# StrokeNUWA—Tokenizing Strokes for Vector Graphic Synthesis

**Zecheng Tang** [* 1 2] **Chenfei Wu** [* 2] **Zekai Zhang** [2] **Minheng Ni** [2] **Shengming Yin** [2] **Yu Liu** [2] **Zhengyuan Yang** [3]
**Lijuan Wang** [3] **Zicheng Liu** [3] **Juntao Li** [1] **Nan Duan** [2]

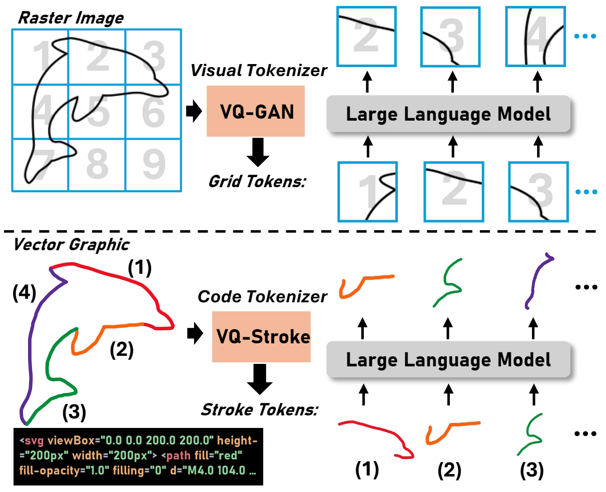
## Abstract

To leverage LLMs for visual synthesis, traditional methods convert raster image information into discrete grid tokens through specialized visual modules, while disrupting the model's ability to capture the true semantic representation of visual scenes. This paper posits that an alternative representation of images, vector graphics, can effectively surmount this limitation by enabling a more natural and semantically coherent segmentation of the image information. Thus, we introduce StrokeNUWA, a pioneering work exploring a better visual representation — "stroke tokens" on vector graphics, which is inherently visual semantics rich, naturally compatible with LLMs, and highly compressed. Equipped with stroke tokens, StrokeNUWA can significantly surpass traditional LLM-based and optimization-based methods across various metrics in the vector graphic generation task. Besides, StrokeNUWA achieves up to a 94× speedup in inference over the speed of prior methods with an exceptional SVG code compression ratio of 6.9%. Our code is available at `https://github.com/ProjectNUWA/StrokeNUWA`.

## 1. Introduction

In recent years, Large transformer-based Language Models, commonly referred as LLMs, have made significant strides, particularly in the domain of Natural Language Processing (NLP) (Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023; Anil et al., 2023). Concurrently, LLMs are gradually expanding their capabilities to other modalities, such as audio (Ghosal et al., 2023), medical (Singhal et al., 2023) and robotics (Brohan et al., 2023).

Current methodologies (Reddy et al., 2021; Wu et al., 2022; Chang et al., 2022; Kondratyuk et al., 2023) enable LLMs to generate visual information by transforming the continuous visual pixels into to discrete grid tokens via specialized visual modules such as VQ-VAE (Van Den Oord et al., 2017) and VQ-GAN (Esser et al., 2021). Subsequently, these transformed grid tokens are processed by the LLM in a manner akin to textual word handling, which facilitates LLMs' generative modeling process. However, when compared with diffusion models (Rombach et al., 2022), LLMs still fall behind (Lee et al., 2022; Sun et al., 2023). The shortcomings of LLMs in visual tasks primarily arise from two reasons: First, the transformation process relies on specific visual modules, which inherently possess limitations. For instance, advanced visual modules like VQ-GAN (Esser et al., 2021) can lead to the generation of images with artifact (Yu et al., 2023); Second, the use of grid tokens can disrupt the visual semantics, as the grids are artificially designed and not inherently semantic-aware. This artificial discretization imposes constraints on the model's ability to capture the true semantic representation of visual scenes.



```
<svg viewBox="0.0 0.0 200.0 200.0" height-
="200px" width="200px"> <path fill="red"
fill-opacity="1.0" filling="0" d="M4.0 104.0 …
```

*Figure 1.* Comparison between the visual representation of "grid" token and our proposed "stroke" token. Instead of tokenizing pixels from raster images, we explore a novel visual representation by tokenizing codes, from another image format—Scalable Vector Graphic (SVG). "Stroke" tokens have the following advantages: (1) inherently contain visual semantics, (2) naturally compatible with LLMs, and (3) highly compressed.

