# OmniSVG: A Unified Scalable Vector Graphics Generation Model

**Yiying Yang**[1,2]* **Wei Cheng**[2]* **Sijin Chen**[1] **Xianfang Zeng**[2] **Fukun Yin**[1,2]
**Jiaxu Zhang**[2] **Liao Wang**[2] **Gang Yu**[2‡] **Xingjun Ma**[1‡] **Yu-Gang Jiang**[1]
[1] Fudan University [2] StepFun

Project Page MMSVG-2M MMSVGBench Models Code

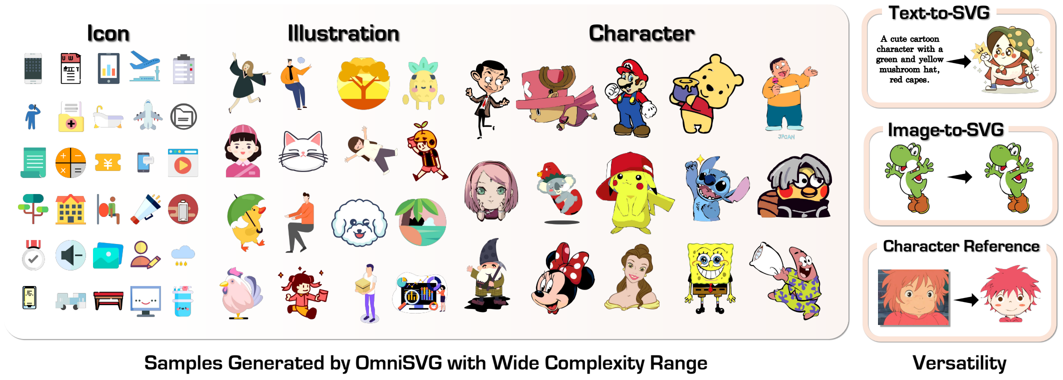

Icon
Illustration
Character
Text-to-SVG
A cute cartoon character with a green and yellow mushroom hat, red capes.
Image-to-SVG
Character Reference
Samples Generated by OmniSVG with Wide Complexity Range
Versatility

Figure 1: **OmniSVG** is capable of autoregressively generating high-quality Scalable Vector Graphs (SVG) across a wide spectrum of complexity, from simple icons to intricate anime characters. OmniSVG demonstrates remarkable versatility in generating high-quality SVGs adhering to multimodal instructions, covering tasks like Text-to-SVG, Image-to-SVG, and Character-Reference SVG, making it a powerful and flexible solution for diverse creative tasks.

## Abstract

Scalable Vector Graphics (SVG) is an important image format widely adopted in graphic design because of their resolution independence and editability. The development of autonomous SVG generation workflows is continuously drawing attention from both designers and researchers in the AIGC community. However, existing methods either produce unstructured outputs at huge computational cost or are limited to generating monochrome icons of over-simplified structures. To produce high-quality and complex SVG adhering to multi-modal instructions, we propose **OmniSVG**, a unified SVG generation framework that inherits knowledge from a pre-trained Vision-Language Model (VLM). By parameterizing SVG commands and coordinates into discrete token sequences, the auto-regressive nature enables us to seaminglessly adapt modern VLMs to the direct SVG generation. To further advance the development of SVG synthesis, we introduce **MMSVG-2M**, a multimodal dataset with two million richly annotated SVG assets, along with a standardized evaluation protocol for conditional SVG generation tasks. Extensive experiments show that OmniSVG outperforms existing methods and demonstrates its potential for integration into professional SVG design workflows.

* Yiying Yang and Wei Cheng contributed equally to this work.
‡ Corresponding Authors.

## 1 Introduction

Scalable Vector Graphics (SVG) have become a cornerstone of modern digital design because of their resolution independence, compact file size, and inherent editability. Widely adopted in professional workflows from UI/UX design to industrial CAD systems, SVG enables precise manipulation of geometric primitives (*e.g.*, Bézier curves, polygons) while maintaining high precision and consistent visual quality across varying resolutions. However, creating high-quality SVG content remains challenging for non-experts, requiring mastery of specialized tools or intricate XML syntax.

Existing methods adopt either optimization-based methods or auto-regressive approaches to generate SVG contents.

The optimization-based methods [34, 12, 29] iteratively refine the SVG parameters by minimizing the differences between the input image and the raster image created by differentiable vector graphics rasterizers. Though these methods are sufficient for reconstructing SVG icons, they suffer from significant computational overhead when scaling up to more intricate samples and produce unstructured outputs with redundant anchor points, harming the editability of the reconstructed SVG samples. In contrast, auto-regressive methods build transformer models or adapt pre-trained Large Language Models (LLMs) to directly generate XML parameters [59] or codes [56, 42] representing SVGs. Benefiting from the end-to-end learning pipeline, the auto-regressive method is a more scalable approach [5] as it can learn directly from a large collection of SVG samples. However, existing auto-regressive approaches are limited to basic SVG contents [11, 24, 53] because of the limited context length and the scarcity of complex SVG data.

In this paper, we propose **OmniSVG** that harnesses native VLMs [1] for various end-to-end multimodal SVG generation tasks. By parameterizing SVG coordinates and commands into discrete tokens, OmniSVG decouples structural logic from low-level geometry, mitigating the "coordinate hallucination" problem prevalent in code-based LLMs, and produces vivid and colorful SVG results. Additionally, the next token prediction training objective enables OmniSVG to complete SVGs with diverse generation results given some partial observations. Compared to traditional auto-regressive SVG generation methods, OmniSVG is able to parameterize SVGs exceeding $30k$ tokens, facilitating the generation of detailed and complex SVG contents. Building upon pre-trained VLMs, our method natively integrates the ability to reason upon visual and textual instructions to synthesize editable, high-fidelity SVGs across diverse domains, from icons to intricate illustrations and anime characters.

To advance the development of SVG synthesis, we introduce **MMSVG-2M**, a multi-modal SVG synthesis dataset with two million richly annotated assets, encompassing icons, illustrations, and anime designs.

We also establish a standardized evaluation protocol, **MMSVG-Bench**, for "Text-to-SVG" and "Image-to-SVG" generation. Extensive experiments show that OmniSVG can produce highly detailed and complex SVG contents, surpassing prior art both quantitatively and qualitatively.

To summarize, our key contributions include:

- We introduce OmniSVG, a family of end-to-end multimodal SVG generators that leverage native VLMs for generating complex and detailed SVGs, from simple icons to intricate anime characters.
- We present MMSVG-2M, a large-scale dataset comprising two million SVG assets, along with a standardized evaluation protocol for various multi-modal SVG generation tasks, providing a comprehensive resource for future research.
- Extensive experiments show that OmniSVG surpasses prior SVG generation methods both qualitatively and quantitatively, highlighting its potential for integration into professional SVG design workflows.

## 2 Related Works

**SVG Generation**. Early attempts to generating SVGs directly utilize architectures like RNNs [18, 41, 19, 44, 45], VAEs [4, 32, 48, 46, 51], and Transformers [4, 57] to compress SVG commands into latent representations. Meanwhile, DeepSVG [4] further parameterizes SVGs using a dual transformer architecture but struggles with geometric consistency. Recently, the advent of large language models

(LLMs) [30, 64, 52, 61, 5, 6, 63, 62, 49] unleashes the potential of generating SVGs via XML code synthesis [59, 56, 42]. However, the limited context length of existing LLM-based SVG generation methods [56, 42, 59] poses significant challenges in handling complex SVGs that exceed $10k$ tokens. In this paper, we explore the potential of native Vision-Language Models (VLMs) in multi-modal SVG generation. By combining pre-trained VLMs with SVG command parameterization, we validate that OmniSVG is able to follow multi-modal instructions and generate vivid and complex SVGs.

**Image Vectorization**. Recent advancements in vectorization harness diffusion models paired with differentiable rasterizers, using techniques like score distillation sampling [37, 22, 7] and specialized regularizers [29, 34] to convert raster images into SVG paths. While these methods achieve remarkable results, they face limitations such as over-smoothing, color over-saturation, and lack of editability, often producing tangled paths that fail to capture hierarchical structures inherent in professional SVG designs. In this paper, we present an end-to-end approach that follows multi-modal instructions to generate high-quality SVGs with improved path clarity and editability.

