of semantic planning and geometric realization.

Extensive experiments demonstrate the effectiveness of the proposed approach. Our model achieves the strongest overall performance on the SVGVerse test set and exhibits strong generalization on MMSVG-Bench (Yang et al., 2025) and SArena (Wang et al., 2026c) when compared with general purpose language models, proprietary models, and specialized SVG generation models. Further analyses show the complementary specialization of the Planning and Drawing experts, while ablation studies validate the effects of command routed distillation, the planning to drawing distillation order, and the asymmetric KL objectives. These results demonstrate that explicitly separating semantic planning from geometric realization and integrating them through structured expert distillation provides an effective approach to Text-to-SVG generation.

2 PRELIMINARIES

Group Relative Policy Optimization. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) optimizes a policy π_θ using relative advantages estimated from a group of sampled outputs. Given a textual description x, GRPO samples K SVG code sequences $\{c_i\}_{i=1}^K$ and evaluates them with a reward function R, yielding $R_i = R(c_i, x)$. The normalized group advantage is

$$A_i = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^K)}{\text{std}(\{R_j\}_{j=1}^K)}. \tag{1}$$

GRPO then optimizes the clipped objective

$$J_{\text{GRPO}}(\theta) = \mathbb{E}[\min(\rho_i A_i,\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)A_i) - \beta D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})], \tag{2}$$

where $\rho_i = \pi_\theta(c_i \mid x)/\pi_{\theta_{\text{old}}}(c_i \mid x)$ is the policy ratio, ϵ is the clipping threshold, β controls KL regularization, and π_{ref} denotes the reference policy.

On-Policy Distillation. On-Policy Distillation (OPD) (Agarwal et al., 2024) transfers knowledge from a teacher policy π_T to a student π_θ along trajectories generated by the student itself. At each step, the teacher evaluates the same student-generated prefix $c_{<t}$. We consider two KL objectives:

$$\mathcal{L}_{\mathrm{FKL}} = \frac{1}{T}\sum_{t=1}^{T} D_{\mathrm{KL}}(\pi_T \| \pi_\theta), \qquad \mathcal{L}_{\mathrm{RKL}} = \frac{1}{T}\sum_{t=1}^{T} D_{\mathrm{KL}}(\pi_\theta \| \pi_T), \tag{3}$$

where the distributions are conditioned on $(x, c_{<t})$. Forward KL is mode covering, encouraging the student to cover predictions supported by the teacher. In contrast, reverse KL is mode seeking, encouraging the student to concentrate on the teacher's most preferred predictions.

3 Dataset

3.1 Data Sourcing and Filtering

We construct our dataset by consolidating SVG–text pairs from five existing SVG datasets, covering both icon-style and illustration-style vector graphics. Since these sources vary substantially in code validity, visual quality, and text–image alignment, we apply a unified filtering pipeline before further processing. Specifically, we retain only samples that satisfy the following criteria. First, the SVG code must be complete, syntactically valid, and renderable. Second, we remove degenerate paths that contain consecutive duplicate drawing commands. For each `<path>`, we represent each drawing command by a signature consisting of its command type, coordinates, and scalar arguments, and reject the sample if two or more identical command signatures occur consecutively. This filter eliminates pathological paths caused by repetitive generation or duplicated segments. Third, we render each SVG and compute its semantic similarity to the paired text using a CLIP (Radford et al., 2021) encoder, retaining only samples with a normalized cosine similarity of at least 0.95. Finally, we perform deduplication and discard SVGs shorter than 100 or longer than 6,000 characters to remove duplicated, trivial, and excessively complex samples. After filtering, the resulting collection contains **196,757** high-quality SVG–text pairs. Table 1 summarizes their source distribution.

Table 1: Source distribution of SVGVerse.

Source	# Samples	Proportion
MMSVG-Icon	129,520	65.83%
SVGX-SFT-GEN	21,293	10.82%
MMSVG-Illustration	17,884	9.09%
SVGX-Core	15,222	7.74%
UniSVG	12,838	6.52%
Total	**196,757**	**100.00%**

3.2 Semantic SVG Reconstruction

The filtered SVGs represent vector graphics using a few complex drawing commands, with limited explicit semantic organization. To obtain semantically structured SVG code while preserving the original visual appearance, we further develop a semantic reconstruction pipeline. Figure 1 illustrates the reconstruction from low-level SVG code to semantic components.

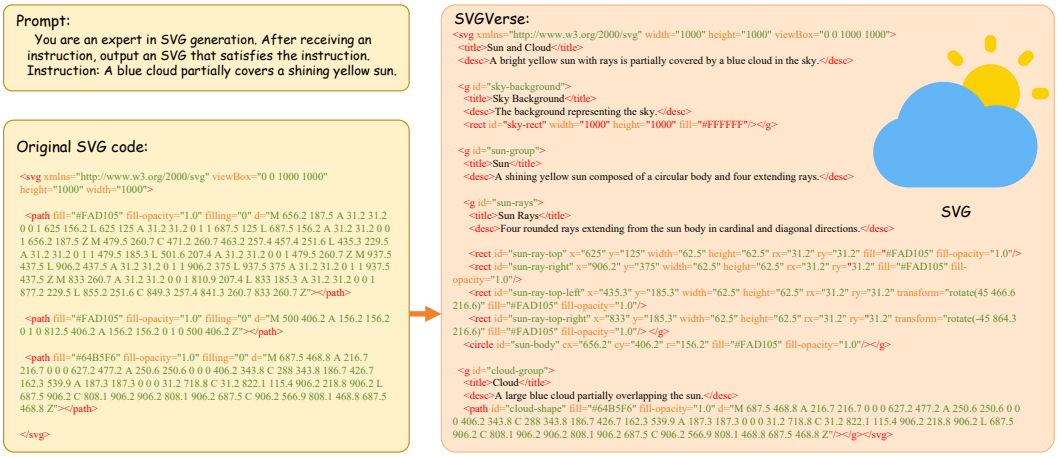

Figure 1: Example of semantic reconstruction from original SVG code to SVGVerse.

We first normalize all filtered SVGs to a 1000×1000 canvas to establish a consistent coordinate space. To reduce model-specific bias and avoid relying on a single reconstruction model, we employ a diverse set of large language models, including Qwen3.8-Max (Team, 2025), GLM-5.2 (Z.ai, 2026), Kimi-K3 (Team, 2026), and Ling-3.0-Flash (inclusionAI, 2026), to reconstruct each SVG conditioned on both its original textual description and SVG code. The reconstruction is constrained to preserve the rendered appearance of the original SVG as faithfully as possible, including its depicted content, spatial layout, color relationships, and relative arrangement of major objects. Under this constraint, the models reorganize the SVG into a more semantically expressive and interpretable representation, introducing clearer object-level grouping and more meaningful structural organization. In particular, visually coherent elements belonging to the same semantic component are organized into explicit groups (`<g>`), yielding SVG code whose structure better reflects the compositional structure of the vector graphic rather than merely its low-level drawing sequence.

Since generative reconstruction may introduce visual discrepancies, we first fine-tune Qwen3-VL-8B (Bai et al., 2025) on annotated Text-to-SVG data to serve as a reconstruction quality evaluator. Given the textual description together with the rendered images of the original and reconstructed SVGs, the evaluator assesses their consistency along three dimensions: content semantics, graphical structure, and visual details. We retain only reconstructions that receive the maximum score on all three dimensions. To further ensure data quality, we recruit trained annotators to review the retained reconstructions, verify their visual fidelity and semantic organization, and manually correct residual semantic or structural errors. Further details are provided in Appendix B.

Importantly, this reconstruction makes semantic planning an explicit and learnable capability in SVG generation. In the original SVG collection, semantic composition is largely implicit in low-level drawing commands, providing no direct supervision for determining which visual components should be generated or how they should be organized. In SVGVerse, each coherent component is represented through structured groups and semantic annotations, making the correspondence between semantic organization and drawing commands explicit. This representation enables the model to learn semantic component selection and organization as a dedicated planning capability, which subsequently provides the basis for separating planning from geometric realization in our method.

4 METHOD

4.1 OVERVIEW

Given a textual description, Text-to-SVG generation aims to produce executable SVG code that faithfully represents the described content. Beyond visual fidelity, the code should also reflect the semantic composition of the vector graphic by organizing drawing commands into meaningful components, making component selection and organization explicit in the generated representation. As illustrated in Figure 2, DEFT follows three stages. First, Two-Stage Supervised Fine Tuning aligns SVG specific tokens and learns semantically structured SVG code, providing a shared initialization for subsequent optimization. Second, since semantic organization and geometric realization involve distinct optimization objectives, Expert GRPO Training separately optimizes a Planning Expert and a Drawing Expert to avoid interference between heterogeneous reward signals. Finally, Command Routed On-Policy Distillation transfers their complementary knowledge according to the semantic roles of SVG commands, integrating both capabilities into a unified model.

