

Supplementary materials of the OmniSVG

A Additional Details of MMSVG-2M dataset

A.1 Samples of MMSVG-2M Dataset

We visualize samples of our MMSVG-2M dataset in Fig. 9. In our MMSVG-2M dataset, 55% of the SVG samples belongs to the MMSVG-Icon, 25% belongs to the MMSVG-Illustration, and the rest 20% belongs to the MMSVG-Character. Among the SVG samples within the MMSVG-Character category, half of them comes from Freepik, while another half is generated by our data creation pipeline. We also collect image-SVG pairs for the character-reference SVG generation tasks during the generation process.

A.2 SVG-Image-Text Pairs Construction

Our *MMSVG-2M* dataset comprises two million SVG samples with the corresponding rasterized images. We generate captions on the rasterized images with BLIP-2 [24], thereby providing textual descriptions that enable us to fine-tune our model to follow these instructions. We use CairoSVG [21] for rasterization and remove samples that produced completely white images.

Annotation. We employ an off-the-shelf VLM, specifically BLIP-2 [24], to generate SVG captions with the prompt below. To reduce hallucinations, we drop the samples with CLIP scores less than 30. We also visualize the distribution annotated keywords of MMSVG-2M dataset in Fig. 11 with word cloud format. And the instruction template for annotation is shown in Tab. 7.

Instruction templates. MMSVGBench provides three tasks, including text-to-SVG task, image-to-SVG task and character-reference SVG generation task. Each task needs different instruction templates. For the text and image conditioning SVG generation, we provide the input text or image with VLM architecture. For character-reference SVG generation, we provide the natural character reference image and the original image with the VLM architecture. The list of instruction templates for different tasks are shown in Tab. 7.

- **Employed BLIP2 for SVG Captioning:** You are a helpful assistant. Your task is to describe this image in a single sentence, including the object, its color, and its overall arrangement. For example: "Yellow cheers with glasses of alcohol drinks." / "Heart emojis represent love on Valentine's Day."
- **Text-to-SVG:** You are a helpful SVG Generation assistant, designed to generate SVG. We provide the text description as input, generate SVG based on the text.
- **Image-to-SVG:** You are a helpful SVG Generation assistant, designed to generate SVG. We provide an image as input, generate SVG for this image.
- **Character-Reference SVG Generation:** You are a helpful SVG Generation assistant, designed to generate SVG. We provide two image as input, the second image is the character reference image of the first image, generate the character reference SVG based on these two input images.

Table 7: The list of instructions for different tasks, including annotation, text-to-SVG, image-to-SVG and character-reference SVG generation.

A.3 Character-SVG Pairs Construction

As illustrated in the Fig. 1, part of our proposed MMSVG-2M-Character subset is constructed using a generative pipeline. As shown in the pipeline diagram in Fig. 2, we employ a FLUX [22]-based generative model enhanced with a vector-style LoRA to enable the generation of SVG-style data. For image-based conditioning, we adopt FLUX-Redux [23], which injects image features via a SigLIP encoder and projects them into image embeddings. These embeddings are then concatenated with the text tokens as conditioning inputs for FLUX [22]. However, in practice, the original Redux [23] conditioning proves to be overly strong. To address this, we adopt a community-implemented variant

