Hierarchical SVG Tokenization: Learning Compact Visual Programs for Scalable Vector Graphics Modeling

Tencent HunYuan

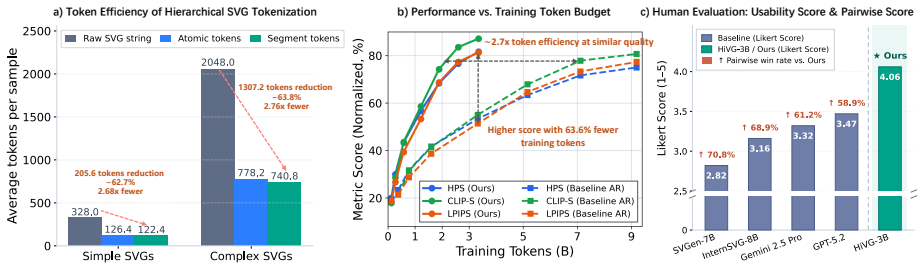

Fig. 1: Sequence-length compression, token-efficient scaling, and human evaluation. (a) HiVG tokenization compresses SVG sequences by 62.7%–63.8% (2.68×–2.76×). (b) HiVG reaches comparable quality with approximately 2.7× fewer training tokens. (c) HiVG achieves the best human evaluation results, scoring 4.06 in usability and winning 58.9%–70.8% in pairwise comparisons against baselines.

Abstract. Recent large language models have shifted SVG generation from differentiable rendering optimization to autoregressive program synthesis. However, existing approaches still rely on generic byte-level tokenization inherited from natural language processing, which poorly reflects the geometric structure of vector graphics. Numerical coordinates are fragmented into discrete symbols, destroying spatial relationships and introducing severe token redundancy, often leading to coordinate hallucination and inefficient long-sequence generation. To address these challenges, we propose HiVG, a hierarchical SVG tokenization framework tailored for autoregressive vector graphics generation. HiVG decomposes raw SVG strings into structured *atomic tokens* and further compresses executable command–parameter groups into geometry-constrained *segment tokens*, substantially improving sequence efficiency while preserving syntactic validity. To further mitigate spatial mismatch, we introduce a Hierarchical Mean–Noise (HMN) initialization strategy that injects numerical ordering signals and semantic priors into new token embeddings. Combined with a curriculum training paradigm that progressively increases program complexity, HiVG enables more stable learning of executable SVG programs. Extensive experiments on both text-to-SVG and image-to-SVG tasks demonstrate improved generation fidelity, spatial consistency, and sequence efficiency compared with conventional tokenization schemes.

Keywords: Autoregressive SVG Generation · Hierarchical Representation · Geometry-Aware Tokenization · Structure-Preserving Token Compression

1 Introduction

Scalable Vector Graphics (SVG) generation has recently attracted increasing attention due to its infinite-resolution rendering and highly compact representation. Early methods [7, 12, 16, 29, 30, 39, 40], typically formulate SVG generation as a differentiable rendering or optimization problem, where vector primitives are iteratively adjusted to approximate a target image. However, such approaches often suffer from high computational cost and limited ability to model the compositional structure of SVG programs. With the rapid advancement of Large Language Models (LLMs), recent works have shifted towards treating SVG as executable code and generating it through autoregressive modeling [24, 32, 34, 37, 43].

However, current LLM-based SVG generation methods still suffer from fundamental representation issues. Alongside this architectural shift, we observe a concerning trend: existing methods inherit coordinate representations from the pre-trained LLM, which often leads to coordinate hallucination [10, 43]. Recent works attempt to alleviate this through pre-processing of raw SVG strings (e.g., converting to relative coordinates [37] or flattened coordinates [43]). Yet the fundamental problem remains unresolved: *tokenized coordinates fail to reflect their underlying geometric relationships.* Specifically, standard byte-level tokenizers [27] treat numerical coordinates as discrete strings rather than continuous spatial values (e.g., "100" is tokenized as "1", "0", "0"). Consequently, this fragmentation not only destroys the inherent spatial relationships of the coordinates but introduces severe token redundancy during generation.

Beyond the coordinate hallucination, another key challenge lies in the inefficient representation of SVG sequences. Existing works also struggle when generating complex SVG content [31, 43]. A common workaround is to expand the model context window. However, we argue that this limitation fundamentally stems from the inherently low information density of raw SVG tokens compared to natural language. While a semantic word usually requires only 1–2 tokens, even a simple SVG shape may be represented by a long string of drawing commands and coordinates, which may expand to tens or even hundreds of tokens after tokenization. Such redundancy contradicts the structural compactness that makes SVG appealing in the first place. Above observations lead us to ask: **how can we rethink the tokenization paradigm to natively align with the underlying properties of vector graphics? The above cues reveal that the devil is in the token compression**. We provide an affirmative answer to this challenge by proposing **HiVG**, a novel, **hi**erarchical S**VG** tokenizer tailored exclusively for autoregressive generation that decomposes vector graphics into structure-preserving components instead of flat byte streams. As shown in Fig. 2, we first transform raw SVG strings into foundational **atomic tokens** that strictly separate structures, drawing commands, coordinates, and attributes. To fully exploit the inherently renderable patterns of SVG, we design a merging strategy that compresses vector sequences into composite **segment tokens** under geometric constraints. This hierarchical compression drastically shortens coordinate-heavy sequences while guaranteeing that every merged token remains a syntactically valid, executable geometric primitive. Under this

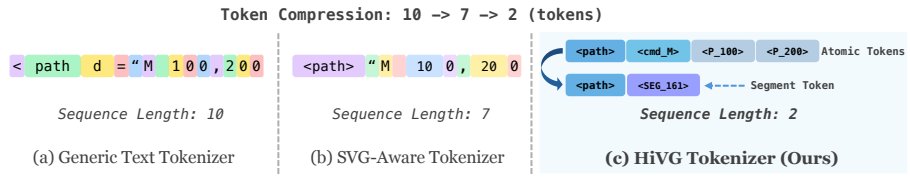

Fig. 2: Comparison of SVG tokenization strategies. (a) A generic LLM tokenizer [2, 42] treats SVG as plain text and splits it into subword tokens, producing long token sequences. (b) An SVG-aware tokenizer [32, 37] improves structural awareness by tokenizing SVG elements and attributes, but geometric primitives remain fragmented into many numeric coordinate tokens. (c) Our HiVG tokenizer introduces a hierarchical representation that groups drawing commands together with their associated coordinates into reusable segment tokens, enabling substantial sequence compression ($10 \rightarrow 7 \rightarrow 2$).

hierarchical representation, segment tokens reduce sequence length by up to 63.8% relative to raw-string tokenization on Qwen [2] (see Fig. 1 (a)).

To further resolve the spatial mismatch caused by discretized coordinate tokens, we introduce a Hierarchical Mean-Noise (HMN) initialization strategy. Instead of randomly initializing the newly introduced SVG vocabulary, HMN explicitly injects numeric ordering signals and semantic priors into token embeddings. These signals are projected through a Gaussian–polynomial basis, enabling embeddings to preserve continuous spatial relationships among coordinates. Our experiments demonstrate that such mathematically grounded initialization provides the model with native spatial awareness from the very beginning of training. Combined with the hierarchical representation, HiVG reaches higher visual quality with approximately 2.7× fewer training tokens (see Fig. 1 (b)).

In summary, our contributions are three-fold: (1) We propose HiVG, a hierarchical SVG tokenization framework that decomposes raw SVG code into atomic tokens and compresses command–parameter groups into executable segment tokens, substantially reducing sequence length while preserving syntactic validity. (2) We introduce Hierarchical Mean–Noise (HMN) initialization, which injects numeric ordering signals and semantic priors into new token embeddings to improve spatial awareness and coordinate consistency. (3) We adopt a three-stage curriculum that progressively increases program depth, leading to more stable optimization and improved generalization to long SVG sequences on both text-to-SVG and image-to-SVG tasks.

2 Related Work

2.1 Parametric Vector Graphics Paradigm

Recent advancements in Scalable Vector Graphics (SVG) generation can be broadly distinguished by how they represent the underlying graphic data. Early optimization-based methods [7, 12, 16, 29, 30, 39, 40] treat SVGs as collections of stroke parameters and iteratively optimize them via differentiable renderers. Another line of research projects SVG commands and their numerical attributes

into continuous latent spaces to learn compact implicit representations [3,28,38]. More recently, research has shifted toward representing SVGs as sequences of discrete tokens. Consistent with this trend, recent efforts closely mirror the evolution of broader LLM frameworks, incorporating large-scale datasets [24,31,32,37, 43], reinforcement learning techniques [25, 31, 36], and unified tasks [15, 32, 43]. While these high-level integrations have led to notable progress, tokenization itself remains relatively underexplored. Although some works embed SVG commands to better capture their semantics [37, 43], this does not necessarily yield a principled tokenization scheme. This observation motivates our work.