4.2 TWO STAGE SUPERVISED INITIALIZATION

To provide a reliable initialization for subsequent expert optimization, we adopt the SVG specific token design introduced in InternSVG (Wang et al., 2026c) and perform supervised fine tuning (SFT) in two stages. Both stages optimize the standard autoregressive objective

$$\mathcal{L}_{\mathrm{SFT}}(\theta; \mathcal{D}) = -\mathbb{E}_{(x,c)\sim\mathcal{D}} \left[\frac{1}{T} \sum_{t=1}^{T} \log \pi_\theta \left(c_t \mid x, c_{<t} \right) \right], \tag{4}$$

where $\mathcal{D}$ denotes the training set. In the first stage, we sample 1M instances from the SAgoge dataset to adapt the model to the extended SVG vocabulary and stabilize the embeddings of the introduced tokens. In the second stage, we train the model on the SFT split of our reconstructed dataset, where

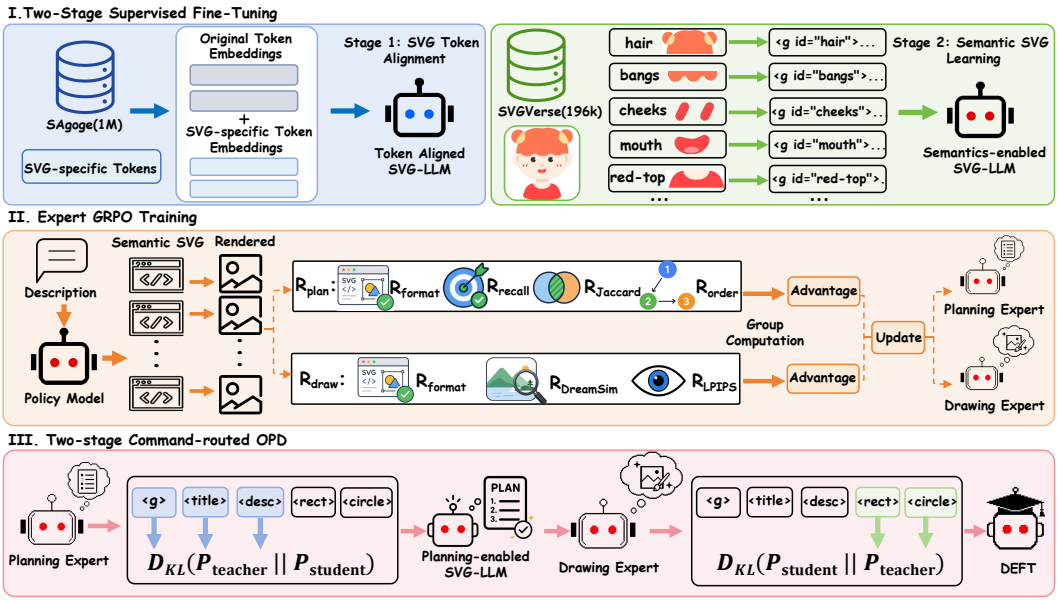

Figure 2: Overview of **DEFT**. **I. Two Stage Supervised Fine Tuning** learns SVG token representations and semantic component structure. **II. Expert GRPO Training** specializes Planning and Drawing Experts for semantic organization and geometric realization. **III. Command Routed On-Policy Distillation** integrates both experts through command specific supervision.

the SVG code is organized according to the semantic components. This supervision encourages the model to first identify the components required by a textual description and then associate each component with its corresponding drawing commands. As a result, the model acquires a structured correspondence between semantic composition and geometric realization.

4.3 Specialized Expert Optimization

The generation of SVG code requires both semantic component organization and geometric realization, which are evaluated through structural correspondence and visual similarity, respectively. Combining these objectives into a single GRPO reward can introduce competing optimization signals. A sample with correct component planning but poor drawing quality may receive a low reward, while visually similar code with incorrect component organization may still be reinforced. In addition, gains in one objective can offset losses in the other after scalar aggregation, allowing the overall reward to increase despite degradation in one capability. We therefore optimize a Planning Expert and a Drawing Expert independently from the same supervised initialization and integrate their complementary knowledge in the subsequent distillation stage.

Planning Expert. The Planning Expert focuses on the semantic composition and organization of SVG code. For each generated SVG and its reference, we extract the ordered component identifiers from the **<g>** groups, obtaining a predicted component sequence $P = (p_1, \ldots, p_n)$ and a reference sequence $G = (g_1, \ldots, g_m)$. Since components with the same semantic role may use different identifiers, exact string matching alone can miss valid correspondences. Therefore, we first normalize component identifiers by lowercasing, tokenizing them into alphabetic terms, and applying singularization, followed by lexical matching. For components that remain unmatched, we encode their identifiers using BERT and associate pairs whose cosine similarity is at least 0.95. This procedure preserves reliable matches from component names while allowing semantically equivalent components with different identifiers to be associated.

Based on the matched components, we evaluate planning quality in terms of component coverage, selection, and ordering:

$$R_{\text{recall}} = \frac{|\mathcal{P} \cap \mathcal{G}|}{|\mathcal{G}|}, \quad R_{\text{Jaccard}} = \frac{|\mathcal{P} \cap \mathcal{G}|}{|\mathcal{P} \cup \mathcal{G}|}, \quad R_{\text{order}} = \frac{\text{LCS}(P, G)}{\max(|P|, |G|)}. \tag{5}$$

Here, $\mathcal{P}$ and $\mathcal{G}$ denote the sets of matched predicted and reference components derived from the sequences P and G, respectively; $|P|$ and $|G|$ denote their sequence lengths, and $\mathrm{LCS}(P, G)$ denotes the length of their longest common subsequence. We further introduce a format reward R_{format} that equals 1 when the generated code forms a complete and valid structure and 0 otherwise. The final reward for the Planning Expert is

$$R_{\text{plan}} = R_{\text{format}} \cdot (R_{\text{recall}} + R_{\text{Jaccard}} + R_{\text{order}}) . \tag{6}$$

Drawing Expert. The Drawing Expert focuses on the geometric realization of semantic components. We render the generated SVG code and compare the resulting vector graphic with its reference using DreamSim (Fu et al., 2023) and LPIPS (Zhang et al., 2018). DreamSim measures perceptual similarity at the semantic level, while LPIPS captures differences in visual appearance and local structure. Since DreamSim and LPIPS produce distances with different numerical scales, we map each distance to a similarity reward in $[0, 1]$ using a sigmoid transformation. We assign equal weights to the two rewards and define the Drawing Expert reward as

$$R_{\text{draw}} = R_{\text{format}} \cdot (R_{\text{DreamSim}} + R_{\text{LPIPS}}) . \tag{7}$$

4.4 Command Routed On-Policy Distillation

The Planning and Drawing experts acquire complementary capabilities through different reward signals, but their advantages are concentrated in different parts of SVG code. Distilling either expert over the entire sequence would also transfer its predictions on tokens outside its specialization, which may interfere with knowledge learned from the other expert. We therefore integrate the two experts through selective distillation, transferring planning knowledge before drawing knowledge. This order follows the generation process of SVG code, where semantic components are organized before their geometric realization is specified.

We perform command routed on-policy distillation in two sequential stages. In the first stage, the student generates SVG trajectories and the Planning Expert predicts next token distributions conditioned on the same student generated prefixes. Distillation supervision is restricted to a planning command set $\mathcal{C}_{\text{plan}}$ that covers semantic organization, including `<g>`, `<title>`, and `<desc>`. The resulting student is then used to initialize the second stage, where the Drawing Expert provides supervision on newly generated student trajectories. This stage uses a drawing command set $\mathcal{C}_{\text{draw}}$ that covers geometric realization, including `<path>`, `<rect>`, `<circle>`, and `<polygon>` together with their drawing attributes. In both stages, the distillation loss is computed only for tokens associated with commands in the corresponding set, while all other tokens receive no distillation loss. The complete definitions of both command sets are provided in Appendix C.

We use different KL directions for the two distillation stages to reflect the distinct distributions of planning and drawing decisions. For planning distillation, we minimize the forward KL divergence

$$\mathcal{L}_{\text{plan}} = D_{\text{KL}} (\pi_{\text{plan}} \,\|\, \pi_\theta) , \tag{8}$$

which encourages the student to preserve the distribution supported by the Planning Expert. Planning decisions are constrained by the semantic components required by the textual description, making coverage of the teacher distribution important. In contrast, a semantic component can admit multiple valid geometric realizations, and forcing the student to reproduce a particular teacher behavior can constrain this flexibility. We therefore use reverse KL for drawing distillation,

$$\mathcal{L}_{\text{draw}} = D_{\text{KL}} (\pi_\theta \,\|\, \pi_{\text{draw}}) , \tag{9}$$

which avoids requiring the student to cover the full teacher distribution and preserves flexibility in geometric realization.