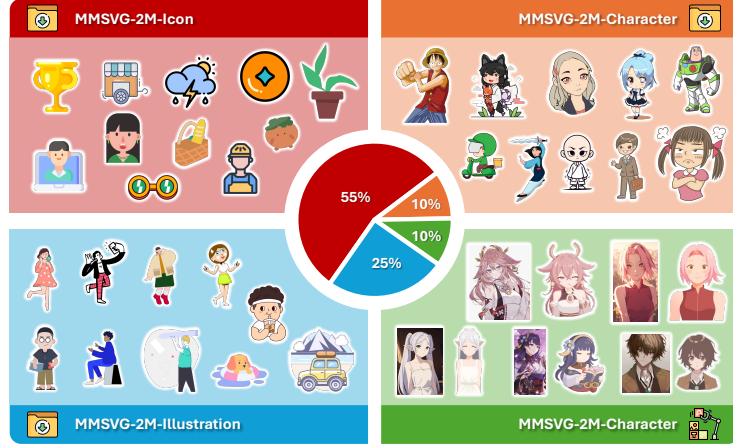


Figure 9: **Samples from MMSVG-2M dataset.** The proposed MMSVG-2M dataset can be separated into three subset, namely Icon, Illustration and Character. Samples from Icon, Illustration and part of Character subsets are downloaded from Internet. Another part of Character subset is generated by our data creation pipeline, which can provide image and SVG pairs for image prompting task.

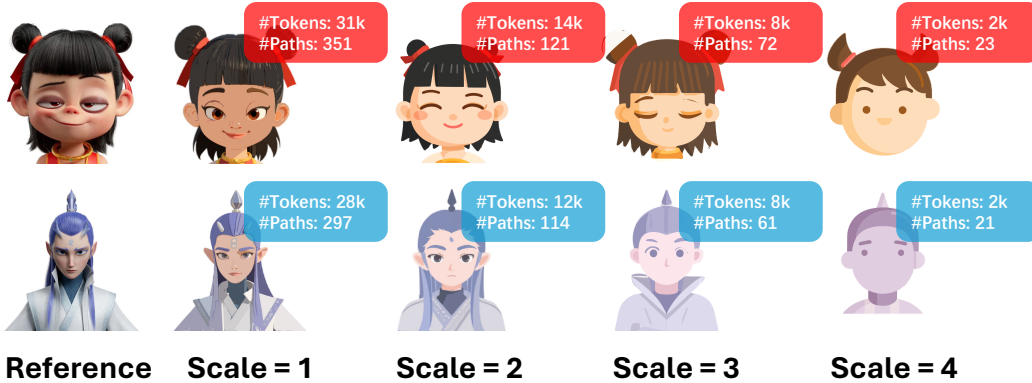


Figure 10: **Image prompting dataset creation of MMSVG-2M Character.** By utilizing FLUX-Redux and SVG vectorization tools, image prompting data pairs can be generated. We adopt FLUX-Redux downsampling scale with 2, 3 in practice by trading-off the character similarity and complexity of generated SVG.

of Redux that downsamples the image embeddings in 2D space. As observed in our experiments shown in Fig. 10, a downsampling factor between $2\times$ and $3\times$ yields the most reasonable SVG-style character references. Finally, we employ VTracer [10] to perform near-instant vectorization of the generated images. To construct the MMSVG-2M-Character subset, we first filter 103k character instances from the Danbooru [11] dataset and apply the aforementioned pipeline. We compare the raw FLUX [22] outputs and their vectorized counterparts, retaining only those samples with PSNR and SSIM scores above a certain threshold as valid data.

B Additional Details

B.1 Scaling Up

To study the effectiveness of scaling up multimodal SVG generation, we scale up OmniSVG from 4B to 8B parameters. We present training perplexity in Fig. 12, where both models are trained from scratch on 250 billion tokens. We show that, as the size of the model grows, the model achieves a lower validation perplexity, indicating a higher probability of producing the validation data.



Figure 11: **Word cloud visualization of label distribution in the MMSVG-2M dataset.** The size of each label corresponds to its frequency of occurrence. The larger the label, the more frequently it appears in the dataset.