2.2 Token Representation and Compression

Efficient token representation is fundamental to autoregressive sequence modeling. A longstanding belief holds that compression is closely connected to intelligence, with some researchers suggesting that they are fundamentally equivalent [6, 11]. In the field of language modeling, this principle is exemplified by Byte-Pair Encoding (BPE) [14, 27], which effectively mitigates token sparsity by merging frequently co-occurring characters into robust subword representations. Building upon this, a growing body of works across diverse modalities has explored task-specific compression strategies to adapt complex data for autoregressive modeling. For example, FAST [20] compresses continuous robot action chunks to learn generalizable robotic behaviors, while FreeMesh [17] quantifies 3D mesh sequence learnability by balancing entropy and compression. In the Computer-Aided Design (CAD) domain, CAD-GPT [33] compresses 3D spatial parameters and 2D sketch coordinates into a 1D linguistic token space to enhance the spatial reasoning capabilities. In the domain of SVG, prior methods have also explored various tokenization schemes. For example, DeepSVG [3] represents SVG paths as sequences of drawing commands with associated parameters, while LLM4SVG [37] serializes SVG elements into textual command tokens for autoregressive generation. However, these approaches typically operate at the level of individual coordinates or drawing commands, resulting in lengthy token sequences that hinder both training and inference efficiency. Different from the free-form combination or coordinate-level discretization seen in these prior works, we identify renderable units under geometric constraints to achieve structural token compression for SVG generation. This structural compression yields substantially shorter sequences while preserving geometric fidelity, leading to a superior compression rate and significantly improved computational efficiency.

3 HiVG: Hierarchical SVG Modeling

3.1 Hierarchical SVG Tokenization

A key challenge in autoregressive SVG generation arises from the program-like nature of vector graphics. Although a typical icon contains only a small number

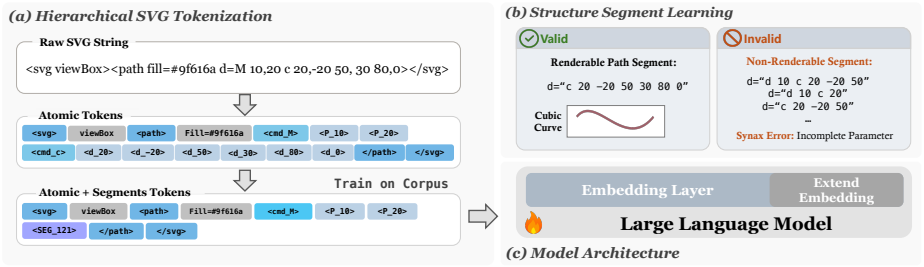

Fig. 3: Overview of HiVG. (a) *Hierarchical SVG Tokenization* (Sec. 3.1). Raw SVG strings are first decomposed into atomic tokens and further compressed into executable segment tokens via structure-aware merging. Only complete command–parameter units are merged to ensure geometric validity. (b) *Structure Segment Learning*. Segment tokens are learned from a large SVG corpus by discovering renderable command–coordinate groups while discarding merges that violate syntactic or geometric constraints. (c) *Model Architecture*. Atomic and Segment tokens extend the embedding space of the base LLM, while training progressively increases program depth to stabilize structural abstraction and global composition.

of visual primitives, its serialized SVG representation is dominated by long sequences of numeric coordinates. Consequently, low-level coordinate tokens overwhelm the context, while higher-level structural signals become sparse. This imbalance makes it difficult for language models to infer element boundaries, preserve structural validity, and reason about geometric relationships across distant parts of the sequence.

To address this issue, we introduce a hierarchical tokenization scheme that decomposes SVG programs into structured atomic tokens and further compresses coordinate-heavy command segments into reusable geometric primitives. An overview of the proposed HiVG framework is illustrated in Fig. 3.

Atomic SVG Tokens. We first transform raw SVG strings into a sequence of atomic tokens that preserve full rendering executability. The atomic vocabulary is partitioned into four disjoint categories:

$$\mathcal{V}_{\text{atomic}} = \mathcal{V}_{\text{struct}} \cup \mathcal{V}_{\text{cmd}} \cup \mathcal{V}_{\text{coord}} \cup \mathcal{V}_{\text{attr}}. \tag{1}$$

Here, $\mathcal{V}_{\text{struct}}$ contains structure tokens that define SVG elements and hierarchical layout (e.g., `<svg>`, `<path>`); $\mathcal{V}_{\text{cmd}}$ consists of path operators such as `<cmd_M>` and `<cmd_C>`; $\mathcal{V}_{\text{attr}}$ represents visual attributes including color and opacity. The coordinate vocabulary $\mathcal{V}_{\text{coord}}$ encodes geometric positions. Given a canvas of size (W, H), raw coordinates are first normalized to the canvas range and uniformly quantized into discrete integer bins, each mapped to a coordinate token.

To improve compositional regularity, path parameters are represented primarily using relative coordinates. Specifically, the first command in each path uses absolute coordinates to establish the starting position, while subsequent parameters are expressed relative to the previous point. This representation reduces global translation variance and exposes recurring geometric patterns across SVG programs. As a result, relative coordinates tend to increase the frequency of re-

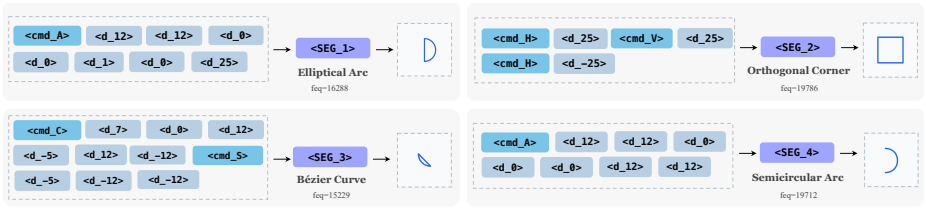

Fig. 4: Learned segment tokens as renderable geometric primitives. Segment-level merging preserves syntactic validity and geometric coherence while shortening token sequences and improving efficiency.

peated command–coordinate groups in the corpus, which facilitates the discovery of reusable geometric primitives during segment learning.

Finally, each SVG command has a fixed parameter arity defined by the SVG specification (e.g., `lineto` requires two coordinates, while cubic Bézier curves require six). This constraint naturally defines executable command–parameter groups consisting of a drawing operator and its required coordinates. We refer to such units as *segments*, which form the basic geometric primitives used for higher-level token construction. Figure 2 illustrates how grouping commands with their parameters enables compact segment-level representations.

Segment Tokens via Structure Segment Learning. As shown in Fig. 2(a,b), conventional tokenization strategies either treat SVG code as plain text or tokenize elements and attributes independently. In both cases, geometric primitives are fragmented into long sequences of coordinate tokens, leading to inefficient and structurally fragmented representations.

To address this issue, we perform token merging over segments rather than individual tokens. Formally, a segment is defined as a command token together with all of its coordinate parameters:

$$s = (\texttt{<cmd>}, c_1, \ldots, c_k), \tag{2}$$

where k is uniquely determined by the command type. Let $\mathcal{S} = \{s_1, s_2, \ldots\}$ denote the multiset of segments extracted from atomic token sequences. We then perform iterative pair merging over $\mathcal{S}$. At iteration t, the most frequent adjacent segment pair is selected

$$(s_i^*, s_j^*) = \arg\max_{(s_i, s_j)} \text{count}(s_i, s_j), \tag{3}$$

and replaced with a new composite segment token if its frequency exceeds $f_{\min}$. After M merging iterations, we obtain a vocabulary of learned segment tokens representing frequently occurring geometric primitives.

Importantly, merging is restricted to segment boundaries, while structure and attribute tokens remain unchanged. As a result, all learned tokens correspond to renderable segment groups as shown in Fig. 4, ensuring syntactic validity and geometric coherence. This segment-level representation significantly reduces sequence length and improves token efficiency.

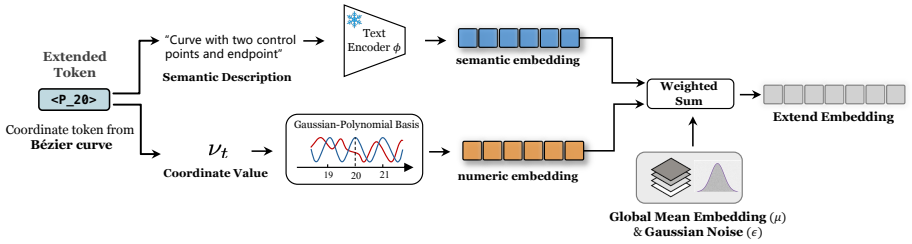

Fig. 5: HMN initialization for structured SVG tokens. Each new token is initialized by combining a global mean-noise prior with a semantic embedding from its textual description; for coordinate tokens, an additional numeric embedding derived from the normalized value through Gaussian–Polynomial basis encoding is added, while non-numeric tokens omit the numeric branch.

3.2 Token Initialization Strategy

Extending the vocabulary of a pretrained language model with domain-specific tokens requires careful embedding initialization. A common practice initializes new embeddings either from isotropic Gaussian noise or from the global mean of the pretrained vocabulary. However, such strategies ignore the internal structure of newly introduced tokens.