5 Experiments

5.1 Experimental Setup

We use Qwen3-VL-8B-Instruct as the base model. Our main experiments are conducted on our SVG dataset. We use 10,000 samples for GRPO optimization and 10,000 samples for on-policy distillation, with a test set of 512 samples for evaluation. We also evaluate the resulting model on MMSVG-Bench (Yang et al., 2025) and the SArena Benchmark (Wang et al., 2026c) to measure generalization beyond our dataset. Implementation details are provided in Appendix C.

Table 2: Results on SArena-Icon and SArena-Illustration. Bold and underlined values indicate the best and second best results, respectively.

Model	SArena-Icon				SArena-Illustration			
	FID ↓	T2I ↑	I2I ↑	Tokens	FID ↓	T2I ↑	I2I ↑	Tokens
Traditional SVG Methods								
IconShop	32.288	20.894	70.922	1.6k	35.259	13.972	63.794	2.7k
VectorFusion	16.594	22.992	70.308	33k	28.198	22.638	71.220	33k
SVGDreamer	26.612	20.329	69.975	132k	32.410	21.026	69.775	132k
Large Language Models								
Qwen3-VL-8B	16.875	22.578	73.187	374	32.124	16.893	65.743	535
InternVL3-8B	23.061	21.897	71.450	269	36.736	18.493	61.964	493
Gemma-3-27B	15.145	22.526	73.277	249	27.838	21.486	67.255	613
Llama-4-Maverick	14.931	23.570	75.816	265	30.835	21.872	67.366	551
Qwen2.5-VL-72B	15.948	22.946	73.681	275	29.521	20.923	65.349	527
InternVL3.5-241B	11.265	22.534	76.807	991	43.339	18.191	65.689	2.9k
GPT-4o	15.178	24.617	77.742	246	28.124	23.637	70.696	473
Gemini-2.5-Flash	16.720	24.658	78.218	451	28.865	<u>24.800</u>	<u>74.796</u>	1.2k
Claude-Sonnet-4	15.840	25.421	80.579	444	27.294	23.094	74.525	1.0k
SVG-LLMs								
LLM4SVG-7B	21.939	19.458	70.726	705	48.704	15.468	62.933	1.2k
OmniSVG-3B	28.292	21.679	74.831	1.7k	42.756	16.861	64.815	4.5k
InternSVG-8B	**8.715**	23.916	80.911	1.0k	<u>22.397</u>	21.116	74.662	8.1k
CTRL-S-8B	11.584	**25.944**	<u>82.291</u>	346	–	–	–	–
IntroSVG-7B	22.145	23.865	77.828	657	33.548	21.799	72.105	1.2k
DEFT-8B	<u>10.723</u>	<u>25.800</u>	**83.046**	674	**21.136**	**25.941**	**77.950**	1.2k

Table 3: Quantitative results on MMSVG-Icon and MMSVG-Illustration.

Method	MMSVG-Icon				MMSVG-Illustration			
	Tokens	CLIP ↑	Aesthetic ↑	HPS ↑	Tokens	CLIP ↑	Aesthetic ↑	HPS ↑
VectorFusion	66.2k	0.240	<u>4.76</u>	0.237	66.1k	0.185	**4.94**	0.226
SVGDreamer	132.0k	0.207	4.26	0.221	132.0k	0.201	4.37	0.221
Chat2SVG	0.6k	<u>0.299</u>	4.41	<u>0.247</u>	1.0k	<u>0.283</u>	4.45	**0.250**
IconShop	2.0k	0.288	4.55	0.244	2.6k	0.233	4.46	0.224
OmniSVG-4B	3.8k	0.275	4.62	0.244	5.8k	0.226	4.56	0.232
OmniSVG-8B	5.7k	0.276	4.60	0.242	6.9k	0.231	4.51	0.232
InternSVG-8B	1.3k	0.289	4.56	<u>0.247</u>	2.6k	0.225	4.59	0.222
IntroSVG-7B	1.3k	0.223	4.51	0.220	1.4k	0.222	4.51	0.218
Qwen3-VL-8B	0.8k	0.274	4.41	0.243	1.2k	0.237	4.42	0.231
DEFT-8B	1.0k	**0.322**	**4.77**	**0.254**	1.5k	**0.294**	<u>4.64</u>	<u>0.235</u>

5.2 Main Results

Tables 2, 3, and 4 report the main results on SArena, MMSVG-Bench, and SVGVerse. On SVGVerse, DEFT achieves the best performance across all four drawing-quality metrics, outperforming both general-purpose models and specialized SVG-LLMs. Notably, these gains are obtained after integrating the Planning and Drawing Experts into a single policy, indicating that incorporating semantic planning does not come at the expense of geometric realization.

The gains also transfer to external benchmarks, but exhibit different patterns across generation settings. On SArena-Icon, DEFT achieves the highest I2I score (83.046) and a T2I score (25.800)

Table 4: Quantitative results on SVGVerse.

Model	DINO ↑	SSIM ↑	PSNR ↑	MSE ↓
Large Language Models				
Qwen3-VL-8B	0.834	0.646	8.805	0.175
GLM-5.2	0.881	0.636	8.978	0.167
Ling-3.0-Flash	0.859	0.640	8.872	0.172
DeepSeek-V4-Flash	0.870	0.638	9.126	0.159
Kimi-K2.7-Code	0.875	0.655	9.231	0.160
DeepSeek-V4-Pro	0.866	0.640	8.999	0.164
SVG-LLMs				
OmniSVG-4B	0.877	0.661	10.293	0.150
OmniSVG-8B	0.885	0.668	10.766	0.145
InternSVG-8B	0.892	0.681	9.419	0.138
DEFT-8B	**0.897**	**0.697**	**12.108**	**0.132**

Table 5: Ablation studies on SVGVerse.

Method	Drawing Quality				Planning Quality	
	DINO ↑	SSIM ↑	PSNR ↑	MSE ↓	F1 ↑	LCS ↑
Qwen3-VL-8B	0.834	0.646	8.805	0.175	–	–
+ SFT	0.859	0.653	9.886	0.168	0.598	0.780
w/o Drawing OPD	0.874	0.662	9.927	0.161	0.683	0.828
w/o Planning OPD	0.884	0.676	11.164	0.143	0.639	0.784
w/ Both FKL	0.884	0.661	10.263	0.162	0.651	0.802
w/ Both RKL	0.883	0.656	10.385	0.160	0.640	0.792
w/o Routing	0.875	0.662	10.550	0.160	0.643	0.780
w/ Reverse Order	0.884	0.662	11.244	0.154	0.621	0.763
DEFT-8B	**0.897**	**0.697**	**12.108**	**0.132**	**0.689**	**0.833**

Table 6: Analysis of reward interference between planning and drawing optimization.

Model	Drawing Quality				Planning Quality	
	DINO↑	SSIM↑	PSNR↑	MSE↓	F1↑	LCS↑
Qwen3-VL-8B	0.834	0.646	8.805	0.175	–	–
+ SFT	0.859	0.653	9.886	0.168	0.598	0.780
Planning Expert	0.874	0.651	9.478	0.166	**0.644**	**0.819**
Drawing Expert	**0.895**	**0.678**	10.194	**0.145**	0.624	0.760
Joint Multi-Reward RL	0.878	0.662	**10.229**	0.164	0.616	0.799

Table 7: Ablation study on the reward design for the Planning and Drawing experts.

Reward	DINO↑	SSIM↑	PSNR↑	MSE↓	F1↑	LCS↑
$R_{\text{format}} \cdot R_{\text{recall}}$	0.873	0.646	9.078	0.168	0.604	0.782
$R_{\text{format}} \cdot (R_{\text{recall}} + R_{\text{order}})$	0.868	0.644	9.401	0.172	0.634	0.798
Full R_{plan}	**0.874**	**0.651**	**9.478**	**0.166**	**0.644**	**0.819**
$R_{\text{format}} \cdot R_{\text{DreamSim}}$	0.881	0.660	9.267	0.160	0.622	0.747
$R_{\text{format}} \cdot R_{\text{LPIPS}}$	0.876	0.664	9.638	0.155	0.612	0.749
Full R_{draw}	**0.895**	**0.678**	**10.194**	**0.145**	**0.624**	**0.760**

comparable to the best-performing CTRL-S, while requiring only 674 output tokens. On the more complex SArena-Illustration benchmark, DEFT achieves the best FID, T2I, and I2I scores among all compared methods. These results suggest that the learned improvements generalize beyond the SVGVerse distribution, particularly to illustration-style graphics with richer visual composition.

A similar trend is observed on MMSVG-Bench. DEFT achieves the best CLIP, Aesthetic, and HPS scores on MMSVG-Icon and the highest CLIP score on MMSVG-Illustration. Although it does not dominate every perceptual metric on the illustration subset, its consistent text–image alignment across both categories provides further evidence that the improvements are not confined to the reconstruction style of SVGVerse. Overall, DEFT achieves a strong balance between visual fidelity and semantic alignment across both in-domain and external evaluation settings.