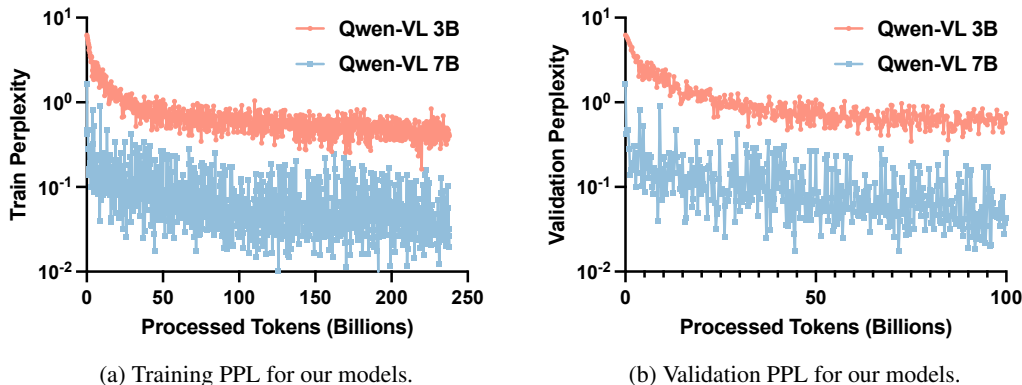


Figure 12: **Training and Validation Perplexity (PPL) for OmniSVG Models.** We train all the models from scratch on 250 billion tokens. We observe that the performance grows with model sizes.

824 B.2 Implementation Details

We train our models in bfloat16 with the ZeRO-2 strategy [35] for memory-efficient training. We also adopt the AdamW [29] optimizer with a learning rate decaying from 3×10^{-4} to 3×10^{-6} and a weight decay of 0.1 to train our model. In practice, we load the pre-trained weights from the Qwen2.5-VL [1] model and initialize the SVG embeddings from scratch. Without further specification, we generate SVGs with the top-k and top-p sampling strategy with $k = 50$ and $p = 0.95$ for diversity.

C Additional Results

We provide full comparisons in Tab. 8, including all the baselines mentioned in Sec. 5. For the text-to-SVG task, we compare our method with language-based (LLM-based) methods, including VectorFusion [19], SVGDreamer [55], Chat2SVG [51] and IconShop [52]. For image-to-SVG task, we compare our method with baseline methods across image vectorization and Multimodal Large Language Modeling approaches, including LIVE [30], DiffVG [25], StarVector [37] and GPT-4o [18] using the official implementations with the hyperparameters proposed by the authors, and apply their pre- and post-processing code as required. Specifically, for the text-to-SVG task, the optimization-based method SVGDreamer excels in enhancing editability by employing a semantic-driven image vectorization process that effectively separates foreground objects from the background, while failing to handle complex scenes. Another optimization-based work, VectorFusion, stands out for generating SVG-exportable vector graphics without relying on large captioned datasets. However, Vectorfusion is also unable to handle complex scenarios and diverse styles. The significant problem with these optimization-based works is that the optimization time is too long. Generating an SVG usually takes more than ten minutes, which is too expensive. For the LLM-based method, Chat2SVG integrates Large Language Models (LLMs) with image diffusion models to create semantically rich SVG templates. However, Chat2SVG still needs to optimize the output SVG script from LLM, which introduces increased computational complexity and poses challenges during model training. In comparison, IconShop utilizes a transformer-based architecture to autoregressively model SVG path sequences, demonstrating exceptional performance in simplified icon SVGs, which offers effective solutions for text-to-SVG generation. It can only generate black simple Icon SVGs.

For the image-to-SVG task, we compare our method with the image vectorization methods. LIVE allows progressive and efficient generation of SVGs, optimizing closed vector paths under raster image supervision with shape complexity control. However, LIVE needs to optimize for a long time when generating complex SVGs. DiffVG enables end-to-end differentiability in vector graphics rasterization, improving optimization through anti-aliasing and gradient-based methods while also is computationally expensive due to the complexity of the forward-backward rasterization process. Recently, the Multimodal Large Language Model (MLLM) based method StarVector leverages the visual understanding to apply accurate SVG primitive to the LLM architecture, which also can generate SVGs from both text and image inputs. However, it still fails to generate complex SVGs. Since Starvector [37] has not yet opened up its text-to-SVG model weights, our MMSVGBench does not evaluate Starvector’s text-to-SVG capabilities. MMSVG-Bench also evaluates our methods with VLM methods, GPT-4o, to conduct a comprehensive assessment. We compare our method with these baselines on our MMSVG-2M dataset, from simple MMSVG-Icon dataset, a bit complex MMSVG-illustration dataset, to the very complex MMSVG-Character dataset.