**SVG Datasets and Benchmarks**. The lack of suitable datasets for complex SVG structures presents a significant challenge. Existing datasets [11, 24, 53] primarily focus on simplified path-based SVGs or monochrome icons, overlooking the intricate layered structures and rich color semantics found in real-world designs. For example, FIGR-8-SVG [11] focuses on monochromatic icons, while StarVector [42] proposes categorized datasets, including illustrations, icons, emojis, and fonts. Therefore, existing datasets only present simple SVG samples that do not exceed $8.2k$ tokens, failing to capture the complexities of layered structures and rich color semantics. Benchmark evaluations, such as VGBench [70], further highlight gaps in multi-format testing and the absence of comprehensive coverage for illustrative SVGs. To this end, we introduce **MMSVG-2M**, a multimodal SVG synthesis dataset comprising two million richly annotated assets, including icons, illustrations, and complex anime designs. We also present a standardized evaluation protocol, **MMSVG-Bench**, to evaluate diverse multi-modal SVG generation tasks with varying complexity.

## 3 OmniSVG Dataset

We present **MMSVG-2M**, a large-scale SVG dataset with two million SVG samples covering website icons, illustrations, graphic designs, anime characters, and etc (Sec. 3.1). To promote the downstream development of SVG generation methods, we also introduce **MMSVG-Bench**, a standardized evaluation protocol for a series of multi-modal instruction following tasks for conditional SVG generation (Sec. 3.2).

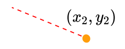
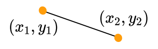
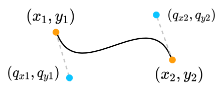
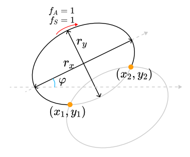
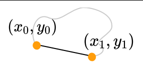
### 3.1 MMSVG-2M

**Data Source.** With increasing visual complexity, MMSVG-2M consists of three subsets, 1) the icon subset MMSVG-Icon collected from Iconfont, 2) the illustration subset MMSVG-Illustration sourced from IconSount, and 3) the complex anime character subset MMSVG-Character both curated from Freepik and created by our data creation pipeline as shown in Fig. 2. All these websites are online platforms where users can publish and share SVGs, encompassing a broad variety of categories. Specifically, our collection of MMSVG-2M contains 1.1 million icons, 0.5 million illustrations, and 0.4 million anime characters as shown in Tab. 6.

**Data Curation.** We extract SVG samples with a comprehensive deduplication process based on filenames, SVG code, and metadata. We first fit the collected SVGs within a viewbox of $200 \times 200$. Then, we employ an off-the-shelf VLM, specifically BLIP-2 [28], to generate captions for the SVGs. Please find more samples from the MMSVG-2M dataset in Fig. 8, and instruction templates in Sec. A.2.

**SVG Simplification** is an essential procedure in SVG data cleansing, since the over-complicated XML grammars in the crawled SVG data will lead to ambiguities while representing basic shapes. To standardize training and evaluation, we simplify all SVG commands with atomic commands as shown in Tab. 1. Inspired by FIGR-8-SVG [11] and IconShop [57], we remove all attributes and simplify each SVG with five basic commands, including “Move To” (M), “Line To” (L), “Cubic Bézier” (C), “Elliptical Arc” (A), “ClosePath” (Z). The introduction of atomic commands further removes the ambiguities, as complex XML grammars can be approximated with the combination of several atomic commands. To efficiently produce a unified and less complex data structure, we utilize

Table 1: **SVG Draw Commands**. Draw commands used in this work along with their arguments and a visualization are listed. The start-position $(x_1, y_1)$ is implicitly defined as the end-position of the preceding command.

| Command | Arguments | Description | Visualization |
|---|---|---|---|
| `<SOP>` | $\varnothing$ | 'Start-of-Path' token. | |
| `M` (MoveTo) | $x_2$, $y_2$ | Move the cursor to the end-point $(x_2, y_2)$ without drawing anything. |  |
| `L` (LineTo) | $x_2$, $y_2$ | Draw a line to the point $(x_2, y_2)$. |  |
| `C` (Cubic Bézier) | $q_{x1}$, $q_{y1}$<br>$q_{x2}$, $q_{y2}$<br>$x_2$, $y_2$ | Draw a cubic Bézier curve with control points $(q_{x1}, q_{y1})$, $(q_{x2}, q_{y2})$ and end-point $(x_2, y_2)$. |  $(x_1, y_1)$<br>$(q_{x2}, q_{y2})$<br>$(q_{x1}, q_{y1})$<br>$(x_2, y_2)$ |
| `A` (Elliptical Arc) | $r_x$, $r_y$<br>$\varphi$, $f_A$, $f_S$<br>$x_2$, $y_2$ | Draw an elliptical arc with radii $r_x$ and $r_y$ (semi-major and semi-minor axes), rotated by angle $\varphi$ to the x-axis, and end-point $(x_2, y_2)$. $(x_2, y_2)$. |  $f_A = 1$<br>$f_S = 1$<br>$r_y$<br>$r_x$<br>$\varphi$<br>$(x_2, y_2)$<br>$(x_1, y_1)$ |
| `Z` (ClosePath) | $\varnothing$ | Close the path by moving the cursor back to the path's starting position $(x_0, y_0)$. |  |
| `F` (Fill) | $fill$ | Draw the fill attribute of the path. | $\varnothing$ |
| `<EOS>` | $\varnothing$ | 'End-of-SVG' token. | |

picosvg to remove grammars like "group" and "transform", and simplify the complex commands to atomic path commands. It is worth noting that atomic path commands are sufficient to represent complex SVGs shown in Fig. 1.

## 3.2 MMSVG-Bench

To compensate for the vacancy of standardized and open evaluation for SVG generation, we introduce **MMSVG-Bench**, a comprehensive benchmark for multi-modal SVG generation. We require the corresponding benchmark to be a sufficient verification whether a model is practically useful in real-world scenarios, and avoid the excessive similarity between the benchmark input data and training data as in traditional train/test splits. Therefore, we opt to generate the benchmark inputs with GPT-4o. Specifically, for Text-to-SVG task, we synthesize 150 textual prompts for each SVG subset (*i.e.* Icon and Illustration). For Image-to-SVG task, we synthesize extra 150 textual descriptions, and prompt GPT-4o to generate vector-style images with transparent backgrounds based on the above texts as the ground truth visual samples. We focus on both the visual quality and semantics of the generation results.

**Text-to-SVG** requires a model to generate SVGs from text instructions. We measure the visual quality with Frechet Inception Distance (FID) [50], aesthetic appeal with Aesthetic score [43], text-SVG alignment with CLIP score [38], and Human Preference Scores (HPS) [58].

**Image-to-SVG** evaluates a model's ability to convert images into SVGs. To quantify the distance between the input and output SVG, we calculate the cosine similarity of DinoV2 features (DinoScore) [35], Structural Similarity Index (SSIM) [54], Learned Perceptual Image Patch Similarity (LPIPS) [66], and Mean Squared Error (MSE).

**Character-Reference SVG Generation** evaluates a model's ability to generate novel SVGs while keeping the profile of the characters depicted in the input image. Different from image-to-SVG, the model does not reconstruct, but generates a specific character SVG for the input image (see

https://github.com/googlefonts/picosvg

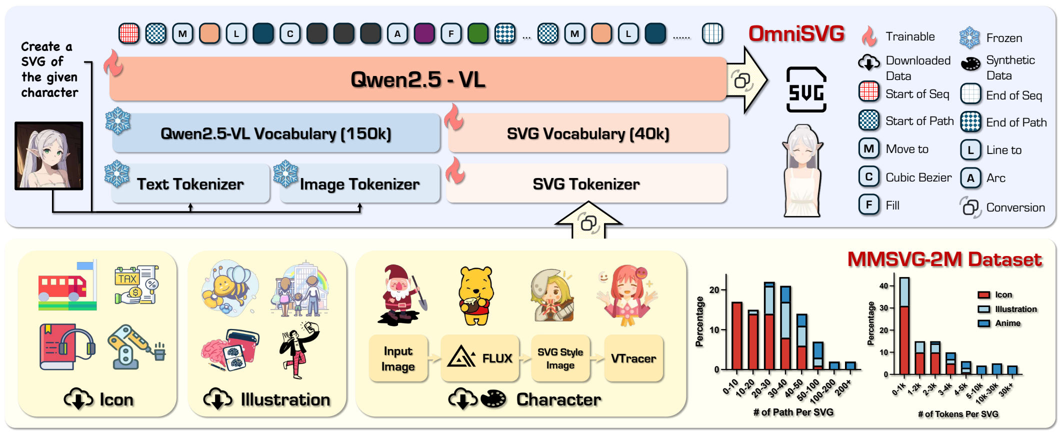


Figure 2: **Overview of OmniSVG**. OmniSVG is built on a pre-trained vision-language model Qwen2.5-VL and incorporates an SVG tokenizer. The model tokenizes both text and image inputs as prefix tokens, while the SVG tokenizer encodes vector graphics commands into a unified representation space.