5.3 Ablation Studies

Table 5 examines the contribution of each component of DEFT on SVGVerse. Two stage SFT improves drawing quality and equips the model with the structured component representations required for subsequent expert optimization. Removing either OPD branch reveals clear specialization between the two experts. Removing Drawing OPD preserves planning gains but weakens drawing quality, while removing Planning OPD retains stronger drawing performance but degrades planning quality. These results confirm that the two experts provide complementary supervision for semantic organization and geometric realization.

We further study the KL divergence used for expert distillation. Using FKL or RKL for both experts consistently performs worse. Planning decisions are more constrained by the semantic components required by the textual description, making FKL suitable for preserving the distribution supported by the Planning Expert. In contrast, geometric realization admits multiple valid SVG implementations, for which RKL avoids forcing the student to cover the complete teacher distribution. The results therefore support using different divergence objectives for the two forms of expertise.

Finally, removing command routing degrades both planning and drawing quality, showing that unrestricted expert supervision introduces interference outside each expert's specialization. Command routing avoids this conflict by restricting distillation to the corresponding SVG regions. Reversing the distillation order also performs worse than DEFT, supporting the planning to drawing order: semantic component organization provides the structural basis for subsequent geometric realization.

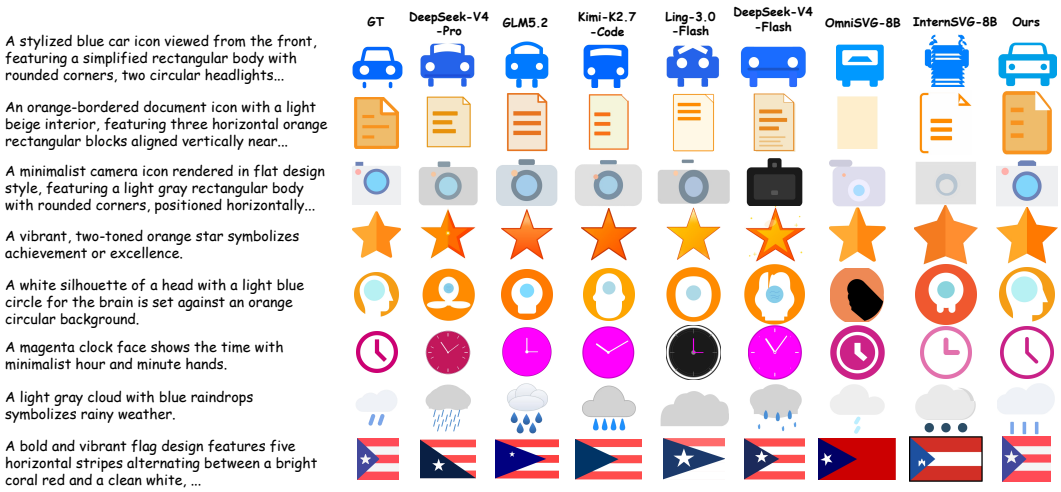

Figure 3: Qualitative comparison between the proposed DEFT and other Text-to-SVG methods.

Together, these results validate the complementary expert branches, command routing, asymmetric KL objectives, and sequential distillation design of DEFT.

5.4 Analysis of Specialized Expert Optimization

To examine the interaction between planning and drawing objectives, we compare four training settings in Table 6. The SFT model serves as the common initialization. Starting from this model, the *Planning Expert* is optimized only with the planning reward for semantic component selection and organization, while the *Drawing Expert* is optimized only with the drawing reward for geometric realization and visual quality. *Joint Multi-Reward RL* instead optimizes a single policy by combining both rewards under the same GRPO setting, with the planning and drawing rewards normalized separately before being summed into a unified reward. The results show clear specialization, with planning-only optimization achieving the strongest planning quality and drawing-only optimization achieving the strongest overall drawing quality at the cost of planning performance. Joint optimization improves upon the SFT initialization in both aspects, yet fails to retain the respective strengths of the two specialized experts, indicating that aggregating heterogeneous objectives within a single policy leads to competing optimization signals and a compromise between the two capabilities.

We further analyze the reward design of the two experts in Table 7. For the Planning Expert, using component recall alone improves component coverage, while incorporating the ordering reward further strengthens planning quality. The full planning reward achieves the best overall planning performance, showing that component coverage, selection, and ordering provide complementary supervision for semantic organization. For the Drawing Expert, DreamSim and LPIPS individually improve drawing quality from different perspectives, while their combination yields the strongest overall performance. This confirms that semantic perceptual similarity and local visual similarity provide complementary signals for geometric realization.

6 Conclusion

In this work, we introduce DEFT, a dual-expert framework for Text-to-SVG generation. We first construct SVGVerse, a semantically structured SVG dataset that provides explicit component organization for learning structured SVG representations. Based on this data, DEFT performs two stage supervised fine tuning, independently optimizes Planning and Drawing experts with specialized reinforcement learning objectives, and subsequently integrates their knowledge through Command Routed On-Policy Distillation. Extensive experiments demonstrate that DEFT consistently outperforms strong baseline methods and improves both semantic organization and drawing quality. We hope this work inspires further research on semantically structured vector graphics and effective integration of complementary generative capabilities.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, 33:16351–16361, 2020.

Hanqi Chen, Zhongyin Zhao, Ye Chen, Zhujin Liang, and Bingbing Ni. Svgthinker: Instruction-aligned and reasoning-driven text-to-svg generation. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 11004–11012, 2025a.

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

Zehao Chen and Rong Pan. Svgbuilder: Component-based colored svg generation with text-guided autoregressive transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 2358–2366, 2025.

Louis Clouâtre and Marc Demers. Figr: Few-shot image generation with reptile. *arXiv preprint arXiv:1901.02199*, 2019.

Kevin Frans, Lisa Soros, and Olaf Witkowski. Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. *Advances in Neural Information Processing Systems*, 35: 5207–5218, 2022.

Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. *arXiv preprint arXiv:2306.09344*, 2023.

inclusionAI. Ling-3.0-flash. `https://huggingface.co/inclusionAI/Ling-3.0-flash`, 2026.

Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1911–1920. IEEE, 2023.

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

Sifan Li, Yujun Cai, Hongkai Chen, and Yiwei Wang. Geosvg-rl: Geometry-aware reinforcement learning for layout-constrained text-to-svg diagram generation. *arXiv preprint arXiv:2605.25447*, 2026.

Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39(6): 1–15, 2020.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, Jinhao Dong, Zhifang Sui, and Fuli Luo. MOPD: multi-teacher on-policy distillation for capability integration in LLM post-training. *CoRR*, abs/2606.30406, 2026. doi: 10.48550/ARXIV.2606.30406. URL `https://doi.org/10.48550/arXiv.2606.30406`.

Kunato Nishina and Yusuke Matsui. Svgeditbench: A benchmark dataset for quantitative assessment of llm's svg editing capabilities. *arXiv preprint arXiv:2404.13710*, 2024.

Kunato Nishina and Yusuke Matsui. Svgeditbench v2: A benchmark for instruction-based svg editing. *arXiv preprint arXiv:2502.19453*, 2025.

Chengyuan Peng. Scalable vector graphics (svg). In *Research Seminar on Interactive Digital Media*, pp. 22. Citeseer, 2000.

Antoine Quint. Scalable vector graphics. *IEEE MultiMedia*, 10(3):99–102, 2003.

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

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Kimi Team. Kimi K3: open frontier intelligence. *CoRR*, abs/2607.24653, 2026. doi: 10.48550/ARXIV.2607.24653. URL https://doi.org/10.48550/arXiv.2607.24653.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Yael Vinker, Ehsan Pajouheshgar, Jessica Y Bo, Roman Christian Bachmann, Amit Haim Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. Clipasso: Semantically-aware object sketching. *ACM Transactions on Graphics (TOG)*, 41(4):1–11, 2022.

Feiyu Wang, Zhiyuan Zhao, Yuandong Liu, Da Zhang, Junyu Gao, Hao Sun, and Xuelong Li. Svgen: Interpretable vector graphics generation with large language models. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 9608–9617, 2025.

Feiyu Wang, Jiayuan Yang, Zhiyuan Zhao, Da Zhang, Bingyu Li, Peng Liu, and Junyu Gao. Introsvg: Learning from rendering feedback for text-to-svg generation via an introspective generator-critic framework. *arXiv preprint arXiv:2603.09312*, 2026a.

Haomin Wang, Qi Wei, Qianli Ma, Shengyuan Ding, Jinhui Yin, Kai Chen, and Hongjie Zhang. Reliable reasoning in svg-llms via multi-task multi-reward reinforcement learning. *arXiv preprint arXiv:2603.16189*, 2026b.