D More details of the baselines

D.1 Text-to-SVG Task

SVGDreamer [55] uses a semantic-driven image vectorization (SIVE) process to separate foreground objects and background, improving editability. The SIVE process utilizes attention-based primitive control and an attention-mask loss function to manipulate individual elements effectively. To address issues in existing text-to-SVG generation methods, the proposed Vectorized Particle-based Score Distillation (VPSD) approach models SVGs as distributions of control points and colors, improving shape, color diversity, and convergence speed.

VectorFusion [19] leverages a text-conditioned diffusion model trained on pixel representations to generate SVG exportable vector graphics without needing large captioned SVG datasets. By optimizing a differentiable vector graphics rasterizer, it distills semantic knowledge from a pretrained diffusion model and uses Score Distillation Sampling to generate an SVG consistent with a caption. Experiments show that VectorFusion improves both quality and fidelity, offering a variety of styles such as pixel art and sketches.

Chat2SVG [51] proposes a hybrid framework that combines the strengths of Large Language Models (LLMs) and image diffusion models for text-to-SVG generation. The approach first uses an LLM to create semantically meaningful SVG templates from basic geometric primitives. A dual-stage

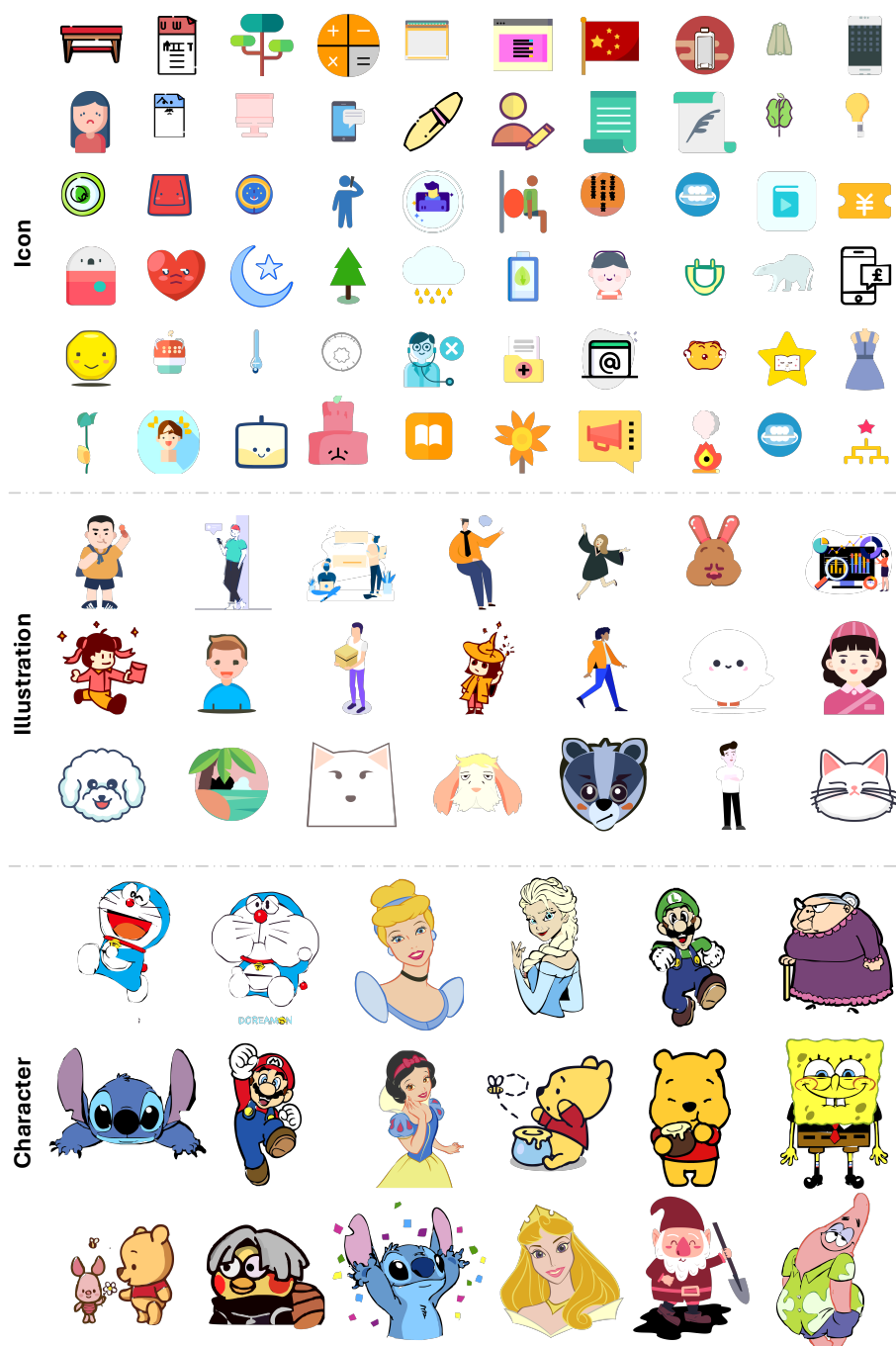


Figure 13: Illustration of the SVG generation capabilities of OmniSVG.

Table 8: **Quantitative Evaluations.** Quantitative results between OmniSVG and current state-of-the-art text-to-SVG and image-to-SVG baseline methods. The bold numbers and underlined numbers represents the best and second best performance respectively. Our OmniSVG model demonstrates superior performance compared SOTA SVG generation baselines. StarVector(8B) fails to generate complex SVGs when evaluated in the complex MMSVG-Character dataset.

Evaluation Dataset	Methods	Time	# Tokens	Text-to-SVG				Image-to-SVG			