Fig. 5). We evaluate the alignment between input character images and generated SVGs by prompting GPT-4o [21] to generate a score ranging from 1 to 10, the higher the better. [15, 23, 17]

## 4 OmniSVG

To support end-to-end training for multi-modal SVG generation, OmniSVG parameterizes a series of atomic SVG path commands into a sequence before feeding into a pre-trained VLM with multi-modal instructions.

**SVG Tokenizer.** As illustrated in Sec. 3, our MMSVG-2M dataset simplifies an SVG by removing all attributes and using five basic path commands (see Tab. 1). After the simplification, an SVG script $G$ is represented as the combination of $M$ paths, $G = \{P_i\}_{i=1}^{M}$. Here, $P_i$ is the $i$-th path containing $N_i$ commands, $P_i = \{C_i^j\}_{j=1}^{N_i}$, where $C_i^j$ is the $j$-th command in the $i$-th path. Each command is represented as $C_i^j = (U_i^j, V_i^j)$, containing both the command type identifier $U_i^j \in \{\mathrm{M}, \mathrm{L}, \mathrm{C}, \mathrm{A}, \mathrm{Z}\}$ and the corresponding location argument $V_i^j$.

To generate colored SVG contents, we assign special tokens for hex values to control the "Fill" (F) attribute, distinguishing it from the original SVG commands and coordinates. To this end, we are able to use a total six types of commands $U_i^j \in \{\mathrm{M}, \mathrm{L}, \mathrm{C}, \mathrm{A}, \mathrm{Z}, \mathrm{F}\}$ to parameterize a colored SVG parameterization.

Specifically, our SVG tokenizer transforms SVG scripts $X_s$ into an ordered SVG token sequence within the same representation space as the pre-trained VLM. Following IconShop [57], we flatten the layered structure of the SVG script by concatenating different paths into a single command sequence, where each path begins with the drawing commands followed by point coordinates. Therefore, each SVG sequence could be represented as a flattened sequence. As the generation identifier, we apply special tokens like `<SOP>` and `<EOS>` to the two ends of a SVG sequence, identifying the begining and ending of a SVG sequence. We assign special tokens for each command type, *i.e.* $\{\mathrm{M}, \mathrm{L}, \mathrm{C}, \mathrm{A}, \mathrm{Z}, \mathrm{F}\}$. To shorten the length of the SVG sequence, we further merge the 2D point coordinates into one token with a mapping function: $< x, y > \rightarrow x \times w + y$, where $w$ is the width of the image. The SVG sequence are then lifted into the same embedding space as the pre-trained VLM with a learnable embedding layer.

**Model Architecture.** OmniSVG adopts Qwen2.5-VL [1], an open-sourced VLM that excels at understanding intricate vision-text inputs, as its backbone (Fig. 2) to produce precise and compact SVG outputs. OmniSVG is trained to predict the SVG suffix tokens ($x_s$) conditioned on the mutli-modal instruction prefix tokens ($x_c$) with the standard next-token prediction objective.

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{L} P\left(x_{s,i} \mid x_{s,<i}, x_c\right) \tag{1}$$

Table 2: **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 repectively. Our OmniSVG model demonstrates superior performance compared SOTA SVG generation baselines.

| Evaluation Dataset | Methods | # Tokens | Text-to-SVG | | | | Image-to-SVG | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | FID↓ | CLIP↑ | Aesthetic↑ | HPS↑ | DINO↑ | SSIM↑ | LPIPS↓ | MSE↓ |
| MMSVG-Icon | Vectorfusion [22] | 66.2k | 250.77 | 0.240 | **4.76** | 0.237 | – | – | – | – |
| | SVGDreamer [60] | 132.0k | 308.94 | 0.207 | 4.26 | 0.221 | – | – | – | – |
| | Chat2SVG [56] | 0.6k | 190.87 | **0.299** | 4.41 | **0.247** | – | – | – | – |
| | IconShop [57] | 2.0k | 213.28 | 0.288 | 4.55 | 0.244 | – | – | – | – |
| | LIVE [34] | 52.5k | – | – | – | – | 0.932 | 0.943 | 0.106 | 0.011 |
| | DiffVG [29] | 322.0k | – | – | – | – | 0.940 | 0.954 | 0.066 | **0.002** |
| | GPT-4o [21] | 0.3k | – | – | – | – | 0.860 | 0.792 | 0.403 | 0.124 |
| | StarVector(8B) [42] | 2.0k | – | – | – | – | 0.895 | 0.881 | 0.231 | 0.059 |
| | Vtracer | 52.4k | – | – | – | – | **0.993** | **0.966** | **0.039** | 0.002 |
| | OmniSVG(4B) | 3.8k | 137.40 | 0.275 | 4.62 | 0.244 | 0.993 | 0.950 | 0.050 | 0.006 |
| | OmniSVG-L(8B) | 5.7k | **130.56** | 0.276 | 4.60 | 0.242 | 0.922 | 0.893 | 0.235 | 0.040 |
| MMSVG-Illustration | Vectorfusion [22] | 66.1k | 253.94 | 0.185 | **4.94** | 0.226 | – | – | – | – |
| | SVGDreamer [60] | 132.0k | 419.70 | 0.201 | 4.37 | 0.221 | – | – | – | – |
| | Chat2SVG [56] | 1.0k | 210.03 | **0.283** | 4.45 | **0.250** | – | – | – | – |
| | IconShop [57] | 2.6k | **107.93** | 0.233 | 4.46 | 0.224 | – | – | – | – |
| | LIVE [34] | 52.2k | – | – | – | – | 0.935 | 0.950 | 0.111 | 0.008 |
| | DiffVG [29] | 322.0k | – | – | – | – | 0.945 | 0.955 | 0.065 | **0.001** |
| | GPT-4o [21] | 0.4k | – | – | – | – | 0.875 | 0.854 | 0.373 | 0.077 |
| | StarVector(8B) [42] | 2.6k | – | – | – | – | 0.877 | 0.900 | 0.238 | 0.046 |
| | Vtracer | 57.6k | – | – | – | – | **0.994** | **0.966** | **0.035** | 0.002 |
| | OmniSVG(4B) | 5.8k | 154.37 | 0.226 | 4.56 | 0.232 | 0.899 | 0.906 | 0.237 | 0.034 |
| | OmniSVG-L(8B) | 6.9k | 138.42 | 0.231 | 4.51 | 0.232 | 0.905 | 0.907 | 0.231 | 0.031 |

## 5 Experiments

To validate the effectiveness of our method, we first introduce the baselines (Sec. 5.1). Then, we make quantitative comparisons with prior arts (Secs. 5.2 and 5.3) and conduct ablations (Sec. 5.4) to study the effectiveness of our design.

### 5.1 Baselines

For the text-to-SVG task, we compare our method with language-based (LLM-based) methods, including VectorFusion [22], SVGDreamer [60], Chat2SVG [56] and IconShop [57]. For image-to-SVG task, we compare our method with baseline methods across image vectorization and Multimodal Large Language Modeling approaches, including LIVE [34], DiffVG [29], StarVector [42], Vtracer [12] and GPT-4o [21] using the official implementations with the hyperparameters proposed by the authors, and apply their pre- and post-processing code as required.

### 5.2 Quantitative Comparisons

We compare our OmniSVG with other baseline methods on the "text-to-SVG" and "image-to-SVG" tasks in our MMSVG-Bench. In addition to the metrics mentioned in Sec. 3, we also report the average token length (# tokens) of a generated SVG sample utilizing the Qwen2.5-VL [1] tokenizer.