Haomin Wang, Jinhui Yin, Qi Wei, Wenguang Zeng, Lixin Gu, Shenglong Ye, Zhangwei Gao, Yaohui Wang, Yanting Zhang, Yuanqi Li, et al. Internsvg: Towards unified svg tasks with multimodal large language models. In *International Conference on Learning Representations*, volume 2026, pp. 38649–38705, 2026c.

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *ACM Transactions on Graphics (TOG)*, 42(6):1–14, 2023.

Ronghuan Wu, Wanchao Su, and Jing Liao. Chat2svg: Vector graphics generation with large language models and image diffusion models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23690–23700. IEEE, 2025.

Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svgdreamer: Text guided svg generation with diffusion model. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4546–4555. IEEE, 2024.

Ximing Xing, Yandong Guan, Jing Zhang, Dong Xu, and Qian Yu. Reason-svg: Hybrid reward RL for aha-moments in vector graphics generation. *CoRR*, abs/2505.24499, 2025a. doi: 10.48550/ARXIV.2505.24499. URL https://doi.org/10.48550/arXiv.2505.24499.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. Empowering llms to understand and generate complex vector graphics. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19487–19497. IEEE, 2025b.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/a510f05a574d4203ef3952973672fe2f-Abstract-Conference.html.

Z.ai. GLM-5.2. https://github.com/zai-org/GLM-5, 2026.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018.

AI USE STATEMENT

In this work, we used generative AI tools as part of the construction of SVGVerse. Generative AI tools were also used to assist with language editing and improving the readability of the manuscript. They were not used to generate experimental results or make final scientific conclusions. All AI-assisted content and data processing procedures were reviewed by the authors, who take full responsibility for the final content of the paper, including its text, claims, experimental results, and artifacts.

A RELATED WORK

A.1 TEXT-TO-SVG GENERATION

Text-to-SVG generation aims to translate natural language descriptions into executable vector graphics. Existing methods mainly follow two paradigms: optimization based generation and autoregressive code generation. Optimization based methods represent geometric primitives or path parameters as optimizable variables and refine them through differentiable rendering or guidance from pretrained vision models. DiffVG (Li et al., 2020) enables gradients to propagate through vector graphics rasterization, while CLIPDraw (Frans et al., 2022) and CLIPasso (Vinker et al., 2022) optimize vector primitives according to CLIP based semantic objectives. VectorFusion (Jain et al., 2023) and SVGDreamer (Xing et al., 2024) further introduce diffusion priors to improve visual quality, and Chat2SVG (Wu et al., 2025) combines language model generated SVG templates with subsequent geometric refinement. Although these approaches can achieve strong visual fidelity, they typically require iterative optimization for each input and provide limited control over the semantic organization of the resulting SVG code.

Autoregressive methods instead formulate SVG generation as structured sequence generation and directly predict SVG code with Transformers, LLMs, or VLMs. Early approaches such as IconShop (Wu et al., 2023), LLM4SVG (Xing et al., 2025b), and StarVector (Rodriguez et al., 2025) demonstrate the effectiveness of learning SVG syntax and geometric representations from large

scale corpora. More recent models introduce SVG-specific representations, multimodal supervision, or explicit reasoning to improve generation capability. InternSVG (Wang et al., 2026c) extends the vocabulary with SVG specific tokens and supports multiple SVG understanding and generation tasks, while OmniSVG (Yang et al., 2025) develops a unified multimodal framework for complex vector graphics generation. SVGen (Wang et al., 2025), SVGThinker (Chen et al., 2025a), Reason-SVG (Xing et al., 2025a), and CTRL-S (Wang et al., 2026b) further incorporate intermediate reasoning or structured planning into SVG generation. GeoSVG-RL (Li et al., 2026) explicitly introduces planning for layout constrained generation, while IntroSVG (Wang et al., 2026a) uses rendering feedback in a generator–critic loop to iteratively refine generated SVG code. Despite this progress, semantic component organization and geometric realization are still generally learned or optimized within a single policy. In contrast, DEFT treats planning what to draw and realizing how to draw as specialized capabilities, optimizes them independently, and subsequently integrates their complementary knowledge into a unified model.

A.2 Reinforcement Learning and On-Policy Distillation

Reinforcement learning has become an important approach for improving language models beyond supervised fine tuning. Proximal Policy Optimization (PPO) (Schulman et al., 2017) optimizes a policy under a clipped objective while controlling deviation from a reference model. Group Relative Policy Optimization (GRPO) (Shao et al., 2024) removes the need for a separate value model and estimates advantages from groups of sampled responses, making it suitable for tasks where outputs can be evaluated with automatic rewards. Recent SVG generation methods have adopted reinforcement learning to optimize properties that are difficult to capture through token level supervision alone. Reason-SVG (Xing et al., 2025a), CTRL-S (Wang et al., 2026b), and GeoSVG-RL (Li et al., 2026), for example, construct task specific rewards for visual fidelity, structural correctness, reasoning, or layout consistency. These approaches demonstrate the benefit of reinforcement learning for SVG generation, but multiple objectives are commonly aggregated within a single policy. DEFT instead separates semantic planning and geometric realization into two expert optimization processes with capability specific rewards, avoiding direct competition between heterogeneous objectives.

Knowledge distillation provides another mechanism for transferring behavior from a stronger teacher to a student model. Conventional distillation typically trains the student on teacher generated data or teacher distributions under externally collected contexts, which can introduce a mismatch between training states and those encountered by the student during generation. On-Policy Distillation (OPD) (Agarwal et al., 2024) addresses this issue by generating trajectories from the student and querying the teacher on the same student generated prefixes, allowing supervision to remain aligned with the student's own state distribution. Recently, Multi-Teacher On-Policy Distillation (MOPD) (Ma et al., 2026) extends this paradigm to capability integration by distilling multiple domain-specialized RL teachers into a unified student on its own rollouts. Forward and reverse KL divergence provide two common distillation objectives with different distributional properties: forward KL encourages coverage of teacher supported predictions, whereas reverse KL favors predictions assigned high probability by the teacher.

A.3 SVG Datasets and Evaluation Benchmarks

The development of SVG generation models has been supported by increasingly large and diverse vector graphics datasets. Early datasets such as FIGR-8-SVG (Clouâtre & Demers, 2019) and DeepSVG (Carlier et al., 2020) primarily contain relatively simple or monochrome vector graphics, while SVGBuilder (Chen & Pan, 2025) extends autoregressive SVG generation to larger collections of colored graphics. With the emergence of large language and vision language models, subsequent works construct SVG–text pairs at larger scales. LLM4SVG (Xing et al., 2025b) and StarVector (Rodriguez et al., 2025) augment SVG corpora with textual supervision, while recent datasets provide richer annotations or support broader SVG tasks. SVG-1M introduced by SVGen (Wang et al., 2025) contains reasoning annotations for SVG generation, SVGX-DwT-10k from Reason-SVG (Xing et al., 2025a) provides design reasoning supervision, UniSVG (Li et al., 2025) supports multiple vector graphics understanding and generation tasks, and MMSVG-2M (Yang et al., 2025) covers large scale icons, illustrations, and more complex vector graphics.

Several benchmarks have also been introduced to evaluate different aspects of SVG understanding and generation. SVGEditBench (Nishina & Matsui, 2024; 2025) focuses on instruction based SVG editing, while SVGenius (Chen et al., 2025b) evaluates SVG understanding, editing, and generation capabilities. MMSVG-Bench (Yang et al., 2025) evaluates generation across diverse vector graphic categories, and SArena provides a complementary evaluation setting for Text-to-SVG generation. Existing datasets and benchmarks have substantially advanced SVG modeling, but their SVG code does not necessarily expose an explicit correspondence between the semantic composition of a vector graphic and its underlying drawing commands. SVGVerse complements these resources by reorganizing visually coherent elements into explicit semantic groups while preserving the rendered content, providing structured supervision for learning component organization and geometric realization.

B Dataset Details

B.1 Dataset Statistics

To further analyze the structural characteristics introduced by our semantic reconstruction pipeline, we compare SVGVerse with the original SVG collection before reconstruction. We report statistics on SVG code length, semantic components, and tag complexity. Code length is measured by the number of characters in the SVG code. A semantic component is defined as a `<g>` element with an explicit `id` attribute, while tag complexity is measured by the total number of opening or self-closing SVG tags in each sample.

Compared with the original collection, SVGVerse exhibits substantially richer structural organization. The average SVG code length increases from 1,167.8 to 2,116.5 characters, while the average number of SVG tags rises from 3.65 to 22.49. More importantly, the original SVGs contain no explicit semantic components defined by `<g>` elements with `id` attributes, whereas SVGVerse contains 4.50 such components on average, with the number of components ranging from 1 to 41 per sample. These statistics show that semantic reconstruction introduces explicit component organization and substantially enriches the structural composition of SVG code.