				FID↓	CLIP↑	Aesthetic↑	HPS↑	DINO↑	SSIM↑	LPIPS↓	MSE↓
MMSVG-Icon	Vectorfusion [19]	69s	40k	88.74	0.2798	4.98	0.225	–	–	–	–
	SVGDreamer [55]	260s	100k	75.31	0.2923	5.32	0.256	–	–	–	–
	Chat2SVG [51]	28s	1.7k	70.39	0.3029	5.38	0.256	–	–	–	–
	IconShop [52]	6.8s	3.3k	80.84	0.2359	3.47	0.138	–	–	–	–
	LIVE [30]	170s	18.2k	–	–	–	–	0.960	0.979	0.034	0.004
	DiffVG [25]	21s	19.8k	–	–	–	–	0.951	0.956	0.056	0.015
	GPT-4o [18]	6.9s	0.6k	–	–	–	–	0.887	0.826	0.179	0.134
	StarVector(8B) [37]	45s	3.5k	–	–	–	–	0.977	0.953	0.048	0.012
	OmniSVG(3B)	15s	3.8k	<u>64.83</u>	<u>0.3194</u>	<u>5.49</u>	<u>0.247</u>	<u>0.980</u>	0.954	0.049	0.011
	OmniSVG(7B)	23s	4.6k	62.16	0.3278	5.63	0.258	0.984	<u>0.973</u>	<u>0.037</u>	<u>0.008</u>
MMSVG-Illustration	Vectorfusion [19]	70s	41k	90.03	0.2639	4.67	0.214	–	–	–	–
	SVGDreamer [55]	260s	102k	83.02	0.2795	5.09	0.219	–	–	–	–
	Chat2SVG [51]	35s	2.1k	78.84	0.2891	5.01	0.221	–	–	–	–
	IconShop [52]	7.8s	3.3k	84.04	0.2198	3.21	0.098	–	–	–	–
	LIVE [30]	250s	18.2k	–	–	–	–	0.959	0.960	<u>0.044</u>	0.011
	DiffVG [25]	31s	19.8k	–	–	–	–	0.929	0.930	0.077	0.021
	GPT-4o [18]	8.0s	0.7k	–	–	–	–	0.801	0.783	0.244	0.199
	StarVector(8B) [37]	65s	5.3k	–	–	–	–	0.807	0.831	0.248	0.181
	OmniSVG(3B)	49s	9.7k	<u>70.45</u>	<u>0.3077</u>	<u>5.39</u>	<u>0.245</u>	<u>0.974</u>	0.944	0.069	0.019
	OmniSVG(7B)	63s	10.9k	66.91	0.3164	5.59	0.253	0.988	<u>0.959</u>	0.041	<u>0.013</u>
MMSVG-Character	Vectorfusion [19]	69s	40k	93.03	0.2218	4.08	0.198	–	–	–	–
	SVGDreamer [55]	260s	100k	86.01	0.2423	4.81	0.205	–	–	–	–
	Chat2SVG [51]	42s	1.7k	82.98	0.2499	4.83	0.209	–	–	–	–
	IconShop [52]	6.8s	3.3k	88.41	0.2001	3.13	0.077	–	–	–	–
	LIVE [30]	190s	18.2k	–	–	–	–	0.911	<u>0.939</u>	0.109	0.038
	DiffVG [25]	21s	19.8k	–	–	–	–	0.876	0.915	0.163	0.051
	GPT-4o [18]	8.1s	0.7k	–	–	–	–	0.712	0.693	0.365	0.349
	StarVector(8B) [37]	–	–	–	–	–	–	–	–	–	–
	OmniSVG(3B)	105s	30.8k	<u>72.18</u>	<u>0.2898</u>	<u>5.19</u>	<u>0.213</u>	<u>0.921</u>	0.917	<u>0.049</u>	<u>0.021</u>
	OmniSVG(7B)	139s	35k	68.02	0.3088	5.33	0.238	0.932	0.941	0.072	0.008