As shown in Tab. 2, OmniSVG demonstrates strong performance compared to state-of-the-art baselines in text-to-SVG generation, achieving superior FID scores and competitive CLIP score, aesthetic quality, and HPS. For image-to-SVG, OmniSVG also achieves competitive results with traditional vectorization methods, *i.e.* LIVE [34], DiffVG [29], and VTracer [12], but with a much shorter sequence length. When comparing to auto-regressive methods, *i.e.* GPT-4o [21] and StarVector [42], OmniSVG also achieves a superior performance across all metrics. The above results indicate that OmniSVG effectively balances the generation cost and the visual quality when generating SVGs according to multi-modal conditions.

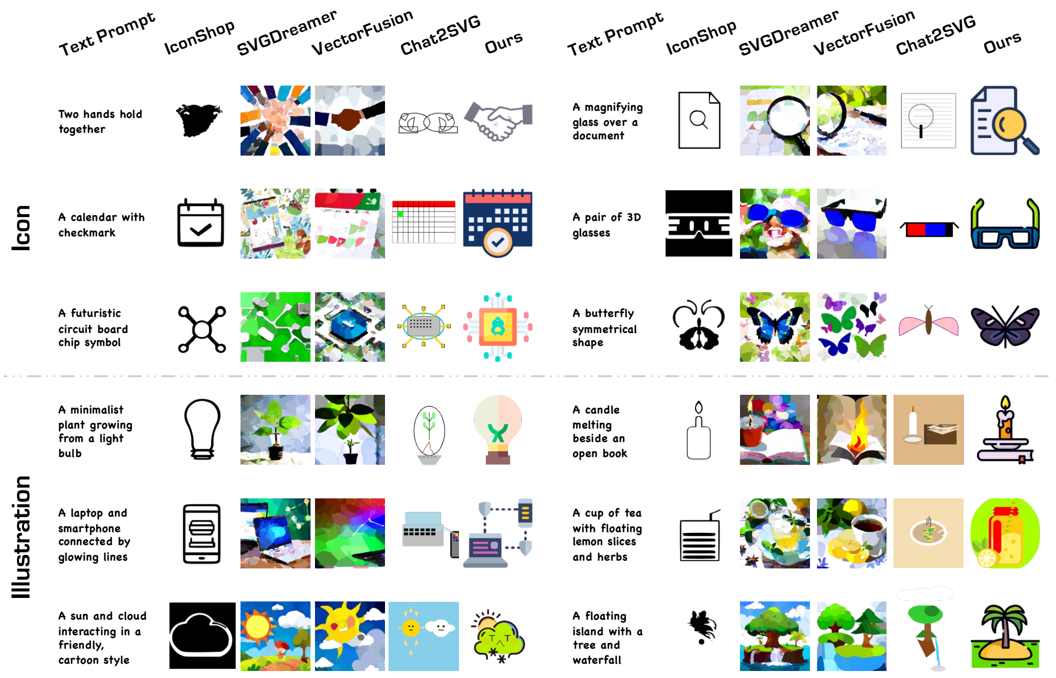

Text Prompt
IconShop
SVGDreamer
VectorFusion
Chat2SVG
Ours
Icon
Illustration
Two hands hold together
A calendar with checkmark
A futuristic circuit board chip symbol
A magnifying glass over a document
A pair of 3D glasses
A butterfly symmetrical shape
A minimalist plant growing from a light bulb
A laptop and smartphone connected by glowing lines
A sun and cloud interacting in a friendly, cartoon style
A candle melting beside an open book
A cup of tea with floating lemon slices and herbs
A floating island with a tree and waterfall

Figure 3: **Qualitative Comparison with SOTA Methods on Text-to-SVG Task**. We compare the propose method with SOTA Text-to-SVG methods on our evaluation benchmarks, namely Icon and Illustration.

## 5.3 Qualitative Evaluations

**Text-to-SVG task**. We compare our method with baseline approaches using seven distinct text prompts for the text-to-SVG task, as shown in Fig. 4. Optimization-based methods like SVG-Dreamer [60] and VectorFusion [22] require significant computation time due to their iterative optimization processes, which, while effective for refining SVG details, are computationally expensive. Auto-regressive methods, such as IconShop [57] and Chat2SVG [56], generate SVGs more quickly by leveraging pre-trained models but have notable limitations. IconShop produces monochrome SVGs, restricting its applicability, while Chat2SVG, though flexible, generates less detailed and semantically consistent SVGs in its first stage. Our OmniSVG consistently outperforms all baselines across various text prompts in generating high-fidelity SVGs with rich color, geometric accuracy, and the ability to handle complex visual cues.

**Image-to-SVG Task.** We compare our method with classical image vectorization approaches, including DiffVG [29], LIVE [34], and VLM-based methods GPT-4o [21], StarVector [42] and Vtracer [12] As shown in Fig. 4, our method outperforms these baselines in both quality and efficiency. Optimization-based methods like DiffVG and LIVE perform well on simple icons but struggle with complex images, often generating visual artifacts. The GPT-4o model, while capable of generating SVGs for complex images, is limited to icon-level outputs and cannot handle detailed illustrations. StarVector excels at simple icons but fails to produce accurate SVGs for more intricate images, highlighting its limited generalization capability. Vtracer is an image processing algorithm designed to convert raster images into SVGs. In contrast, OmniSVG effi-

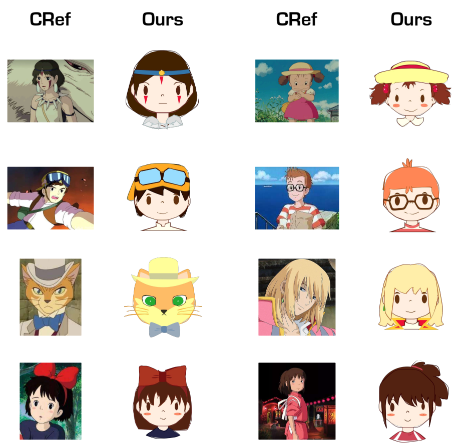


Figure 5: **Generated SVG with Character-Reference (CRef) by OmniSVG.**

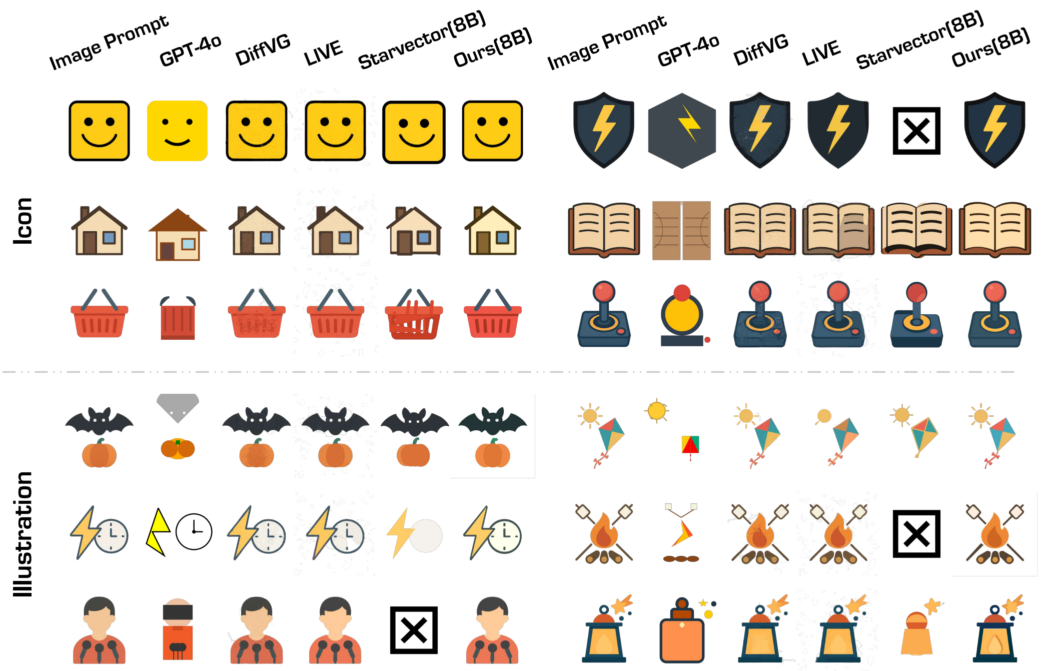


Figure 4: **Qualitative Comparison with SOTA Methods on Image-to-SVG Task**. We compare the propose method with SOTA Image-to-SVG methods on our evaluation benchmarks.

ciently converts a wide range of images, from icons to complex illustrations and character images, into high-quality, editable SVGs. This superior performance in handling diverse visual cues distinguishes OmniSVG from traditional vectorization methods. Additional visual results can be found in Fig. 12. We provide more detailed discussions with existing methods, particularly the recent works LLM4SVG [59] and StarVector [42], in the Sec. D.