Table 8 further compares the usage of individual SVG tags. The original collection is dominated by `<path>` elements, which appear in every sample, while other primitives occur only occasionally. In contrast, SVGVerse consistently introduces `<g>`, `<title>`, and `<desc>` to encode semantic component structure and descriptions. It also exhibits a more diverse use of geometric primitives, including `<rect>`, `<circle>`, and `<polygon>`. This distribution reflects the richer structural organization introduced by semantic reconstruction rather than merely an increase in code length.

Table 8: Statistics of SVG tags in the original collection and SVGVerse. Avg. denotes the average number of occurrences per sample, and Occ. denotes the percentage of samples containing the corresponding tag.

	Original Dataset		SVGVerse	
Tag	Avg.	Occ.	Avg.	Occ.
`<title>`	–	–	6.09	100.0%
`<desc>`	–	–	5.92	100.0%
`<g>`	<0.01	0.2%	4.50	100.0%
`<path>`	2.63	100.0%	2.01	75.5%
`<rect>`	<0.01	0.3%	1.75	54.9%
`<circle>`	0.02	0.8%	0.74	30.0%
`<polygon>`	–	–	0.32	15.0%
`<line>`	–	–	0.08	3.1%
`<ellipse>`	<0.01	0.1%	0.07	3.3%

B.2 Why Semantic Structure Matters

Beyond making the organization of SVG code more interpretable, the semantic structure introduced by SVGVerse provides component-level supervision for SVG generation. In conventional SVG code, the visual realization of an object is often entangled within low-level drawing commands,

making it difficult for the model to associate a semantic concept with the commands responsible for rendering it. SVGVerse instead groups visually coherent elements into explicit components and annotates them with semantic descriptions. During training, this structure encourages the model to establish correspondences between component semantics and their geometric realizations. Consequently, semantic components can serve as transferable compositional representations, allowing the model to leverage learned correspondences between component semantics and drawing commands when similar concepts appear in new generations.

B.3 Examples of SVGVerse

Figure 4 provides qualitative examples of the semantic reconstruction in SVGVerse. The original SVG code often represents the vector graphic as a sequence of low level drawing commands with limited semantic organization. After reconstruction, visually coherent elements are reorganized into explicit semantic groups, with `<title>` and `<desc>` annotations describing the role of each component. This transformation makes the code structure better reflect the semantic composition of the vector graphic while preserving its original visual content. For reproducibility, we additionally provide the instruction prompt used for semantic SVG reconstruction in Figure 5.

C Implementation Details

We use Qwen3-VL-8B-Instruct as the base model. During the two stage SFT, we set the learning rate to 1×10^{-4} in the first stage and 5×10^{-5} in the second stage. SFT is conducted on 8 H20 GPUs with a global batch size of 64. For expert optimization, we implement GRPO with the verl framework. Both the Planning and Drawing experts are trained for two epochs on 16 GPUs with a global batch size of 32 and a learning rate of 1×10^{-6}, sampling eight responses for each prompt and requiring about 30 hours. For on-policy distillation, we compute KL divergence over the full vocabulary and sample four student trajectories for each training instance. Distillation is performed on 16 GPUs with a global batch size of 96 and a learning rate of 1×10^{-6}, requiring about 24 hours. All results obtained from our experiments are averaged over five inference runs with different random seeds.

Test Set Contamination Mitigation. We explicitly exclude all instances belonging to the MMSVG-Bench and SArena-Bench test sets from our training data and verify that no benchmark test instance is included in any stage of SFT, GRPO, or on-policy distillation. To reduce potential test contamination from overlap with the training data of evaluated baselines, we further process the SVGVerse test set by rewriting both the SVG code and textual descriptions. In addition to the semantic SVG reconstruction described in the main paper, we paraphrase each textual description using DeepSeek-V4-Flash (Xu et al., 2026) while preserving its original semantics, thereby reducing direct code and text overlap with existing training corpora.

Design of SVG-specific Tokens. To better model the structured syntax and numerical parameters of SVG code, we extend the original Qwen3-VL tokenizer with a dedicated SVG-specific vocabulary, following the tokenization design adopted by prior SVG-LLMs (Wang et al., 2026c;b). As summarized in Table 9, the added vocabulary consists of three types of tokens. First, 49 tag-level tokens represent frequently used SVG structural and graphical elements, including container tags such as `<svg` and `<g`, geometric primitives such as `<path`, `<rect`, and `<circle`, as well as text, gradient, clipping, and filter elements. Second, 35 attribute-level tokens represent commonly used geometric and styling fields, such as `d="`, `fill="`, `stroke="`, and `transform="`. Third, dedicated numeric tokens are introduced for frequently occurring coordinate and parameter values, including integer tokens spanning -128 to 128, together with one-decimal and two-decimal fractional tokens. These tokens reduce the fragmentation of frequent SVG patterns into long subword sequences and provide a more compact representation of SVG syntax and geometric parameters. For initialization, each newly introduced token is first decomposed using the original tokenizer, and its embedding is initialized as the average of the embeddings of its constituent subwords, preserving compatibility with the pretrained embedding space.

Command Routing Sets for Expert Distillation. For command routed distillation, we partition SVG tokens into three categories according to their semantic roles. The planning command set $\mathcal{C}_{\text{plan}}$ contains tokens associated with semantic component organization, including `<g>`, `</g>`,

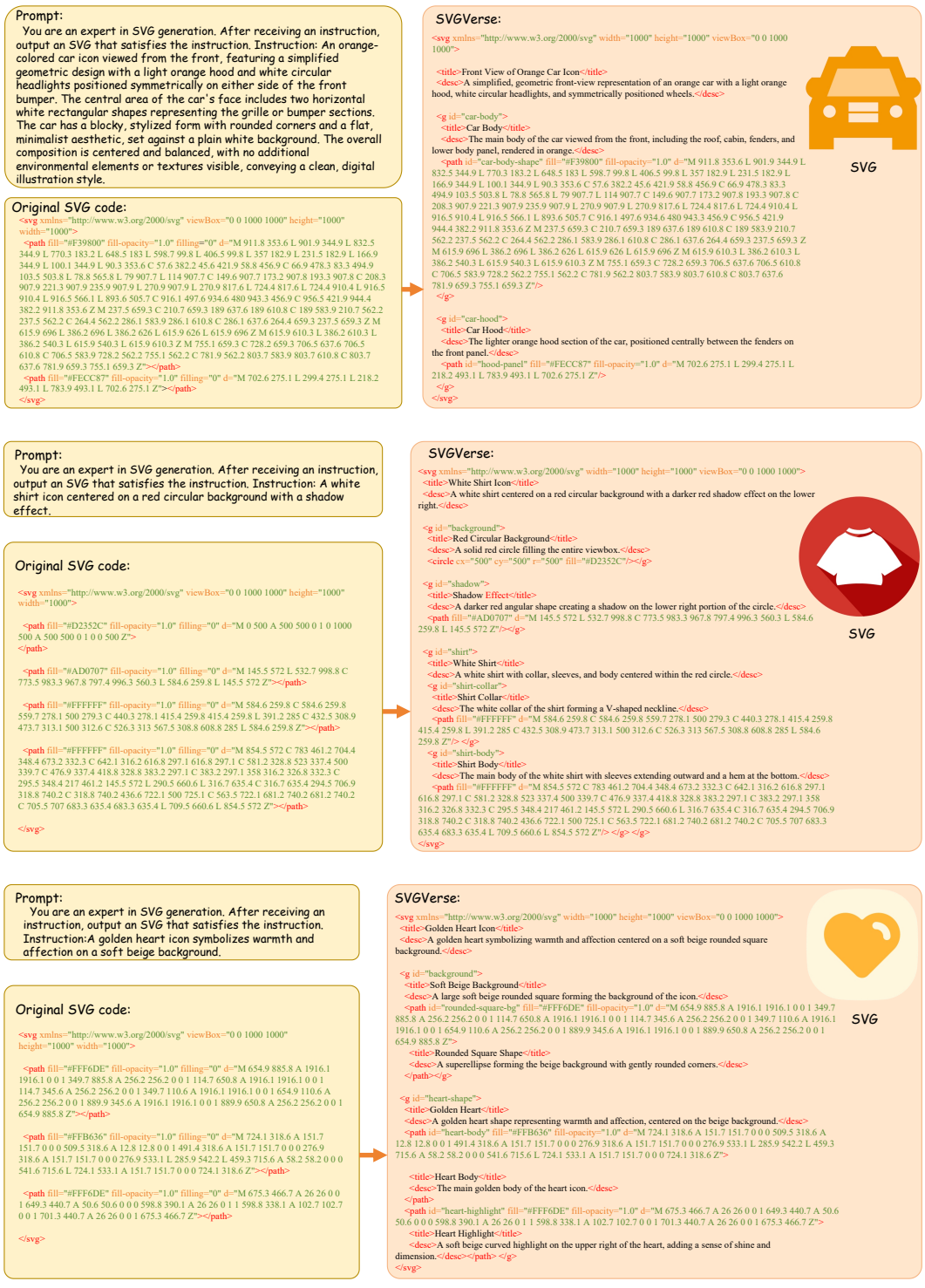

Figure 4: Additional examples of semantic reconstruction from original SVG code to SVGVerse.