optimization pipeline, guided by image diffusion models, refines paths in latent space and adjusts point coordinates to enhance geometric complexity.

IconShop [52] uses a transformer-based architecture to encode path commands and learn to model SVG path sequences autoregressively. It has shown excellent results in simplified icon scenarios and provides a good solution to Text-to-SVG generation by extending the FIGR-8-SVG dataset with captions. We have access to their dataset and original splits and have trained our model on that data using a pre-trained checkpoint (trained on OmniVG dataset). We have extracted the results from IconShop and included them here to compare our method.

LLM4SVG [54] is a framework that leverages Large Language Models (LLMs) to understand and generate Scalable Vector Graphics (SVGs). It employs a structured SVG encoding approach, utilizing learnable semantic tokens to accurately represent SVG components and their properties. This design enables LLMs to produce SVGs that are both semantically aligned with textual descriptions and visually coherent. However, LLM4SVG also has a maximum token length of 2048, limiting its ability to generate highly complex SVGs that require longer sequences.

D.2 Image-to-SVG Task

LIVE (Layer-wise Image Vectorization) [30] is a method for progressively generating SVGs that closely fit a given raster image by recursively adding and optimizing closed vector paths. Using a differentiable renderer (based on DiffVG [25]), LIVE enables direct optimization of paths under raster image supervision while controlling shape complexity by adjusting the number of path segments. It

901 introduces component-wise path initialization, identifying key visual components to ensure efficient
902 topology extraction and minimize redundant shapes.

903 **DiffVG** [25] is a landmark in vector graphics research, pioneering deep learning-based methods with
904 the first differentiable vector graphics rasterization pipeline. By leveraging a combination of anti-
905 aliasing techniques and gradient-based optimization, DiffVG ensures differentiability. Unlike methods
906 relying on non-differentiable curve-to-mesh conversions, DiffVG employs a forward-backward
907 rasterization process, where the forward pass generates antialiased images and the backward pass
908 computes gradients with respect to vector graphic parameters.

909 **StarVector** [37] works directly in the SVG code space, leveraging visual understanding to apply
910 accurate SVG primitives. StarVector employs a transformer-based architecture that integrates an
911 image encoder with a language model, enabling it to process visual inputs and produce precise
912 SVG code. StarVector effectively handles diverse SVG types, including icons, logos, and complex
913 diagrams, demonstrating robust generalization across various vectorization tasks. However, with a
914 16k token context window, StarVector may struggle to process highly complex SVGs that require
915 longer sequences.