**Character-Reference SVG generation task.** As shown in Fig. 5, by training on MMSVG-Character with natural character image and SVG pair data, OmniSVG is capable of generating character SVGs through image references.

## 5.4 Ablation studies

**Effectiveness of SVG Parameterization.** We present a comprehensive comparison among different SVG parameterization strategy with the traditional non-parameterized methods for SVG representation in large language models. We ablates on the parameterization on both coordinate and color attributes of the SVG.

The results, shown in Tab. 3 and Fig. 6 demonstrate that parameterizing both coordinate and color attributes yields a better generation results under all metrics with the shortest token length. It further validates that the efficient token representation allows our method to generate complex SVGs with fewer computational resources. Additionally, qualitative results show that our method outperforms

Table 3: **Quantitative Study on SVG Parameterization.** Ablation studies on color parametrization (abbreviated as color param.) and coordinate paramterization (abbreviated as coord param.) are conducted.

| **Methods** | **Text-to-SVG** | | | | **Image-to-SVG** | | | | **# Tokens** |
|---|---|---|---|---|---|---|---|---|---|
| | FID↓ | CLIP↑ | Aesthetic↑ | HPS↑ | DINO↑ | SSIM↑ | LPIPS↓ | MSE ↓ | |
| w/o param. | 218.76 | 0.185 | 3.43 | 0.138 | 0.741 | 0.718 | 0.315 | 0.182 | 18.5k |
| w/o coordinate param. | 193.42 | 0.216 | 3.90 | 0.169 | 0.826 | 0.809 | 0.248 | 0.119 | 10.2k |
| w/o color param. | <u>167.28</u> | <u>0.269</u> | <u>4.31</u> | <u>0.211</u> | <u>0.895</u> | <u>0.879</u> | <u>0.179</u> | <u>0.053</u> | 6.3k |
| **OmniSVG(4B)** | **145.89** | **0.308** | **4.59** | **0.238** | **0.946** | **0.928** | **0.138** | **0.020** | 4.8k |

Table 4: **Ablation of the Model Size.** As the model size grows, the generated samples are of higher quality.

| Methods | Input | Size | Text-to-SVG | | | | Image-to-SVG | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | FID↓ | CLIP↑ | Aesthetic↑ | HPS↑ | DINO↑ | SSIM↑ | LPIS↓ | MSE↓ |
| FLAN-T5-Base[10] | Text | 223M | 198.48 | 0.158 | 3.38 | 0.085 | – | – | – | – |
| FLAN-T5-Large[10] | Text | 770M | 175.24 | 0.208 | 3.92 | 0.142 | – | – | – | – |
| FLAN-T5-xl[10] | Text | 3B | 160.28 | 0.258 | 4.31 | 0.192 | – | – | – | – |
| blip2-flan-t5-xl[28] | Text/Image | 3.94B | 152.11 | 0.235 | 4.48 | 0.215 | 0.898 | 0.891 | 0.255 | 0.041 |
| **OmniSVG(4B)** | Text/Image | 3.7B | 145.89 | **0.308** | **4.59** | **0.238** | **0.946** | **0.928** | **0.138** | **0.020** |

others, particularly as SVG complexity increases. The non-parameterization method fails to generate SVGs for complex images. These findings underscore the effectiveness of our full parameterization strategy in balancing performance and resource efficiency for SVG generation tasks.

**Ablation studies on model size.** To analyze whether training a larger model benefits SVG generation, we evaluate OmniSVG base models with different sizes on the MMSVG-2M dataset in Tab. 4. We evaluate OmniSVG with base models of varying sizes on the MMSVG-2M dataset in Tab. 4 by progressively scaling up the model size. The results show that as the model size grows, we can generate SVG samples with a better quality.

Table 5: **Ablation on VLM architecture.**

| Vision Model | Language Model | Text-to-SVG | | | | Image-to-SVG | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | FID↓ | CLIP↑ | Aesthetic↑ | HPS↑ | DINO↑ | SSIM↑ | LPIPS↓ | MSE↓ |
| CLIP | Qwen2.5 | 185.31 | 0.249 | 4.52 | 0.215 | 0.867 | 0.856 | 0.267 | 0.058 |
| VQGAN | Qwen2.5 | 198.74 | 0.234 | 4.49 | 0.203 | 0.839 | 0.828 | 0.295 | 0.071 |
| **Qwen2.5-VL-3B-Instruct** | | 145.89 | **0.308** | **4.59** | **0.238** | **0.946** | **0.928** | **0.138** | **0.020** |
| **Qwen2.5-VL-7B-Instruct** | | **134.45** | 0.254 | 4.56 | 0.237 | 0.914 | 0.900 | 0.233 | 0.036 |

**Ablation Studies on the VLM Architecture.** To evaluate the effectiveness of the VLM architecture, we conducted an ablation study replacing it with alternative LLM-based architectures incorporating image encoders such as CLIP ViT-B/32 [39], VQGAN [14], and Qwen2.5-VL [1].

The results in Tab. 5 show that Qwen2.5-VL consistently outperformed all alternatives under all evaluation metrics.

**User Study.** We extract one-tenth of the samples from the evaluation dataset and conducted a user study with 15 participants to evaluate user preferences, vividness, and the alignment between text-to-SVG and image-to-SVG. Participants are asked to assess SVGs generated by different models based on 150 text descriptions and 150 image prompts, comparing the results generated using our method and baseline models. The results in Fig. 7 show that OmniSVG is widely preferred, with higher scores for vividness and superior semantic alignment with the input conditions.

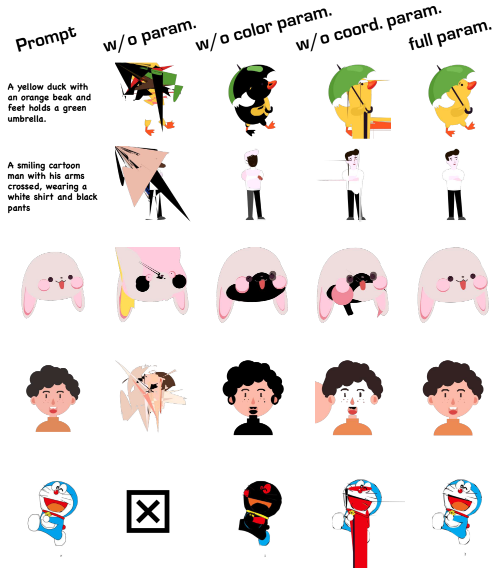

Prompt
w/o param.
w/o color param.
w/o coord. param.
full param.
A yellow duck with an orange beak and feet holds a green umbrella.
A smiling cartoon man with his arms crossed, wearing a white shirt and black pants

Figure 6: **Qualitative Study on Parametrization.**

## 6 Conclusions

**Conclusions.** We introduce OmniSVG, a unified framework for multimodal SVG generation that leverages pre-trained Vision-Language Models (VLMs). By parameterizing SVG commands and coordinates as discrete tokens, OmniSVG efficiently decouples structural logic from geometry, addressing issues like "coordinate hallucination" while maintaining design expressiveness. Our method outperforms existing approaches in both quality and efficiency, offering high-quality, editable SVG across various design domains. Additionally, we proposed MMSVG-2M, a large-scale multimodal dataset with two million annotated SVG assets and a standardized evaluation protocol.

Extensive experiments show that OmniSVG surpasses prior SVG generation methods in various conditional generation tasks, highlighting its potential for integration into professional SVG design workflows.