System Prompt:

You are an SVG semantic enhancement expert. Rewrite the given SVG into a semantically richer, well-structured version preserving the original visual output.

User Prompt:

1. Output ONLY SVG source code. No markdown, no explanations, no code fences.
2. Root: xmlns="http://www.w3.org/2000/svg" width="1000" height="1000" viewBox="0 0 1000 1000"
3. Preserve original visual: main objects, layout, colors, geometry, composition, text content, rendering result. Do NOT add objects not in the original.
4. If original viewBox differs, scale to fit 1000x1000 preserving appearance.
5. Top level must include <title> (image theme) and <desc> (1-2 sentences on objects/spatial relations).
6. Group main objects with <g id="semantic-name"> (e.g. id="main-building", NOT id="shape1"). Each group should have <title> and <desc>.
7. Build hierarchy: background / ground / main-objects / secondary-objects / labels via <g id>.
8. Reuse original geometry. Add semantic grouping, <title>/<desc>, meaningful ids.
9. Prefer basic elements: <rect> <circle> <ellipse> <line> <polyline> <polygon> <text> <g> <title> <desc>
10. Avoid <path> unless in original and necessary. Keep needed <path> with semantic id/title/desc. No <path> for simple shapes.
11. Don't add objects/text not in original. Preserve original text. Priority: preserve rendering > enhance semantics > improve readability.
12. Absorb caption info into title/desc/id where helpful but don't contradict the image.
13. Keep colors, coordinates close to original. No JS/CSS/external resources. Valid XML/SVG.

Figure 5: Instruction prompt used for semantic SVG reconstruction during the construction of SVG-Verse.

Table 9: SVG-specific tokens used in DEFT.

(a) Tag tokens

Category	Tokens
Root	`<svg, </svg>, <defs, </defs>, <use, </use>, />`
Grouping	`<g, </g>`
Shapes	`<path, </path>, <rect, </rect>, <circle, </circle>, <ellipse, </ellipse>, <line, </line>, <polyline, </polyline>, <polygon, </polygon>`
Text	`<text, </text>, <tspan, </tspan>, <textPath, </textPath>`
Gradients	`<linearGradient, </linearGradient>, <radialGradient, </radialGradient>, <stop, </stop>`
Clipping	`<clipPath, </clipPath>, <mask, </mask>`
Filters	`<filter, </filter>, <feGaussianBlur, </feGaussianBlur>, <feColorMatrix, </feColorMatrix>, <feComposite, </feComposite>, <feBlend, </feBlend>`

(b) Attribute tokens

Category	Tokens
Geometry	`width=", height=", viewBox=", x=", y=", x1=", y1=", x2=", y2=", cx=", cy=", r=", rx=", ry=", d=", points="`
Styling	`fill=", stroke=", stroke-width=", stroke-linecap=", stroke-linejoin=", stroke-miterlimit=", fill-rule=", opacity="`
Transform	`transform="`
Text	`font-size=", font-family=", text-anchor="`
Gradients	`gradientUnits=", gradientTransform=", offset=", stop-color="`
Identifiers	`id=", class=", clip-path="`

<title>, </title>, <desc>, and </desc>. Distillation loss on these tokens is provided exclusively by the Planning Expert, and they account for approximately 27.0% of the supervised tokens during training. The drawing command set $\mathcal{C}_{\text{draw}}$ contains tokens belonging to SVG elements that specify geometric and visual realization, such as <path>, <rect>, and <circle>, together with other drawing elements and their associated attributes. These tokens receive supervision exclusively from the Drawing Expert and constitute approximately 68.8% of the training tokens. The remaining 4.2% are left unsupervised during distillation. They mainly include global structural commands such as <svg> and </svg>, as well as whitespace, line breaks, indentation, and text located between element spans. Excluding these tokens prevents either expert from imposing supervision on shared structural regions that do not correspond specifically to semantic planning or geometric realization. Importantly, routing is determined by the semantic role of each command rather than its hierarchical position in the SVG structure. Thus, drawing elements nested within a <g> group are still supervised by the Drawing Expert, while the group-related commands themselves are supervised by the Planning Expert.

D Additional Experiments and Analysis

Effect of Semantic Reconstruction. To isolate the effect of semantic reconstruction from additional supervised training, we compare second-stage SFT on the original SVG collection and SVGVerse under the same Stage 1 initialization. As shown in Table 10, further training on the original SVG collection provides only limited gains, with several drawing metrics even degrading relative to Stage 1 SFT. One possible explanation is the distribution mismatch between the Stage 1 training data and the original SVG collection, which may partially disrupt the representation acquired during the first stage. In contrast, as shown in Figure 6, training on SVGVerse not only leads to a faster decrease in SFT loss but also yields better final performance. We attribute this improvement to the explicit semantic organization introduced by SVGVerse. Although the reconstructed SVGs are longer and contain more complex command structures than those in the original collection, their richer semantic representations organize visually coherent elements into meaningful components and establish clearer correspondences between semantic composition and drawing commands. Such structured supervision provides a more learnable representation of SVG generation by making the correspondence between what to draw and how to draw more explicit.

Extended Evaluation on SArena-Icon. We further provide a more comprehensive comparison on SArena-Icon in Table 12. In addition to the metrics reported in the main paper, the extended results include a broader set of baseline models and the SVG rendering success rate (SR). DEFT maintains strong visual and semantic performance while achieving a high rendering success rate, demonstrating that its improvements are not obtained at the expense of SVG executability.

Component-Level Semantic Consistency. To assess whether the semantic descriptions of generated components correspond to their geometric content, we conduct a component-level analysis on the SVGVerse test set. The reference SVGs contain 2,312 semantic components in total. Across inference runs, DEFT generates an average total of 2,292 components on this test set, compared with 2,063 for the variant without Planning OPD.

We randomly sample 200 generated components from each model and render each component within a fixed <svg> root with the SVG namespace, a canvas size of 1000×1000, and viewBox="0 0 1000 1000". We then manually compare each rendered component with its associated textual description to assess semantic consistency. DEFT achieves a consistency rate of 98.5%, compared with 91.0% for the variant without Planning OPD, an improvement of 7.5 percentage points. The main observed inconsistencies involve missing or extraneous visual elements and incorrect colors.

These results provide direct evidence that the semantic descriptions of components generated by DEFT generally correspond to their rendered geometric content. The improvement over the variant without Planning OPD further supports the contribution of planning distillation to this correspondence. DEFT also generates an aggregate component count closer to that of the reference SVGs, although aggregate counts alone do not establish component-level coverage or correctness.

SVG Editing with Semantic Structure. To assess the utility of semantic SVG organization for downstream editing, we evaluate Qwen3-VL-8B and DEFT on editing tasks using the original SVG code and the semantically reconstructed SVGVerse code from our test set. To construct the editing tasks, we provide GPT-5.6-Thinking with the original SVG code and its rendered image and ask it to generate an editing instruction together with the corresponding edited SVG code. The edited code is then rendered to obtain the target reference image. The resulting editing instructions and target reference images undergo manual review. For each test instance, both input representations are evaluated using the same editing instruction and target reference image. Table 11 reports the results under both settings.

Both models achieve better editing results with SVGVerse inputs across all four visual-quality metrics and SR. For Qwen3-VL-8B, semantic reconstruction increases DINO from 0.629 to 0.798 and SR from 74.61% to 79.69%. DEFT also benefits from the reconstructed representation, with SSIM increasing from 0.690 to 0.761, PSNR from 10.803 to 12.472, and SR from 96.68% to 99.41%. These consistent gains suggest that the explicit component organization and semantic annotations in SVGVerse provide useful structure for SVG editing.

DEFT further outperforms Qwen3-VL-8B under both input representations, indicating that its editing advantages extend to original SVG code without semantic reconstruction. Although SVGVerse inputs lead to longer outputs for both models, DEFT uses fewer output tokens than Qwen3-VL-8B in each setting. Overall, these results support the practical utility of SVGVerse for downstream editing, while evaluating the reconstructed representation as a whole rather than isolating the effects of grouping, semantic annotations, or planning distillation.