For structured vocabularies such as SVG primitives, tokens encode heterogeneous semantics, including element categories, geometric operators, and ordered numeric coordinates. We therefore introduce **Hierarchical Mean–Noise (HMN)** initialization, which combines semantic priors with a structured numeric perturbation. The detailed initialization is illustrated in Fig. 5.

For each newly added token t, its embedding is initialized as

$$\mathbf{e}_t = \lambda_\mu \boldsymbol{\mu} + \lambda_n \boldsymbol{\epsilon} + w_{\text{sem}}\, \phi(\text{desc}_t) + w_{\text{num}}\, \mathbf{d}_t, \tag{4}$$

where $\boldsymbol{\mu}$ denotes the mean embedding of the original vocabulary $\mathcal{V}_0$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ introduces stochastic perturbation, and $\phi(\cdot)$ maps the textual description of token t into the pretrained embedding space using frozen model weights. The final term $\mathbf{d}_t$ encodes numeric information for coordinate tokens.

To construct $\mathbf{d}_t$, the scalar coordinate value v_t (normalized to $[0, 1]$) is first encoded using a low-dimensional basis representation combining Gaussian radial basis functions [23] and polynomial features. This encoding captures both local smoothness and global ordering among coordinate values. The resulting representation is then projected to the model embedding dimension using a fixed random projection matrix inspired by the Johnson–Lindenstrauss transform [8]. Finally, the projected vector is normalized to unit length and used as a small directional perturbation.

This design allows semantic information to remain dominant in the embedding space while numeric structure provides a consistent directional bias for coordinate tokens. Consequently, HMN preserves distributional alignment with the pretrained vocabulary while injecting structured geometric information during initialization.

3.3 Curriculum Training Paradigm

Autoregressive SVG generation requires simultaneously aligning newly introduced structured tokens with the pretrained embedding space and modeling long-range geometric dependencies. Direct training over the full sequence spectrum often destabilizes optimization. We therefore adopt a structure-aware curriculum that progressively increases effective program depth.
Stage 1: Embedding Alignment. Training begins with atomic SVG tokens and moderate-length sequences. This stage aligns newly introduced tokens with the pretrained embedding manifold while stabilizing local geometric transitions.
Stage 2: Structural Abstraction. Segment tokens are then activated, shifting learning from primitive transitions to executable geometric units. The dependency horizon expands while preserving token-space stability.
Stage 3: Global Composition. Finally, full-length SVG programs are introduced. The model focuses on layout coherence and long-range inter-path dependencies.

Each stage expands the training distribution without discarding earlier regimes, separating embedding alignment, structural abstraction, and global composition into distinct optimization phases.

4 Experiments

4.1 Experimental Setup

Dataset Construction. We construct our training corpus by merging three open-source SVG datasets and performing cross-source deduplication, resulting in 2.45M unique SVG samples covering diverse vector graphic categories, including icons, emojis, logos, and interface elements. Before tokenization, we apply a unified filtering and preprocessing pipeline to improve rendering consistency and representation quality. In particular, we remove malformed, unsafe, and non-renderable content, normalize SVG structure and styling, resolve reusable elements and transformations into explicit geometry, map all samples into a unified coordinate space, and quantize coordinates into a discrete tokenizer-friendly format. Samples that remain unstable after preprocessing are discarded. More detailed descriptions of dataset construction, filtering, and preprocessing are provided in Sec. A of the supplementary material.
Training Details. We fully fine-tune Qwen2.5-VL-3B-Instruct [2] under a supervised fine-tuning (SFT) setting. The vision tower and multi-modal projector are frozen, while the language model and newly introduced SVG token embeddings are optimized. All experiments are conducted at a fixed canvas resolution of 784×784. We train for 2 epochs using AdamW with a learning rate of 1×10^{-5} and a warmup ratio of 0.2. New SVG tokens are initialized using the proposed Hierarchical Mean-Noise strategy. The three-stage curriculum is implemented by progressively expanding the training dataset with increasing sequence length ranges while keeping optimization hyperparameters fixed. More detailed descriptions of hyperparameters (Sec. C.1) and training prompt template (Sec. C.2) are provided in Sec. C of the supplementary material.

SVG Token Vocabulary. At a fixed canvas resolution of 784×784, our atomic SVG vocabulary contains 2,450 tokens. It consists of 2,384 coordinate tokens and 66 non-coordinate tokens. The coordinate set includes 795 absolute position tokens (`P_0` ∼ `P_794`) and 1,589 relative offset tokens (`d_-794` ∼ `d_794`), enabling both absolute anchoring and full-range relative moves. The non-coordinate set includes 42 structure tokens (21 SVG elements with paired open/close tags), 20 path-command tokens (10 commands with absolute/relative variants), and 4 arc-flag tokens (`large_{0,1}`, `sweep_{0,1}`). Segment tokens are learned on top of this atomic vocabulary via Structure Segment Learning.

Evaluation. We evaluate structural validity, semantic alignment, visual fidelity, diversity, and perceptual quality under both text-to-SVG and image-to-SVG settings. *(1) Validity and Efficiency.* We report render success rate (Render), average token count (TokCnt), path count (PathCnt), and path command count (CmdCnt). Lower TokCnt, PathCnt, and CmdCnt indicate more compact SVG programs while maintaining rendering fidelity. *(2) Semantic and Visual Quality.* For text-to-SVG, we measure CLIP [22] similarity between rendered images and text prompts. For image-to-SVG, we additionally report CLIP-visual similarity (CLIP-S) between the rendered image and the input reference image, as well as SSIM and LPIPS to assess structural fidelity and perceptual similarity. *(3) Diversity and Preference.* To quantify sample diversity, we extract DINOv2-ViT-Large [19] features from generated images and compute the average pairwise cosine similarity across L samples. Diversity is defined as

$$\text{Diversity} = 1 - \frac{2}{L(L-1)} \sum_{i<j} \cos\left(x_\theta^{(i)}, x_\theta^{(j)}\right), \tag{5}$$

where $x_\theta^{(i)}$ denotes the DINO feature of the i-th sample. Higher diversity corresponds to lower feature similarity among generated outputs.

Perceptual quality is further evaluated using HPSv2 [35], ImageReward [41], PickScore (PickS) [13], and Aesthetic score (Aes) [26].

4.2 Qualitative and Quantitative Analysis

Quantitative Results. Table 1 reports the quantitative comparison on both Image-to-SVG and Text-to-SVG tasks. Compared with existing SVG generation models, HiVG achieves competitive or superior performance across multiple metrics. This improvement suggests that grouping commands with their parameters reduces long-range dependencies and improves structural consistency. Notably, improvements in CLIP-S and LPIPS indicate that segment-level tokens better preserve global geometry while reducing local coordinate drift. On Image-to-SVG reconstruction, our method obtains strong CLIP-S and aesthetic scores while maintaining stable structural validity and visual similarity. On Text-to-SVG generation, HiVG achieves higher PickScore and competitive CLIP and HPS scores, indicating improved prompt alignment and perceptual quality.

Qualitative Results. Figures 6, 7 show representative outputs generated by HiVG. For Image-to-SVG reconstruction, the model accurately preserves object shapes, typography, and layout structures across icons, logos, and UI-style

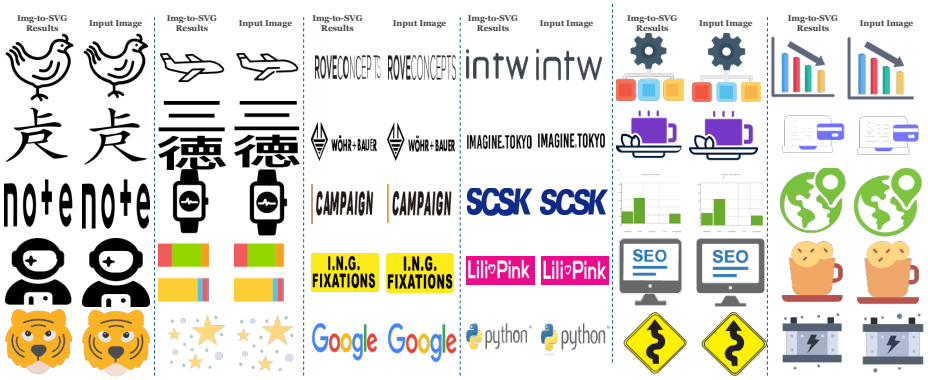

Fig. 6: Image-to-SVG generation results. For each example, the raster input image is shown on the right and the generated SVG rendering on the left. The examples include icons, logos, typography, UI elements, and emoji-style graphics.