**Limitations and Future Work.** During inference, OmniSVG generates tens of thousands of tokens for complex samples, which inevitably leads to a considerable generation time. OmniSVG is only bounded by vector style image prompt and fails on natural images. As for future work, recent endeavors on multi-token prediction [15, 2] and KV-cache compression [68, 3] provide a promising way to save the generation cost. Additionally, the auto-regressive nature of OmniSVG also unlocks future opportunities for in-context learning [67, 69, 47], chain-of-thought reasoning [55, 16], and multi-turn interleaved generation [20, 31], thereby providing a more precise user control.

Figure 7: **User Study of OmniSVG and baselines.**

| Method | Preference ↑ | Vividity↑ | Alignment↑ |
|---|---|---|---|
| Vectorfusion [22] | 35 | 58 | 76 |
| SVGDreamer [60] | 41 | 65 | 79 |
| Chat2SVG [56] | 55 | 61 | 86 |
| IconShop [57] | 79 | 57 | 75 |
| GPT-4o [21] | 38 | 54 | 80 |
| StarVector(8B) [42] | 37 | 81 | 68 |
| DiffVG [29] | 88 | 76 | 96 |
| LIVE [34] | 86 | 70 | 95 |
| **OmniSVG** | **96** | **88** | **98** |

## Acknowledgements

This work is in part supported by National Key R&D Program of China (Grant No. 2022ZD0160103), National Natural Science Foundation of China (Grant No. 62276067), and National Natural Science Foundation of China (Grant No. 62472104).

The computations in this research were performed using the CFFF platform of Fudan University.

## References

[1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[2] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, and Tri Dao. Medusa: Simple framework for accelerating llm generation with multiple decoding heads. *Retrieved December*, 2023.

[3] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

[4] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. Deepsvg: A hierarchical generative network for vector graphics animation. *NeurIPS*, 2020.

[5] Sijin Chen, Xin Chen, Anqi Pang, Xianfang Zeng, Wei Cheng, Yijun Fu, Fukun Yin, Billzb Wang, Jingyi Yu, Gang Yu, et al. Meshxl: Neural coordinate field for generative 3d foundation models. *NeurIPS*, 2024.

[6] Sijin Chen, Xin Chen, Chi Zhang, Mingsheng Li, Gang Yu, Hao Fei, Hongyuan Zhu, Jiayuan Fan, and Tao Chen. Ll3da: Visual interactive instruction tuning for omni-3d understanding reasoning and planning. In *CVPR*, 2024.

[7] Zehao Chen and Rong Pan. Svgbuilder: Component-based colored svg generation with text-guided autoregressive transformers. *arXiv preprint arXiv:2412.10488*, 2024.

[8] Wei Cheng, Ruixiang Chen, Siming Fan, Wanqi Yin, Keyu Chen, Zhongang Cai, Jingbo Wang, Yang Gao, Zhengming Yu, Zhengyu Lin, et al. Dna-rendering: A diverse neural actor repository for high-fidelity human-centric rendering. In *ICCV*, 2023.

[9] Wei Cheng, Su Xu, Jingtan Piao, Chen Qian, Wayne Wu, Kwan-Yee Lin, and Hongsheng Li. Generalizable neural performer: Learning robust radiance fields for human novel view synthesis. *arXiv preprint arXiv:2204.11798*, 2022.

[10] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *JMLR*, 2024.

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

[12] Vision Cortex. Vtracer. `https://www.visioncortex.org/vtracer-docs`, 2023.

[13] Nyanko Devs. Danbooru2023: A large-scale crowdsourced and tagged anime illustration dataset. Hugging Face, 2023.

[14] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021.

[15] Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

[16] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[17] Han Guo, Songlin Yang, Tarushii Goel, Eric P Xing, Tri Dao, and Yoon Kim. Log-linear attention. *arXiv preprint arXiv:2506.04761*, 2025.

[18] David Ha and Douglas Eck. A neural representation of sketch drawings. In *ICLR*, 2018.

[19] Teng Hu, Ran Yi, Baihong Qian, Jiangning Zhang, Paul L Rosin, and Yu-Kun Lai. Supersvg: Superpixel-based scalable vector graphics synthesis. In *CVPR*, 2024.

[20] Minbin Huang, Yanxin Long, Xinchi Deng, Ruihang Chu, Jiangfeng Xiong, Xiaodan Liang, Hong Cheng, Qinglin Lu, and Wei Liu. Dialoggen: Multi-modal interactive dialogue system for multi-turn text-to-image generation. *arXiv preprint arXiv:2403.08857*, 2024.

[21] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[22] Ajay Jain, Amber Xie, and Pieter Abbeel. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In *CVPR*, 2023.

[23] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[24] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, et al. The stack: 3 tb of permissively licensed source code. *arXiv preprint arXiv:2211.15533*, 2022.

[25] Kozea. Cairosvg. `https://cairosvg.org/`, 2023.

[26] Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2024.

[27] Black Forest Labs. Flux.1-redux-dev. `https://huggingface.co/black-forest-labs/FLUX.1-Redux-dev`, 2024.

[28] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023.

[29] Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *SIGGRAPH Asia*, 2020.

[30] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

[31] Ziyu Liu, Tao Chu, Yuhang Zang, Xilin Wei, Xiaoyi Dong, Pan Zhang, Zijian Liang, Yuanjun Xiong, Yu Qiao, Dahua Lin, et al. Mmdu: A multi-turn multi-image dialog understanding benchmark and instruction-tuning dataset for lvlms. *arXiv preprint arXiv:2406.11833*, 2024.

[32] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. A learned representation for scalable vector graphics. In *CVPR*, 2019.

[33] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[34] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *CVPR*, 2022.

[35] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[36] Dongwei Pan, Long Zhuo, Jingtan Piao, Huiwen Luo, Wei Cheng, Yuxin Wang, Siming Fan, Shengqi Liu, Lei Yang, Bo Dai, et al. Renderme-360: a large digital asset library and benchmarks towards high-fidelity head avatars. *NeurIPS*, 2023.

[37] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *ICLR*, 2023.

[38] 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 *ICML*, 2021.

[39] 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 *ICML*, 2021.

[40] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020.

[41] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *CVPR*, 2021.

[42] Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556*, 2023.

[43] Christoph Schuhmann. Improved aesthetic predictor. `https://github.com/christophschuhmann/improved-aesthetic-predictor`, 2022.

[44] I-Chao Shen and Bing-Yu Chen. Clipgen: A deep generative model for clipart vectorization and synthesis. *TVCG*, 2022.

[45] Yiren Song, Xuning Shao, Kang Chen, Weidong Zhang, Zhongliang Jing, and Minzhe Li. Clipvg: Text-guided image manipulation using differentiable vector graphics. In *AAAI*, 2023.

[46] Hao Su, Xuefeng Liu, Jianwei Niu, Jiahe Cui, Ji Wan, Xinghao Wu, and Nana Wang. Marvel: Raster gray-level manga vectorization via primitive-wise deep reinforcement learning. *TCSVT*, 2023.

[47] Quan Sun, Yufeng Cui, Xiaosong Zhang, Fan Zhang, Qiying Yu, Yueze Wang, Yongming Rao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Generative multimodal models are in-context learners. In *CVPR*, 2024.

[48] Zecheng Tang, Chenfei Wu, Zekai Zhang, Mingheng Ni, Shengming Yin, Yu Liu, Zhengyuan Yang, Lijuan Wang, Zicheng Liu, Juntao Li, et al. Strokenuwa: Tokenizing strokes for vector graphic synthesis. *arXiv preprint arXiv:2401.17093*, 2024.

[49] Zecheng Tang, Chenfei Wu, Zekai Zhang, Mingheng Ni, Shengming Yin, Yu Liu, Zhengyuan Yang, Lijuan Wang, Zicheng Liu, Juntao Li, et al. Strokenuwa: Tokenizing strokes for vector graphic synthesis. *arXiv preprint arXiv:2401.17093*, 2024.

[50] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[51] Yingtao Tian and David Ha. Modern evolution strategies for creativity: Fitting concrete images and abstract concepts. In *Artificial Intelligence in Music, Sound, Art and Design*, 2022.

[52] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

[53] Yizhi Wang and Zhouhui Lian. Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning. *TOG*, 2021.

[54] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 2004.

[55] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

[56] Ronghuan Wu, Wanchao Su, and Jing Liao. Chat2svg: Vector graphics generation with large language models and image diffusion models. *arXiv preprint arXiv:2411.16602*, 2024.

[57] Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *TOG*, 2023.