[*]Equal contribution [1]Soochow University [2]Microsoft Research Asia [3]Microsoft Azure AI. Correspondence to: Nan Duan <nanduan@microsoft.com>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

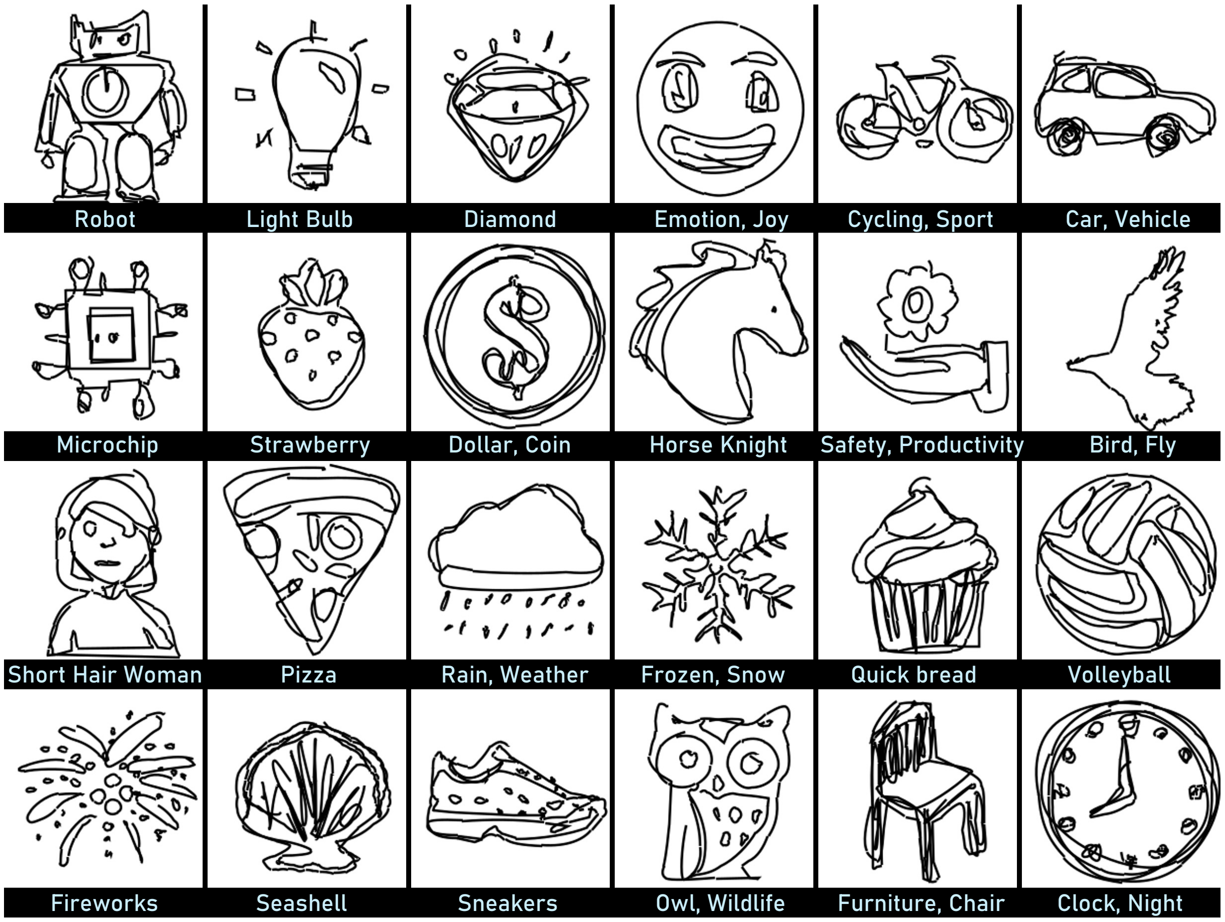


*Figure 2.* SVG generated by StrokeNUWA. For each image, we provide partial keywords for clarity.