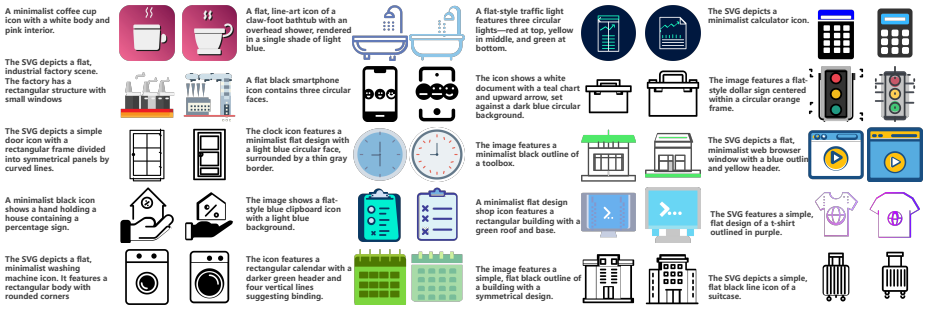

Fig. 7: Text-to-SVG generation results. Text prompts are shown on the left and the generated SVG renderings on the right. The examples cover various object types such as household items, UI elements, buildings, clothing, and symbolic icons.

graphics. For Text-to-SVG generation, HiVG produces visually coherent SVG outputs that follow the prompt description while maintaining geometric layouts.

Comparison with Existing Methods. Figures 8 and 9 provide side-by-side comparisons with recent large multimodal models and SVG generation approaches. In Text-to-SVG generation, several baselines produce incomplete shapes, incorrect layouts, or text mismatches, while HiVG generates more structurally consistent SVG programs. In Image-to-SVG reconstruction, competing methods often introduce geometric distortions or color inconsistencies, whereas HiVG better preserves the global structure and visual details of the input image. It is worth noting that HiVG excels not only at generating iconographic elements, but also at producing textual content with remarkable consistency, a capability rarely achieved by existing methods.

4.3 Human Evaluation.

Automatic metrics mainly measure raster-domain similarity, but do not fully capture human preference or the practical usability of generated SVG code.

Fig. 8: Text-to-SVG comparison. Each row corresponds to a text prompt (left). Columns show SVG renderings generated by different methods. Compared with existing models, HiVG produces SVG outputs with more consistent layout structure and better alignment with the prompt description.

Fig. 9: Image-to-SVG comparison. The first column shows the raster input image, and the remaining columns show SVG reconstructions generated by different methods.

Table 1: Quantitative comparison on Img2SVG and Text2SVG tasks. '-' denotes the method can only accept text input.

Method	Img2SVG						Text2SVG			
	SSIM↑	LPIPS↓	CLIP-S↑	PickS↑	HPS↑	Aes↑	CLIP↑	PickS↑	HPS↑	Aes↑
DeepSeekv3.2 [5]	-	-	-	-	-	-	0.272	20.331	0.192	4.594
Qwen3.5 Plus [21]	0.775	0.228	0.896	22.019	0.175	4.672	0.291	20.972	0.206	4.671
Gemini-2.5-pro [9]	0.790	0.215	0.904	22.346	0.185	4.732	0.284	20.943	0.210	4.765
GPT-5.2 [18]	0.780	0.205	0.930	23.977	0.222	4.841	0.291	21.268	0.214	4.806
Claude-Sonnet-4.5 [1]	0.669	0.292	0.842	22.012	0.164	4.435	0.281	20.562	0.195	4.711
SVGen-7B [31]	-	-	-	-	-	-	0.223	19.023	0.202	4.708
OmniSVG-4B [43]	0.727	0.257	0.813	19.703	0.142	4.466	0.214	19.044	0.150	4.572
OmniSVG-8B [43]	0.764	0.229	0.853	21.401	0.172	4.541	0.229	19.101	0.153	4.662
InternSVG-8B [32]	0.764	0.209	0.877	22.181	0.204	4.638	0.241	19.451	0.174	4.684
HiVG-3B (ours)	0.896	0.114	0.957	21.652	0.221	4.681	0.239	20.575	0.194	4.632

We therefore conduct human evaluation from two perspectives: pairwise visual preference and SVG code usability review.

We randomly sample 60 images from the image-to-SVG test set, covering simple icons, medium-complexity graphics, and more challenging logo- or interface-style compositions, and collect SVG outputs from HiVG-3B and representative open- and closed-source baselines. All results are rasterized at the same resolution for comparison. We recruit 8 professional SVG practitioners as evaluators, and fully randomize method names and output order. In pairwise visual preference, evaluators are shown the reference image and two rendered SVG results, and asked which better reconstructs the reference, with a *tie* allowed. We compare HiVG-3B against SVGen-7B, OmniSVG-8B, InternSVG-8B, Qwen3.5 Plus, Gemini-2.5-pro, GPT-5.2, and Claude-Sonnet-4.5. Each comparison is annotated by 3 evaluators, and the final result is determined by majority vote. Recalling Fig. 1 (c), HiVG achieves the best human evaluation results in usability and pairwise comparisons against other methods.

4.4 Ablation Study

Table 2: Impact of structured SVG modeling. ↑ higher is better, ↓ lower is better. †AR baseline trains on raw SVG string. "Aes" denotes the Aesthetic score.

Variant	Text-to-SVG					Image-to-SVG					
	CLIP↑	DINO↑	HPS↑	PickS↑	Aes↑	SSIM↑	LPIPS↓	CLIP-S↑	HPS↑	PickS↑	Aes↑
AR baseline†	0.2146	0.1520	0.162	19.628	4.548	0.301	0.396	0.797	0.179	19.793	4.553
Ours	**0.2392**	**0.2795**	**0.194**	**20.576**	**4.632**	**0.896**	**0.114**	**0.957**	**0.221**	**21.652**	**4.681**
Improvement	+11.5%	+83.9%	+19.8%	+4.8%	+1.8%	+197.7%	-39.3%	+20.1%	+23.5%	+9.4%	+2.8%

We conduct controlled ablations to verify that each component of HiVG contributes to the final performance under both text-to-SVG and image-to-SVG. We start by comparing the full structured modeling pipeline against an autoregressive baseline trained on raw SVG strings (Tab. 2), establishing the overall gain from modeling SVG as executable programs. We then probe key design choices: the corpus scale used for Structure Segment Learning (Tab. 3), the embedding

Table 3: Effect of Structure Segment Learning (SSL) corpus scale. Dataset statistics and evaluation metrics are reported on both Text-to-SVG and Image-to-SVG tasks. All models use M=500 merges. AT: Atomic Token, ST: Segment Token. Delta rows show changes from previous scale.

Scale	Tokenization Stats			Text-to-SVG					Image-to-SVG					
	Avg Toks	Raw→AT	AT→ST	CLIP↑	DINO↑	HPS↑	PickS↑	Aes↑	SSIM↑	LPIPS↓	CLIP-S↑	HPS↑	PickS↑	Aes↑
$\mathcal{D}_{50k}$	317	2.59x	1.03x	0.2158	0.4072	0.157	19.398	4.412	0.696	0.313	0.803	0.174	19.710	4.398
$\Delta_{50k\to500k}$	*+301*	*+0.04 ×*	*+0.01 ×*	*+4.6%*	*-3.3%*	*+9.6%*	*+3.0%*	*+2.4%*	*+9.1%*	*-26.5%*	*+10.7%*	*+13.8%*	*+5.5%*	*+3.5%*
$\mathcal{D}_{500k}$	618	2.63x	1.04x	0.2257	0.3938	0.172	19.981	4.518	0.759	0.230	0.889	0.198	20.791	4.551
$\Delta_{500k\to1.5M}$	*-66*	*0.00 ×*	*+0.01 ×*	*+1.2%*	*-2.3%*	*+3.5%*	*+0.7%*	*+1.0%*	*+2.4%*	*-9.6%*	*+2.4%*	*+3.5%*	*+1.3%*	*+0.8%*
$\mathcal{D}_{1.5M}$	552	2.63x	1.05x	**0.2283**	0.3848	**0.178**	**20.113**	**4.564**	**0.777**	**0.208**	**0.910**	**0.205**	**21.056**	**4.587**

Table 4: Ablation on token embedding initialization strategies. We report image-to-SVG reconstruction metrics and text-to-SVG generation quality after 1 epoch of training. **Bold**: best result; underline: second best. All methods use the same ∼3,000 SVG tokens and identical training hyperparameters. †Lerp: linear interpolation between text embeddings of "0" and "784", which does *not* preserve numeric semantics due to character-level BPE tokenization of digit strings.

#	Method	Components		Img2SVG			Text2SVG			
		Semantic	Numeric	LPIPS↓	SSIM↑	CLIP-S↑	CLIP↑	PickScore↑	HPS↑	Aes↑
1	Noise	✗	✗	0.226	0.440	0.795	0.207	19.831	0.144	4.250
2	Mean	✗	✗	0.242	0.244	0.755	0.195	**20.110**	0.142	4.830
3	Mean+Noise	✗	✗	0.237	0.523	0.821	0.205	19.645	0.137	4.785
4	Semantic	✓	✗	0.236	0.477	0.811	0.205	19.535	0.132	4.675
5	Semantic+Noise	✓	✗	0.233	0.550	0.830	0.206	19.585	0.135	4.715
6	HMN (Lerp)†	✓	✓	0.182	0.680	0.830	0.206	19.798	0.136	4.755
7	HMN (J-L)	✓	✓	**0.170**	**0.720**	**0.880**	**0.208**	19.965	**0.146**	**4.870**

initialization strategy for new SVG tokens (Tab. 4), and the three-stage curriculum training (Tabs. 5, 6). Finally, we analyze whether scaling segment learning introduces undesirable redundancy patterns in learned segments (Fig. 10).