[58] Xiaoshi Wu, Keqiang Sun, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score: Better aligning text-to-image models with human preference. In *ICCV*, 2023.

[59] Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. Empowering llms to understand and generate complex vector graphics. *arXiv preprint arXiv:2412.11102*, 2024.

[60] Ximing Xing, Haitao Zhou, Chuang Wang, Jing Zhang, Dong Xu, and Qian Yu. Svgdreamer: Text guided svg generation with diffusion model. In *CVPR*, 2024.

[61] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[62] Yiying Yang, Fukun Yin, Wen Liu, Jiayuan Fan, Xin Chen, Gang Yu, and Tao Chen. Pm-inr: Prior-rich multi-modal implicit large-scale scene neural representation. In *AAAI*, 2024.

[63] Fukun Yin, Xin Chen, Chi Zhang, Biao Jiang, Zibo Zhao, Wen Liu, Gang Yu, and Tao Chen. Shapegpt: 3d shape generation with a unified multi-modal language model. *TMM*, 2025.

[64] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

[65] Zhengming Yu, Wei Cheng, Xian Liu, Wayne Wu, and Kwan-Yee Lin. Monohuman: Animatable human neural field from monocular video. In *CVPR*, 2023.

[66] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[67] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. What makes good examples for visual in-context learning? *NeurIPS*, 2023.

[68] Xiabin Zhou, Wenbin Wang, Minyan Zeng, Jiaxian Guo, Xuebo Liu, Li Shen, Min Zhang, and Liang Ding. Dynamickv: Task-aware adaptive kv cache compression for long context llms. *arXiv preprint arXiv:2412.14838*, 2024.

[69] Yucheng Zhou, Xiang Li, Qianning Wang, and Jianbing Shen. Visual in-context learning for large vision-language models. *arXiv preprint arXiv:2402.11574*, 2024.

[70] Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. Vgbench: A comprehensive benchmark of vector graphics understanding and generation for large language models. In *EMNLP*, 2024.

# Appendix

## A Additional Details of MMSVG-2M dataset

### A.1 Samples of MMSVG-2M Dataset

We visualize samples of our MMSVG-2M dataset in Fig. 8. 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.

Table 6: **Data Statistics for MMSVG-2M.** Our MMSVG-2M consists of 1.1 million SVG icons, 0.5 million SVG illustrations, and 0.4 million SVG anime characters.

| Dataset | Train | Val | Total | Source | Token Length |
|---|---|---|---|---|---|
| **MMSVG-Icon** | 990k | 110k | 1,100k | Iconfont | 2.2k $\pm$ 0.9k |
| **MMSVG-Illustration** | 450k | 50k | 500k | IconScout | 8.1k $\pm$ 3.3k |
| **MMSVG-Character** | 350k | 50k | 400k | Freepik & generated | 28k $\pm$ 7.3k |

### 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 [28], thereby providing textual descriptions that enable us to fine-tune our model to follow these instructions. We use CairoSVG [25] for rasterization and remove samples that produced completely white images.

**Annotation.** We employ an off-the-shelf VLM, specifically BLIP-2 [28], 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. 10 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 charecter

**Instructions for Different Tasks**

- **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 a natural image as input, please generate the simplified character SVG based on the reference input image.

Table 7: **Instructions for Different Tasks**. Instructions including annotation, text-to-SVG, image-to-SVG and character-reference SVG generation.

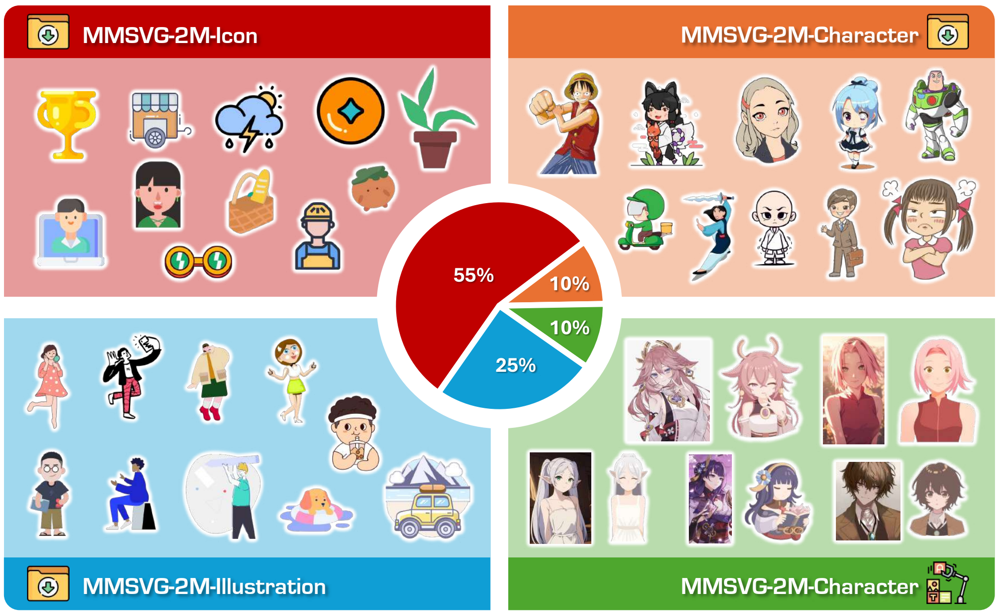


Figure 8: **Samples from MMSVG-2M Dataset**. The proposed MMSVG-2M dataset can be separated into three subset, namely Icon, Illustration and Character. Samples from Icon, Illustration and part of Character subsets are downloaded from Internet. Another part of Character subset is generated by our data creation pipeline, which can provide image and SVG pairs for image prompting task.

reference image and the original image with the VLM architecture. The list of instruction templates for different tasks are shown in Tab. 7.

### A.3 Character-SVG Pairs Construction

As illustrated in the Fig. 6, 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 [26]-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 [27], 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 [26]. However, in practice, the original Redux [27] conditioning proves to be overly strong. To address this, we adopt a community-implemented variant of Redux that downsamples the image embeddings in 2D space. As observed in our experiments shown in Fig. 9, a downsampling factor between $2\times$ and $3\times$ yields the most reasonable SVG-style character references. Finally, we employ VTracer [12] 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 [13] dataset and apply the aforementioned pipeline with motion and expression keywords like previous works [8, 9, 36, 65]. We compare the raw FLUX [26] 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. 11, 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.

### B.2 Implementation Details

We train our models in bfloat16 with the ZeRO-2 strategy [40] for memory-efficient training. We also adopt the AdamW [33] optimizer with a learning rate decaying from $3\times10^{-4}$ to $3\times10^{-6}$ 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.

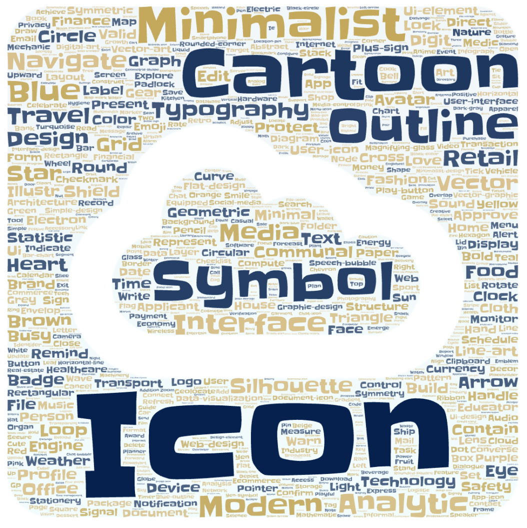


Figure 10: **Word Cloud Visualization of Label Distribution in the MMSVG-2M Dataset.** The size of each label corresponds to its frequency of occurrence. The larger the label, the more frequently it appears in the dataset.

## C Additional Results

As list in full comparisons in Tab. 2, 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 [22], SVGDreamer [60], Chat2SVG [56] and IconShop [57]. For image-to-SVG task, we compare our method with baseline methods across image vectorization and Multimodal Large Language Modeling approaches, including LIVE [34], DiffVG [29], StarVector [42] and GPT-4o [21] 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.