Qualitative Analysis. Figure 7 provides qualitative comparisons between the base Qwen3-VL-8B-Instruct model (Bai et al., 2025) and DEFT. The base model can generate executable SVG code, but its outputs are primarily organized as low-level drawing commands with limited correspondence to the semantic composition of the depicted objects. In contrast, DEFT produces more visually faithful vector graphics while organizing the generated code into explicit semantic components through structured `<g>`, `<title>`, and `<desc>` elements. For example, the generated car is decomposed into meaningful components such as the car body and headlights, while the checklist example explicitly separates the card and individual checklist rows. These examples illustrate that DEFT improves not only geometric realization but also the semantic organization of generated SVG code.

Table 10: Ablation study on semantic SVG reconstruction in the second-stage SFT.

Model	DINO ↑	SSIM ↑	PSNR ↑	MSE ↓	F1 ↑	LCS ↑
Qwen3-VL-8B	0.834	0.646	8.805	0.175	–	–
+ Stage 1 SFT	0.844	0.650	8.945	0.173	–	–
+ Original SVG SFT	0.852	0.638	8.815	0.176	–	–
+ SVGVerse SFT	**0.859**	**0.653**	**9.886**	**0.168**	**0.598**	**0.780**

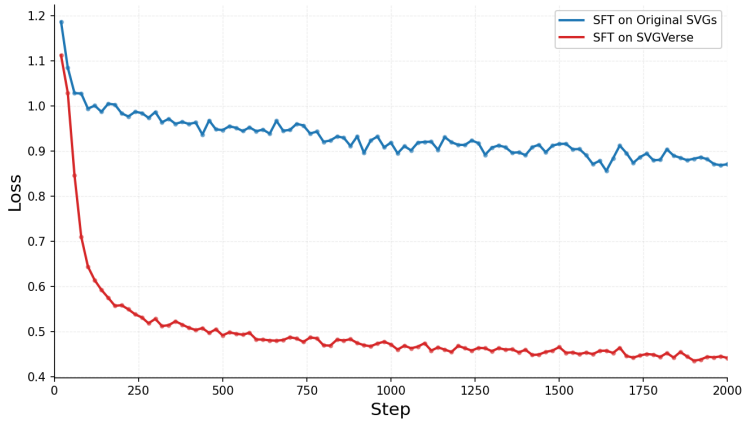

Figure 6: Training loss of the second-stage SFT on the original SVG collection and SVGVerse.

Table 11: SVG editing results using original and semantically reconstructed SVGVerse inputs from our test set.

Model	Input SVG	DINO↑	SSIM↑	PSNR↑	MSE↓	SR↑	Tokens↓
Qwen3-VL-8B	Original	0.629	0.508	8.126	0.178	74.61%	944
	SVGVerse	0.798	0.565	9.781	0.164	79.69%	1.06K
	Δ *Improvement*	+0.169	+0.057	+1.655	−0.014	+5.08 pp	
DEFT	Original	0.852	0.690	10.803	0.141	96.68%	879
	SVGVerse	**0.936**	**0.761**	**12.472**	**0.129**	**99.41%**	936
	Δ *Improvement*	+0.084	+0.071	+1.669	−0.012	+2.73 pp	

Table 12: Extended results on SArena-Icon. SR denotes the SVG rendering success rate.

Model	FID ↓	T2I ↑	I2I ↑	SR ↑	Tokens
Traditional SVG Methods					
IconShop	32.288	20.894	70.922	86.51%	1.6k
VectorFusion	16.594	22.992	70.308	**100.0%**	33k
SVGDreamer	26.612	20.329	69.975	**100.0%**	132k
Large Language Models					
Qwen3-VL-8B	16.875	22.578	73.187	99.55%	374
InternVL3-8B	23.061	21.897	71.450	93.96%	269
Gemma-3-27B	15.145	22.526	73.277	99.37%	249
Llama-4-Scout	17.908	22.849	73.563	96.03%	256
Llama-4-Maverick	14.931	23.570	75.816	98.12%	265
Qwen2.5-VL-72B	15.948	22.946	73.681	98.50%	275
Qwen3-VL-235B-A22B	18.404	24.024	76.289	94.96%	326
InternVL3.5-241B	11.265	22.534	76.807	96.99%	991
GPT-4o	15.178	24.617	77.742	99.47%	246
Gemini-2.5-Flash	16.720	24.658	78.218	97.34%	451
Claude-Sonnet-4	15.840	25.421	80.579	99.57%	444
GPT-5.2	14.965	25.457	<u>82.701</u>	97.02%	623
Claude-Sonnet-4.5	16.451	25.322	81.038	99.63%	320
SVG-LLMs					
LLM4SVG-7B	21.939	19.458	70.726	90.95%	705
OmniSVG-3B	28.292	21.679	74.831	99.68%	1.7k
InternSVG-8B	**8.715**	23.916	80.911	97.24%	1.0k
CTRL-S-8B	11.584	**25.944**	82.291	<u>99.85%</u>	346
IntroSVG-7B	22.145	23.865	77.828	91.80%	657
DEFT-8B	<u>10.723</u>	<u>25.800</u>	**83.046**	99.75%	674

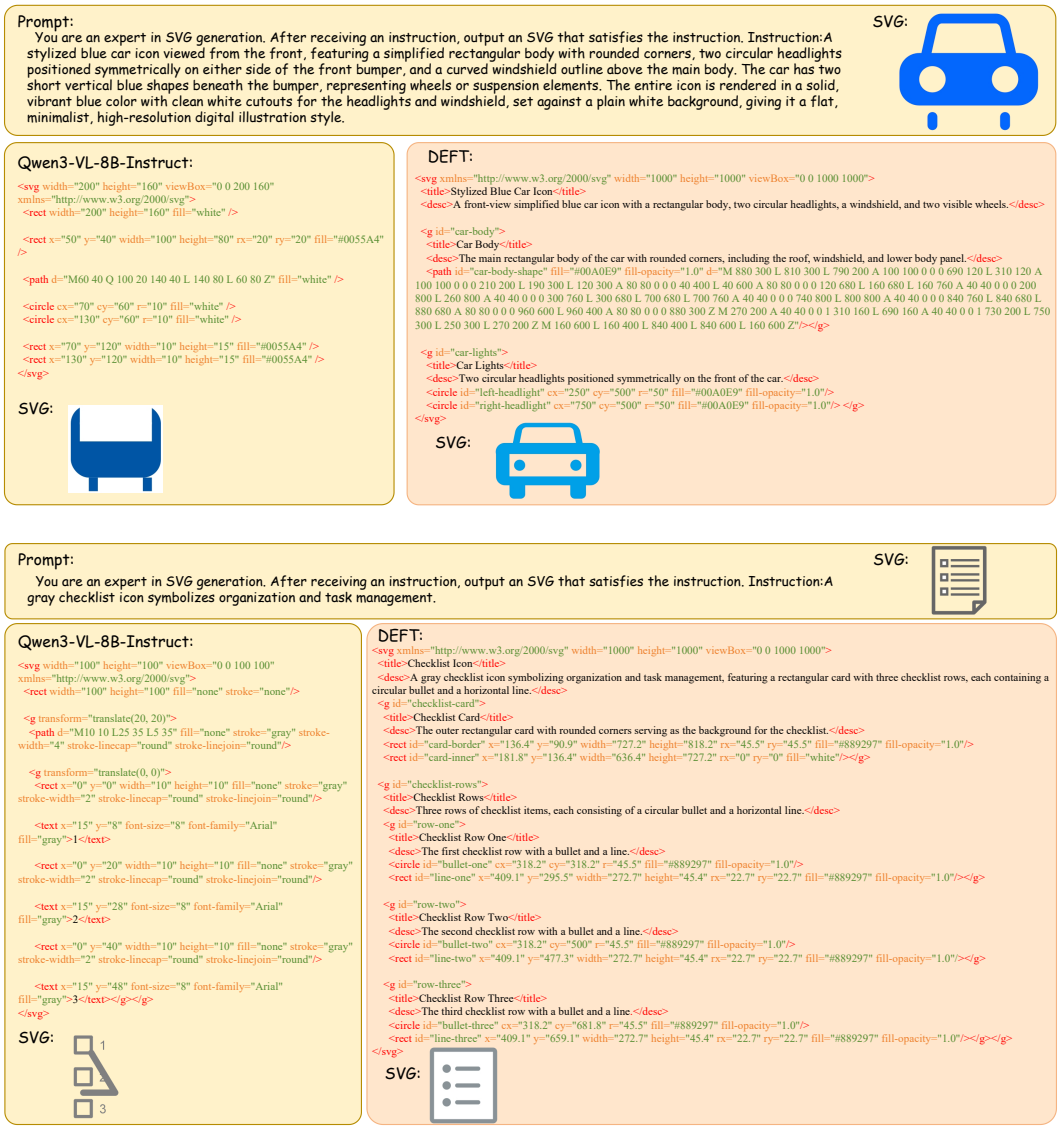

Figure 7: Qualitative comparison between Qwen3-VL-8B-Instruct and DEFT. DEFT generates vector graphics with improved visual fidelity while organizing SVG code into explicit semantic components.