Overall, these studies reveal three consistent observations: (1) representing SVGs as structured executable programs substantially improves geometric fidelity and semantic alignment; (2) incorporating geometric priors into token embeddings stabilizes early-stage optimization and improves spatial consistency; and (3) progressively increasing sequence complexity through curriculum learning enhances generalization to longer SVG programs.

A. Impact of Structured SVG Modeling. To isolate the effect of our proposed structured, geometry-aware SVG modeling pipeline, we compare against an autoregressive baseline trained on raw SVG sequences with a conventional tokenization scheme. The baseline directly predicts flattened SVG strings with a generic tokenizer, without explicit atomic/segment decomposition. This ablation assesses whether modeling SVG as a structured executable program yields consistent improvements in semantic alignment, perceptual similarity, and human-preference-related metrics across both generation settings.

B. Effect of Structure Segment Learning (SSL) Scale. We investigate how the corpus scale used for Structure Segment Learning affects downstream SVG generation. Specifically, we learn structure segment merges from three SVG

Table 5: Effect of three-stage curriculum training on Image-to-SVG with stage-to-stage changes. ↑ higher is better, ↓ lower is better. Delta rows show changes from previous stage (positive values indicate improvement for ↑ metrics).

	Validity / Efficiency				Visual Similarity			Preference / Aesthetic			
Method	Render↑	TokCnt	PathCnt	CmdCnt	SSIM↑	LPIPS↓	CLIP-S↑	ImgR↑	HPS↑	PickS↑	Aes↑
Stage1-L1	95.20%	273	3.5	54.8	0.8052 ± 0.13	0.1716 ± 0.09	0.9360 ± 0.07	-0.0764	0.2110	21.348	4.6159
Stage1-L2	94.20%	489	5.9	107.2	0.7223 ± 0.13	0.2392 ± 0.09	0.8884 ± 0.10	-0.1836	0.2026	20.945	4.6175
Stage1-L3	90.09%	656	6.7	144.2	0.7015 ± 0.13	0.2405 ± 0.10	0.8750 ± 0.11	-0.2500	0.2050	21.015	4.6285
ΔS1→S2	*+0.2%*	*+27.0%*	*+15.3%*	*+19.5%*	*+4.1%*	*-19.7%*	*+4.9%*	*+151.7%*	*+5.9%*	*+21.0%*	*+1.3%*
Stage2-L1	93.50%	335	4.8	70.4	0.8152 ± 0.12	0.1611 ± 0.08	0.9540 ± 0.06	0.0294	0.2150	21.522	4.6226
Stage2-L2	94.40%	621	6.8	128.1	0.7520 ± 0.12	0.1920 ± 0.08	0.9320 ± 0.08	0.0950	0.2145	21.385	4.6780
Stage2-L3	90.29%	897	9.6	181.5	0.7180 ± 0.13	0.2185 ± 0.09	0.9210 ± 0.09	0.0520	0.2105	21.185	4.7150
ΔS2→S3	*-2.9%*	*+26.3%*	*+16.7%*	*+27.9%*	*-2.2%*	*+7.7%*	*+2.7%*	*+186.5%*	*+1.5%*	*+8.4%*	*+0.7%*
Stage3-L1	94.60%	376	6.3	85.9	0.8129 ± 0.12	0.1611 ± 0.08	0.9581 ± 0.06	0.0742	0.2166	21.606	4.6200
Stage3-L2	92.40%	825	9.7	179.9	0.7352 ± 0.12	0.2097 ± 0.08	0.9471 ± 0.07	0.1603	0.2163	21.475	4.7050
Stage3-L3	87.69%	1133	11.2	232.1	0.7019 ± 0.13	0.2353 ± 0.08	0.9454 ± 0.07	0.1490	0.2136	21.362	4.7501

Table 6: Effect of three-stage curriculum training on Text-to-SVG with stage-to-stage changes. ↑ indicates higher is better, ↓ indicates lower is better. Delta rows show changes from previous stage (positive values indicate improvement for ↑ metrics).

	Validity / Efficiency				Semantic	Diversity	Preference / Aesthetic			
Method	Render↑	TokCnt↓	PathCnt↓	CmdCnt↓	CLIP↑	DINO↑	ImgR↑	HPS↑	PickS↑	Aes↑
Stage1-L1	95.60%	288	3.1	60.9	0.2346	0.2949	-0.4789	0.1909	20.552	4.5891
Stage1-L2	95.37%	403	4.3	89.7	0.2322	0.3535	-0.7787	0.1751	20.046	4.5663
Stage1-L3	94.08%	446	5.3	104.9	0.2305	0.4015	-0.8520	0.1730	19.895	4.5580
ΔS1→S2	*-0.8%*	*+58.1%*	*+37.2%*	*+52.4%*	*+0.6%*	*-1.3%*	*+19.3%*	*+5.1%*	*+1.2%*	*+0.6%*
Stage2-L1	95.45%	428	4.3	87.6	0.2356	0.2931	-0.3768	0.1953	20.643	4.6279
Stage2-L2	94.62%	637	5.9	136.7	0.2335	0.3490	-0.6285	0.1840	20.285	4.5920
Stage2-L3	93.69%	717	7.7	160.6	0.2320	0.3930	-0.7180	0.1785	20.115	4.5850
ΔS2→S3	*-3.5%*	*+53.3%*	*+27.3%*	*+49.4%*	*+0.6%*	*-1.8%*	*+2.9%*	*-0.3%*	*-1.3%*	*+0.7%*
Stage3-L1	94.53%	532	5.8	112.2	0.2356	0.3032	-0.3776	0.1954	20.672	4.6331
Stage3-L2	91.78%	943	8.5	202.5	0.2345	0.3450	-0.5380	0.1865	20.345	4.6095
Stage3-L3	90.41%	1099	9.8	239.9	0.2335	0.3859	-0.6975	0.1779	20.021	4.6164

corpora of increasing sizes: 50k, 500k, and 1.5M samples. For each scale, the resulting segment tokenizer is applied to construct Segment Tokens from Atomic Tokens for both model training and inference, while keeping all other components unchanged. This study examines whether larger corpora enable more reliable discovery of frequent and structurally valid path segments, leading to improved token composition efficiency, sequence compactness, and geometric fidelity.

C. Effect of Token Initialization Strategy. To isolate the contribution of each component in Eq. 4, we design seven initialization variants with increasing structural priors, summarized in Table 4. All experiments use identical training configurations: Qwen2.5-VL-3B as the backbone, full-parameter SFT with frozen vision encoder and projector, learning rate 10^{-5} with the same warmup setting, and 1 epoch on a mixed dataset of image-to-SVG, image-to-caption, and text-to-SVG tasks (∼3,000 domain-specific tokens). **D. Impact of Three-Stage Curriculum Training.** To investigate the effect of curriculum learning on

SVG generation, we adopt a three-stage training paradigm based on sequence length. Specifically, the training corpus is partitioned according to SVG token length into three complexity levels: **Stage-1** (30∼326 tokens), **Stage-2** (326∼605 tokens), and **Stage-3** (605∼1k tokens). The model is progressively trained from shorter to longer sequences. To evaluate generalization across complexity levels, we construct three test subsets corresponding to these ranges, denoted as **L1**, **L2**, and **L3**. This design enables fine-grained analysis of how curriculum learning affects performance on simple versus complex SVG structures. This suggests that gradually increasing program depth stabilizes token embedding learning and improves long-range geometric reasoning.

We observe that curriculum training consistently improves performance on longer sequences (L2/L3) without sacrificing accuracy on simpler cases (L1), suggesting that progressive exposure to structural complexity stabilizes optimization and enhances generalization to high-token-length SVG programs.

E. Path-Level Structural Noise and Segment Properties Discovered by SSL. We further analyze the types of path-level structural noise discovered during Structure Segment Learning (SSL), as well as the geometric properties of the extracted segments. Figure 10 summarizes the cleaning statistics, redundancy motifs, and segment characteristics across different corpus scales.

As shown in Figure 10 (a), the amount of command-level cleaning remains stable across corpus scales (∼0.86 removed commands per sample), indicating that the underlying noise patterns are largely data-inherent rather than scale-dependent. Most of these removals are concentrated in line-related commands (`l`, `h`) and cubic curves (`c`), while `v` and `s` contribute marginally. Figure 10 (b) details the frequency mass of strict degenerate motifs. Although segments containing `<d_0>` are common (47.8%–51.3%), strictly degenerate patterns remain rare across all scales. Specifically, consecutive `<d_0><d_0>` pairs account for only 0.51%–0.63%, zero-move commands for 0.21%–0.22%, and degenerate arcs (`rx=ry=0`) for 0.21%–0.22%.

Beyond filtering noise, SSL effectively captures meaningful and compact geometric primitives. Figure 10 (c) illustrates the command types distribution within the learned segments across different frequency buckets (Top 50, 51–200, and 201–500). Cubic Bézier curves (`c`) are the dominant components, particularly in mid-frequency segments where they account for 40% of the share, followed by strong representations of arcs (`a`, peaking at 24% in the Top 50 bucket) and smooth curves (`s`, 22% in Top 50 and 51–200 buckets).