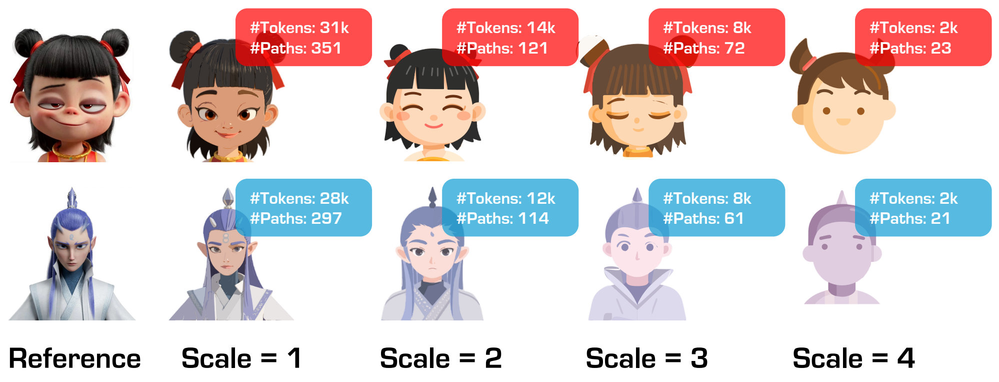
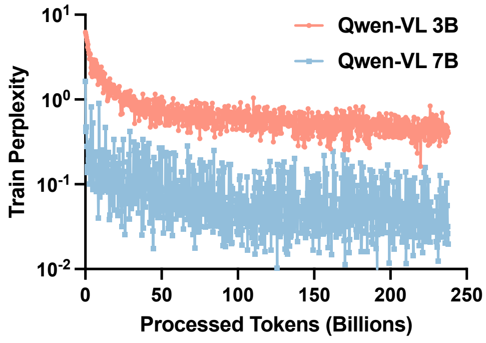
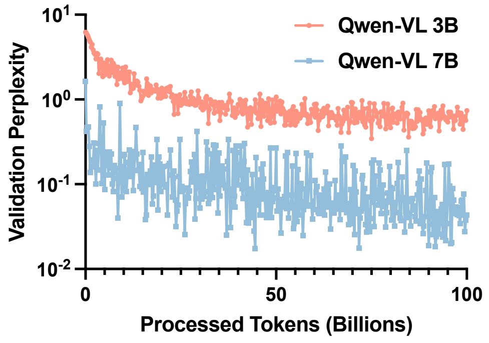


Figure 9: **Image Prompting Dataset Creation of MMSVG-2M Character**. By utilizing FLUX-Redux and SVG vectorization tools, image prompting data pairs can be generated. We adpot FLUX-Redux downsampling scale with 2, 3 in practice by trading-off the character similarity and complexity of generated SVG.



(a) Training PPL for our models.



(b) Validation PPL for our models.

Figure 11: **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.

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 [42] 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 datset, 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** [60] 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** [22] 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** [56] 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 optimization pipeline, guided by image diffusion models, refines paths in latent space and adjusts point coordinates to enhance geometric complexity.

**IconShop** [57] 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

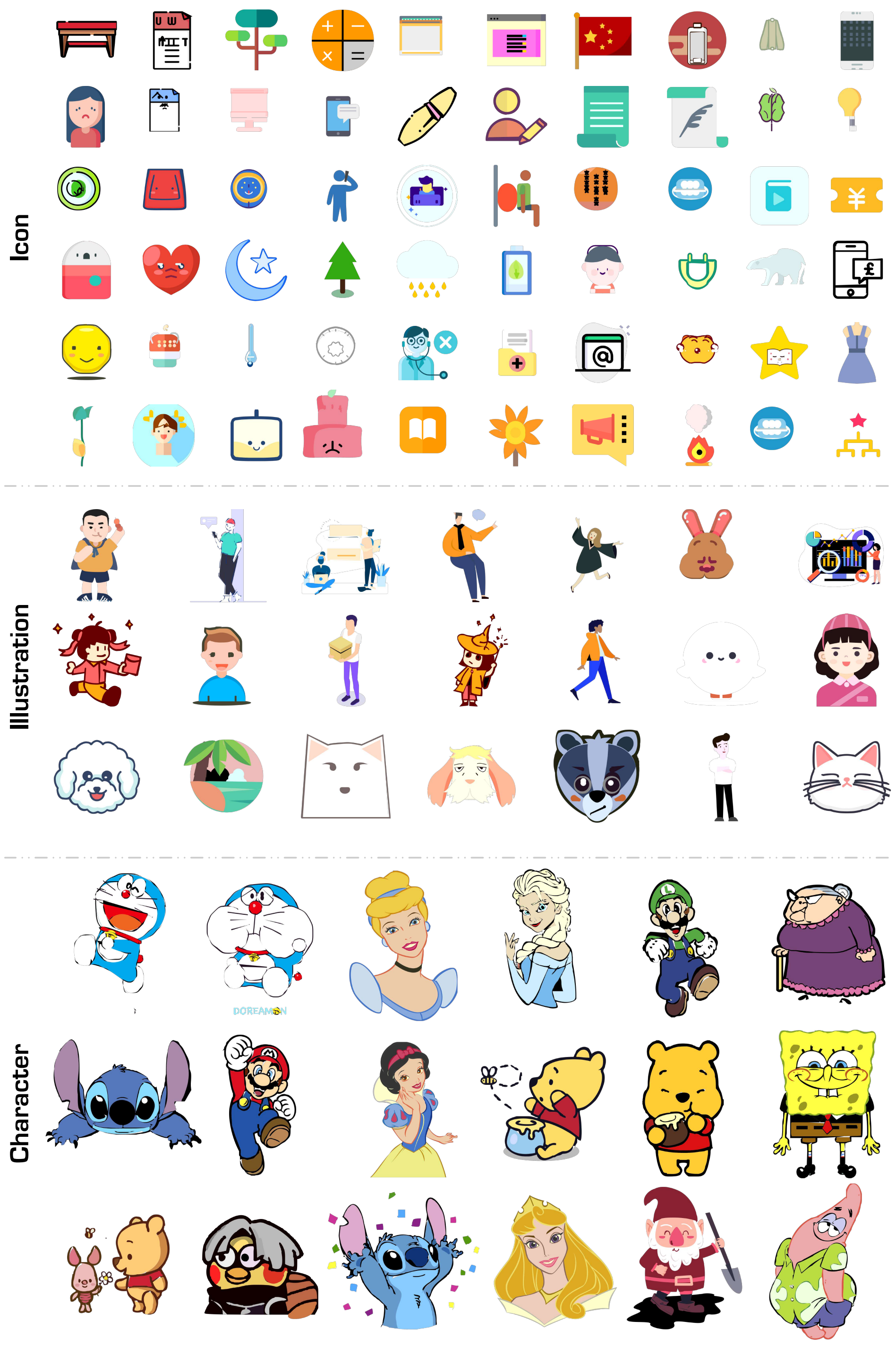


Figure 12: **Illustration of the SVG Generation Capabilities of OmniSVG.**

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** [59] 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) [34] 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 [29]), LIVE enables direct optimization of paths under raster image supervision while controlling shape complexity by adjusting the number of path segments. It introduces component-wise path initialization, identifying key visual components to ensure efficient topology extraction and minimize redundant shapes.

**DiffVG** [29] is a landmark in vector graphics research, pioneering deep learning-based methods with the first differentiable vector graphics rasterization pipeline. By leveraging a combination of anti-aliasing techniques and gradient-based optimization, DiffVG ensures differentiability. Unlike methods relying on non-differentiable curve-to-mesh conversions, DiffVG employs a forward-backward rasterization process, where the forward pass generates antialiased images and the backward pass computes gradients with respect to vector graphic parameters.

**StarVector** [42] works directly in the SVG code space, leveraging visual understanding to apply accurate SVG primitives. StarVector employs a transformer-based architecture that integrates an image encoder with a language model, enabling it to process visual inputs and produce precise SVG code. StarVector effectively handles diverse SVG types, including icons, logos, and complex diagrams, demonstrating robust generalization across various vectorization tasks. However, with a 16k token context window, StarVector may struggle to process highly complex SVGs that require longer sequences.

**Vtracer** [12] is an image processing algorithm designed to convert raster images into SVGs. The algorithm follows a three-step pipeline, which involves the hierarchical clustering of images for vectorization. Initially, the pixels are transformed into paths, which are subsequently simplified into polygons. In the final step, these polygons are smoothed and approximated using a Bezier curve fitter.