Furthermore, Figure 10 (d) demonstrates that the atomic token length of these learned segments is highly consistent. The median length stays robust at roughly 9 tokens regardless of the segment's overall frequency, though lower-frequency segments display slightly more variance and longer outliers. These results suggest that SSL successfully identifies compact, reusable, and structurally diverse geometric segments while effectively filtering a small set of redundant path structures.

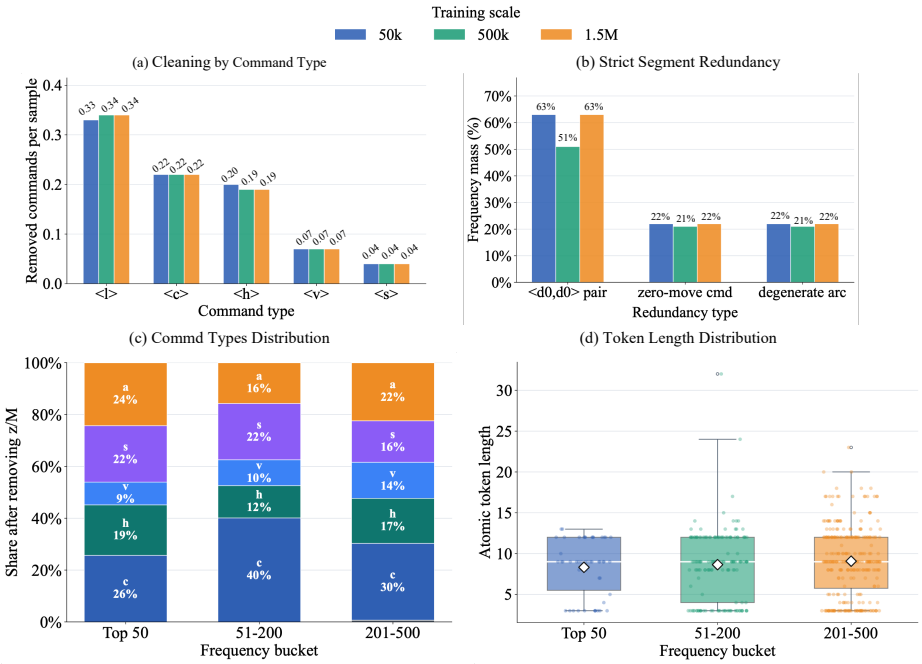

Fig. 10: Path cleaning statistics, degenerate segment patterns, and learned segment properties discovered by Structure Segment Learning (SSL). (a) Command-level cleaning statistics measured as removed commands per sample across data scales (50k, 500k, and 1.5M). The cleaning rate remains stable across different scales, primarily concentrated in specific command types. (b) Frequency mass of strict degenerate motifs in learned segments. Strictly degenerate patterns remain rare across all data scales. (c) Command type distribution within learned segments across frequency buckets. Cubic Bezier curves (`c`) and other complex commands consistently constitute significant portions. (d) Atomic token length distribution of the segments. The median token length remains highly stable across frequency buckets, with varying distributions of outliers in lower-frequency segments.

5 Conclusion

We introduced a hierarchical tokenization framework for scalable SVG generation. By redefining the representation unit from character-level fragments to executable geometric segments, the proposed approach aligns token structure with the semantics of vector graphics. The hierarchical design reduces sequence length while preserving structural validity, enabling more stable autoregressive modeling. Together with structured initialization and scalable training, the framework demonstrates that representation design plays a crucial role in reliable SVG generation. Our results suggest that improving geometric consistency does not rely solely on increasing model scale. Instead, aligning tokenization with executable structure provides a principled foundation for vector graphics modeling. Future

work may extend this framework to other structured graphical formats and explore integration with differentiable rendering objectives.

6 Contributors

Contributors: Ximing Xing*, Ziteng Xue, Zhenxi Li, Weicong Liang, Linqing Wang, Zhantao Yang, Tiankai Hang, Zijin Yin, Qinglin Lu, Chunyu Wang‡, Qian Yu‡

‡ **Corresponding Author**: Qian Yu & Chunyu Wang

Supplementary Material

Contents

A Dataset Construction, Filtering & Preprocessing

Our training corpus is built by merging three open-source SVG datasets: SVG-Stack [24] (2,283,875 samples), SVGX-Dataset [37] (257,086 samples), and MMSVG-Icon [43] (1,159,423 samples). After cross-source merging and deduplication, the resulting corpus contains 2,445,092 unique SVG samples. The merged dataset covers a broad range of vector graphic categories, including icons, emojis, logos, interface elements, and other structured graphic designs.

To improve rendering consistency and reduce malformed or non-executable samples, we apply a unified preprocessing pipeline prior to tokenization. The pipeline consists of three stages: data cleaning, coordinate transformation, and coordinate quantization.

Data cleaning. We first parse each SVG and remove unsupported or unsafe elements. Specifically, non-renderable or undesirable tags such as `<foreignObject>` are filtered out, while external-content or executable elements, including `<image>` and `<script>`, are rejected. At the root level, we remove redundant SVG attributes and retain only the `viewBox` as the canonical geometric reference. We further inline CSS style rules into element attributes, normalize the SVG structure through a pure-Python preprocessing pipeline, remove unnecessary line breaks, convert color specifications into compact hexadecimal form, and repair missing `fill` values when necessary.

Coordinate transformation. After structural cleaning, all SVGs are mapped into a unified geometric space. We first expand `<use>` references by inlining reused elements, ensuring that subsequent transformations operate on explicit geometry only. For selected light-color SVGs, a dark background may be added to improve rendering visibility. We then bake all `transform` attributes directly

Fig. 1: Text-to-SVG generation results. For each example, we show the text prompt together with the rendered SVG output generated by HiVG.

into coordinates, eliminating residual transformation matrices from the final representation. Finally, we normalize the `viewBox` by translating its origin to $(0, 0)$ and rescaling the canvas to a target resolution of 784×784.

Coordinate quantization. After global scaling, all coordinates are quantized by rounding to integers. Absolute coordinates are then converted into relative coordinates to better match the sequential geometric representation used by our tokenizer. As a final compatibility step, we clip out-of-bound subpaths and clamp minor numerical overflow within a tolerance range of ± 10, which improves tokenizer robustness in borderline cases.

Several implementation choices are important in practice. First, `transform` baking is performed only after `<use>` expansion, preventing duplicated geometric transformations. Second, coordinate quantization is applied after global scaling to minimize unnecessary precision loss. Third, boundary correction is deferred to the final stage so that the processed SVGs remain compatible with downstream tokenization and decoding. Samples that still cannot be parsed, normalized, or rendered stably after preprocessing are discarded.

A.1 SVG Code Usability Review

Motivation and Protocol. Raster-domain metrics cannot assess whether a generated SVG remains structurally meaningful and editable after being im-

Fig. 2: Image-to-SVG generation results. For each example, the raster input image is shown on the right and the generated SVG rendering on the left.

ported into professional vector-graphics software. We therefore conduct an additional expert review in Adobe Illustrator [1]. The same 8 professional SVG practitioners import the generated SVGs and evaluate their structural usability. Specifically, they examine whether primitives and path groups correspond to coherent visual-semantic parts, whether local components can be selected and edited conveniently, and whether the SVG contains excessive redundant fragments or implausible decomposition.

Each SVG is scored on a 1–5 Likert scale along four dimensions: *semantic layering*, *editability*, *redundancy control*, and *overall code usability*. Because

[1] We use Adobe Illustrator as a representative industry-standard vector graphics editor for assessing practical SVG editability.

Table 1: Expert review of SVG code usability in Adobe Illustrator. Eight professional SVG practitioners import the generated SVGs into Adobe Illustrator and score their structural usability on a 1–5 Likert scale. Higher is better for all metrics.

Method	Semantic Layering ↑	Editability ↑	Redundancy Control ↑	Overall Code Usability ↑
SVGen-7B [31]	2.88	2.83	2.74	2.82
InternSVG-8B [32]	3.22	3.18	3.09	3.16
Gemini-2.5-pro [4]	3.39	3.34	3.23	3.32
GPT-5.2 [18]	3.56	3.49	3.37	3.47
HiVG-3B	**4.11**	**4.05**	**3.96**	**4.06**

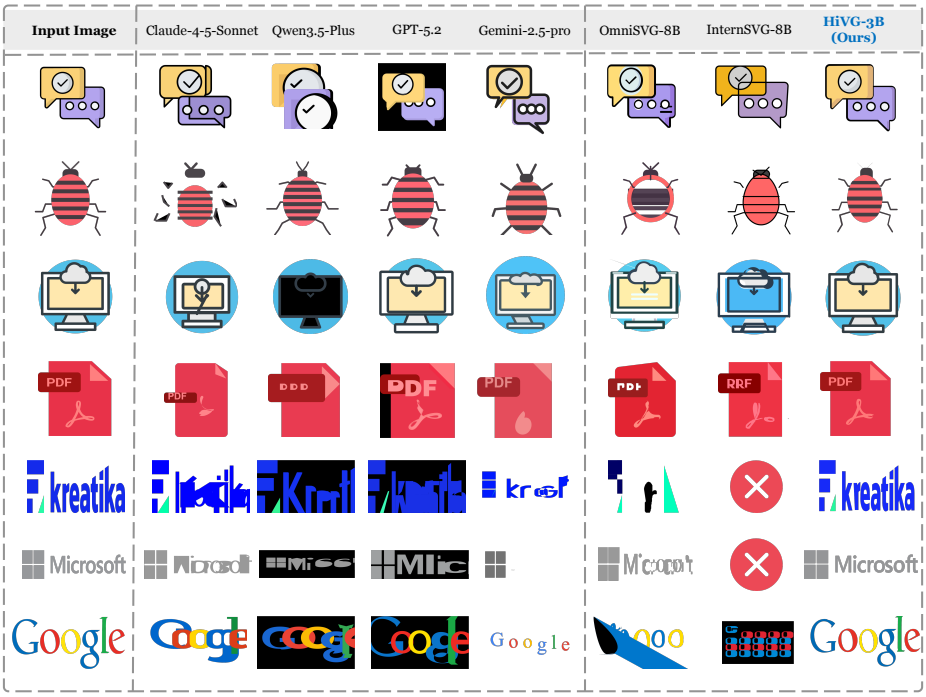

Fig. 3: Additional Image-to-SVG comparison.

this review is substantially more time-consuming than raster-only inspection, we evaluate five representative methods: SVGen-7B, InternSVG-8B, Gemini-2.5-pro, GPT-5.2, and HiVG-3B.

Results. Table 1 reports the Illustrator-based usability review. HiVG-3B achieves the best scores on all four dimensions, with the clearest gains in semantic layering and editability. These results suggest that HiVG improves not only rendered reconstruction quality, but also the structural organization of SVG code in a way that better matches human editing workflows.

Summary. Together, the two protocols provide a compact but more complete assessment of image-to-SVG reconstruction. Pairwise comparison measures what human experts prefer, while Illustrator-based review evaluates structural usability beyond automatic metrics. Across both settings, HiVG-3B shows consistent

Fig. 4: Additional Text-to-SVG comparison.

advantages, indicating that its improvements extend from raster-domain reconstruction to the practical usability of generated SVG code.

B Extended Results

B.1 More Text-to-SVG and Image-to-SVG Results

We provide additional text-to-SVG and image-to-SVG generation examples covering diverse prompts, including flat icons, stylized symbols, logos, and multi-part graphic compositions. The results in Figure 7 complement the main paper by illustrating how HiVG handles varying semantic granularity, object composition, and layout structure under open-ended textual descriptions. Compared with the limited examples shown in the main paper, the additional results in Figure 6 provide a broader view of the model's reconstruction behavior across different levels of geometric complexity.

B.2 Comparison with Existing Methods

We provide more text-to-svg and image-to-svg comparison results with existing methods in Figure 3 & 4 respectively. As visually demonstrated, our method exhibits exceptional proficiency in generating SVGs that contain typographical elements and letters. This success clearly reflects our method's advanced capacity for precise geometric generation and complex topology preservation. Furthermore, achieving such high-quality typography generation with an exceptionally lightweight 3B-parameter model underscores its remarkable efficiency.

C Extended Implementation Details

C.1 Initialization Details & Hyperparameters

The main paper introduces Hierarchical Mean-Noise (HMN) initialization to stably incorporate newly introduced structured SVG tokens into the pretrained

Table 2: Hyperparameter settings of HiVG.

Hyperparameter	HiVG
Architecture / Tokenization	
Backbone model	Qwen2.5-VL-3B-Instruct
Canvas size	784×784
Atomic token range	30–1000
Number of curriculum stages	3
Context length scaling	progressive across stages
Atomic vocabulary size	2450
Segment vocabulary size	500
Coordinate quantization bins	-794 $\sim$ 794
Optimization	
Optimizer	AdamW
Learning rate	1e-5
Weight decay	0.2
Warmup ratio	0.1
Global batch size	128
Training epochs	2
Max context length (S1 / S2 / S3)	1792 / 2176 / 2432
HMN Initialization	
Mean anchor weight λ_μ	0.8
Noise scale λ_n	0.02
Semantic prior weight w_{sem}	0.1
Numeric prior weight w_{num}	0.08
Number of RBF bases K	16
Numeric projection matrix	fixed random
Inference	
Decoding strategy	autoregressive
Temperature	0.7
Top-p	0.9
Top-K	50
Repetition penalty	1.0
Evaluation rendering resolution	512×512

language model vocabulary. Here we provide additional implementation details together with the main hyperparameter settings used in training and inference. Table 2 summarizes the overall configuration, while the discussion below focuses on the initialization design.

In our implementation, the weighting coefficients are set to $\lambda_\mu = [0.8]$, $\lambda_n = [0.02]$, $w_{\text{sem}} = [0.1]$, and $w_{\text{num}} = [0.08]$. For the numeric projection branch, each normalized scalar value $v_t \in [0, 1]$ is expanded using $K = [16]$ Gaussian radial basis functions together with low-order polynomial features, and the resulting vector is projected to the model embedding dimension using a fixed random projection matrix. This design improves local continuity among coordinate tokens and stabilizes early-stage optimization when the model begins to learn structured SVG geometry.

C.2 Training and Inference Prompt Templates

We use unified instruction-style prompts for all training and evaluation settings. For text-to-SVG generation, the model is prompted to produce SVG code directly from a textual description:

Text-to-SVG Token Generation (T2ST)

```
<|im_start|>user
[T2ST] Generate SVG tokens from text. {text description}
<|im_end|>
<|im_start|>assistant
<svg>viewBox=0 0 784 784<cmd_M><P_242><P_674><cmd_c><d_7><d_-195>
...</path></svg>
<|im_end|>
```

Image-to-SVG Token Generation (I2ST)

```
<|im_start|>user
[I2ST] <image>. Generate SVG tokens from image.
<|im_end|>
<|im_start|>assistant
<svg>viewBox=0 0 784 784<cmd_M><P_283><P_567><cmd_c><d_0><d_0>
...</path></svg>
<|im_end|>
```

Fig. 5: Training templates for text- and image-conditioned SVG token generation. For text-to-SVG Token generation (T2ST), the model takes a textual description as input and autoregressively predicts hierarchical SVG tokens. For image-to-SVG Token generation (I2ST), an image placeholder (`<image>`) is inserted into the user turn, and the model outputs the corresponding SVG token sequence under the same conversational format. Using unified templates across both tasks simplifies multi-task training and keeps the supervision interface consistent.

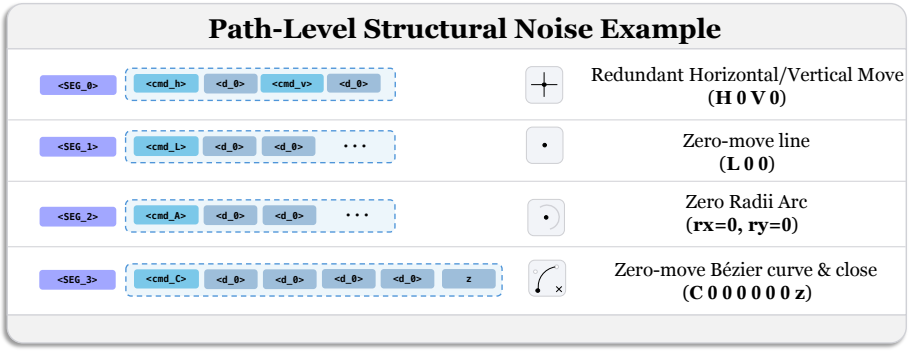

Fig. 6: These redundant or zero-move command patterns act as structural noise, artificially inflating token length and computational cost without contributing to the final visual representation.

For image-to-SVG reconstruction, the image token is prepended to the same instruction template, and the model is asked to reconstruct a valid SVG program that faithfully matches the input image. During evaluation, we use fixed prompt templates across all methods whenever possible, together with unified rendering and post-processing rules, to reduce prompt-induced variance in downstream comparisons.

D Additional Analysis of Structured Tokens

D.1 Path-Level Structural Noise Patterns

To better understand what SSL learns from large-scale SVG corpora, we first analyze the path-level structural noise that appears before segment token construction. In practice, raw SVG paths often contain redundant command patterns, near-degenerate line fragments, repeated coordinate sequences, or overly fragmented local geometry introduced by upstream authoring tools and conversion pipelines. Such noise is not always visually obvious after rasterization, but it inflates sequence length and weakens the consistency of reusable segment extraction.

Our cleaning and segmentation pipeline reveals that these irregularities are concentrated in a limited set of recurring path motifs, especially redundant short line transitions, repeated local offsets, and command groups that do not contribute meaningful geometry (see Figure 6). This observation supports the design of SSL: rather than compressing arbitrary text spans, the tokenizer should operate on executable geometric units and suppress unstable path fragments that do not reflect reusable structure.

References

1. Anthropic: Claude 4.5 model card (2025), https://www-cdn.anthropic.com/claude-4-5-model-card.pdf, accessed: 2026-03-04
2. Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al.: Qwen2. 5-vl technical report. arXiv preprint arXiv:2502.13923 (2025)
3. Carlier, A., Danelljan, M., Alahi, A., Timofte, R.: Deepsvg: A hierarchical generative network for vector graphics animation. Advances in Neural Information Processing Systems **33**, 16351–16361 (2020)
4. Comanici, G., Bieber, E., Schaekermann, M., Pasupat, I., Sachdeva, N., Dhillon, I., Blistein, M., Ram, O., Zhang, D., Rosen, E., et al.: Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261 (2025)
5. DeepSeek-AI: Deepseek-v3.2: Pushing the frontier of open large language models (2025), https://arxiv.org/abs/2512.02556
6. Delétang, G., Ruoss, A., Duquenne, P.A., Catt, E., Genewein, T., Mattern, C., Grau-Moya, J., Wenliang, L.K., Aitchison, M., Orseau, L., et al.: Language modeling is compression. arXiv preprint arXiv:2309.10668 (2023)
7. Frans, K., Soros, L., Witkowski, O.: CLIPDraw: Exploring text-to-drawing synthesis through language-image encoders. In: Advances in Neural Information Processing Systems (NeurIPS) (2022)
8. Ghojogh, B., Ghodsi, A., Karray, F., Crowley, M.: Johnson-lindenstrauss lemma, linear and nonlinear random projections, random fourier features, and random kitchen sinks: Tutorial and survey. arXiv preprint arXiv:2108.04172 (2021)
9. Google DeepMind: Gemini 3 Pro model card. https://deepmind.google/models/gemini/pro/ (2025), Model Card, December 2025
10. Huang, Q., Dong, X., Zhang, P., Wang, B., He, C., Wang, J., Lin, D., Zhang, W., Yu, N.: Opera: Alleviating hallucination in multi-modal large language models via over-trust penalty and retrospection-allocation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13418–13427 (2024)
11. Huang, Y., Zhang, J., Shan, Z., He, J.: Compression represents intelligence linearly. In: First Conference on Language Modeling (2024), https://openreview.net/forum?id=SHMj84U5SH
12. Jain, A., Xie, A., Abbeel, P.: Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1911–1920 (2023)
13. Kirstain, Y., Polyak, A., Singer, U., Matiana, S., Penna, J., Levy, O.: Pick-a-pic: An open dataset of user preferences for text-to-image generation. Advances in neural information processing systems **36**, 36652–36663 (2023)
14. Kudo, T., Richardson, J.: Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In: Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations. pp. 66–71 (2018)
15. Li, J., Yu, J., Wei, C., Dong, H., Lin, Q., Yang, L., Wang, Z., Hao, Y.: 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)
16. Li, T.M., Lukáč, M., Gharbi, M., Ragan-Kelley, J.: Differentiable vector graphics rasterization for editing and learning. ACM Transactions on Graphics (TOG) **39**(6), 1–15 (2020)

17. Liu, J., Weng, H., Lei, B., Yang, X., Zhao, Z., Chen, Z., Guo, S., Han, T., Guo, C.: Freemesh: Boosting mesh generation with coordinates merging. In: Forty-second International Conference on Machine Learning (2025), https://openreview.net/forum?id=LBE7HKLQDa
18. OpenAI: Update to GPT-5 System Card: GPT-5.2. https://openai.com/index/introducing-gpt-5-2/ (2025), System Card, December 2025
19. Oquab, M., Darcet, T., Moutakanni, T., Vo, H.V., Szafraniec, M., Khalidov, V., Fernandez, P., HAZIZA, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P.Y., Li, S.W., Misra, I., Rabbat, M., Sharma, V., Synnaeve, G., Xu, H., Jegou, H., Mairal, J., Labatut, P., Joulin, A., Bojanowski, P.: DINOv2: Learning robust visual features without supervision. Transactions on Machine Learning Research (TMLR) (2024), https://openreview.net/forum?id=a68SUt6zFt
20. Pertsch, K., Stachowicz, K., Ichter, B., Driess, D., Nair, S., Vuong, Q., Mees, O., Finn, C., Levine, S.: Fast: Efficient action tokenization for vision-language-action models. arXiv preprint arXiv:2501.09747 (2025)
21. Qwen Team: Qwen3.5: Towards native multimodal agents (February 2026), https://qwen.ai/blog?id=qwen3.5
22. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International Conference on Machine Learning. pp. 8748–8763. PMLR (2021)
23. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Proceedings of the 21st International Conference on Neural Information Processing Systems. p. 1177–1184. NIPS'07, Curran Associates Inc., Red Hook, NY, USA (2007)
24. Rodriguez, J.A., Puri, A., Agarwal, S., Laradji, I.H., Rodriguez, P., Rajeswar, S., Vazquez, D., Pal, C., Pedersoli, M.: Starvector: Generating scalable vector graphics code from images and text. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 16175–16186 (2025)
25. Rodriguez, J.A., Zhang, H., Puri, A., Feizi, A., Pramanik, R., Wichmann, P., Mondal, A., Samsami, M.R., Awal, R., Taslakian, P., et al.: Rendering-aware reinforcement learning for vector graphics generation. arXiv preprint arXiv:2505.20793 (2025)
26. Schuhmann, C.: Improved aesthetic predictor. https://github.com/christophschuhmann/improved-aesthetic-predictor (2022)
27. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. In: Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers). pp. 1715–1725 (2016)
28. Tang, Z., Wu, C., Zhang, Z., Ni, M., Yin, S., Liu, Y., Yang, Z., Wang, L., Liu, Z., Li, J., Duan, N.: Strokenuwa: tokenizing strokes for vector graphic synthesis. In: Proceedings of the 41st International Conference on Machine Learning. ICML'24, JMLR.org (2024)
29. Tian, Y., Ha, D.: Modern evolution strategies for creativity: Fitting concrete images and abstract concepts. In: Artificial Intelligence in Music, Sound, Art and Design. pp. 275–291. Springer (2022)
30. Vinker, Y., Pajouheshgar, E., Bo, J.Y., Bachmann, R.C., Bermano, A.H., Cohen-Or, D., Zamir, A., Shamir, A.: Clipasso: Semantically-aware object sketching. ACM Transactions on Graphics (TOG) **41**(4), 1–11 (2022)

31. Wang, F., Zhao, Z., Liu, Y., Zhang, D., Gao, J., Sun, H., Li, X.: Svgen: Interpretable vector graphics generation with large language models. In: Proceedings of the 33rd ACM International Conference on Multimedia. pp. 9608–9617 (2025)
32. Wang, H., Yin, J., Wei, Q., Zeng, W., Gu, L., Ye, S., Gao, Z., Wang, Y., Zhang, Y., Li, Y., et al.: Internsvg: Towards unified svg tasks with multimodal large language models. arXiv preprint arXiv:2510.11341 (2025)
33. Wang, S., Chen, C., Le, X., Xu, Q., Xu, L., Zhang, Y., Yang, J.: Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, pp. 7880–7888 (2025)
34. Wu, R., Su, W., Liao, J.: Chat2svg: Vector graphics generation with large language models and image diffusion models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 23690–23700 (2025)
35. Wu, X., Hao, Y., Sun, K., Chen, Y., Zhu, F., Zhao, R., Li, H.: Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. arXiv preprint arXiv:2306.09341 (2023)
36. Xing, X., Guan, Y., Zhang, J., Xu, D., Yu, Q.: Reason-svg: Hybrid reward rl for aha-moments in vector graphics generation. arXiv preprint arXiv:2505.24499 (2025)
37. Xing, X., Hu, J., Liang, G., Zhang, J., Xu, D., Yu, Q.: Empowering llms to understand and generate complex vector graphics. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 19487–19497 (2025)
38. Xing, X., Hu, J., Zhang, J., Xu, D., Yu, Q.: Svgfusion: Scalable text-to-svg generation via vector space diffusion. arXiv preprint arXiv:2412.10437 (2024)
39. Xing, X., Wang, C., Zhou, H., Zhang, J., Yu, Q., Xu, D.: Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. Advances in Neural Information Processing Systems **36**, 15869–15889 (2023)
40. Xing, X., Zhou, H., Wang, C., Zhang, J., Xu, D., Yu, Q.: Svgdreamer: Text guided svg generation with diffusion model. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4546–4555 (2024)
41. Xu, J., Liu, X., Wu, Y., Tong, Y., Li, Q., Ding, M., Tang, J., Dong, Y.: Imagereward: Learning and evaluating human preferences for text-to-image generation. Advances in Neural Information Processing Systems **36**, 15903–15935 (2023)
42. Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al.: Qwen2. 5 technical report. arXiv preprint arXiv:2412.15115 (2024)
43. Yang, Y., Cheng, W., Chen, S., Zeng, X., Yin, F., Zhang, J., Wang, L., Yu, G., Ma, X., Jiang, Y.G.: Omnisvg: A unified scalable vector graphics generation model. arXiv preprint arXiv:2504.06263 (2025)