*Is there a visual representation that preserves the semantic integrity of visual information while being conducive to processing by LLMs?* Finding such a representation within the framework of grid tokens is non-trivial, as the arrangement of grid tokens is typically regular and uniform, whereas the semantic structure within images is often irregular and complex. As illustrated in Fig. 1, the dolphin's body is arbitrarily segmented into different grid tokens. Although there have been efforts to improve the VQ-VAE method (Esser et al., 2021; Yu et al., 2023), enhancing the visual representation quality, they are fundamentally constrained by the limitations inherent to raster image formats, leading to bottlenecks in semantic preservation. In light of these challenges, we propose a novel approach that fundamentally retains the semantic concepts of images by utilizing an alternative image format: vector graphics. Different from pixel-based formats, vector graphics intrinsically reveal the construction of objects, naturally encapsulating the semantic concepts of the image. For example, our proposed "stroke" tokens segment the dolphin into sequentially connected strokes, where each stroke unit contains complete semantic information, such as the dolphin's fin (stroke ①) and back (stroke ②).

It is worth mentioning that our intention is not to claim that vector graphics are superior to raster images, but rather to introduce a fresh perspective on visual representation. The advantages of our "stroke" token concept include: (1) Inherently contains visual semantics: each stroke token intrinsically contains visual semantics, offering a more intuitive semantic segmentation of the image content; (2) Naturally compatible with LLMs: the creation process of vector graphics is naturally sequential and interconnected, which mirrors the way LLMs process information. In other words, Each stroke is created in relation to the ones before and after it, establishing a contiguous and coherent sequence that LLMs can process more naturally; (3) Highly compressed: strokes in vector graphics can be highly compressed, allowing each stroke token to encapsulate a rich, compressed representation of the visual information, significantly reducing the data size while maintaining quality and semantic integrity.

Based on the above analysis, we introduce StrokeNUWA, a model that crafts vector graphics without the reliance on the visual module. StrokeNUWA consists of a VQ-Stroke module and an Encoder-Decoder model. The VQ-Stroke, based on the residual quantizer model architecture (Martinez et al., 2014), can compress serialized vector graphic information into several SVG tokens. The Encoder-Decoder model primarily utilizes the capabilities of a pre-trained LLM to generate SVG tokens guided by text prompts.

We compare StrokeNUWA with optimization-based methods in the text-guided Scalable Vector Graphic (SVG) generation task. Our approach achieves higher CLIPScore (Hessel et al., 2021) metrics, suggesting that utilizing stroke tokens can yield content with richer visual semantics. When benchmarked against LLM-based baselines, our method surpasses them across all metrics, indicating that stroke tokens can integrate effectively with LLMs. Finally, due to the compression capabilities inherent in vector graphics, our model demonstrates significant efficiency in generation, achieving speed improvements of up to 94 times.

In a nutshell, our contributions can be outlined as follows:

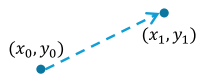
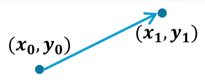
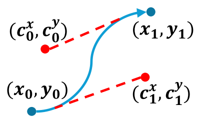
- We introduce StrokeNUWA, the pioneering study exploring a better visual representation—stroke token, to synthesize vector graphics solely through LLMs without relying on specialized visual modules.
- We propose VQ-Stroke, a specialized Vector Quantized Variational Autoencoder (VQ-VAE) designed to compress vector graphics into stroke tokens, providing an exceptional compression ratio of 6.9%.
- We conduct detailed experiments that demonstrate the significant potential of stroke tokens in the text-guided vector graphic synthesis task.

## 2. Related Work

### 2.1. Visual Representation

In the realm of computer graphics, two predominant image formats prevail: raster images, characterized by pixel matrices; and vector images, a.k.a, Scalable Vector Graphic (SVG), characterized by a series of code language commands (Zhang et al., 2023). Recent developments in visual synthesis have predominantly centered on generating raster images. The basic idea is to transform the continuous image pixels into discrete grid tokens via specialized visual modules such as VQ-VAE (Van Den Oord et al., 2017) and VQ-GAN (Esser et al., 2021), and then leverage LLMs to generate these tokens (Wu et al., 2022; Kondratyuk et al., 2023). Most recently, some works have tried to improve "grid" tokens by designing advanced architectures such as Lookup-Free Quantization (Yu et al., 2023) and Efficient VQ-GAN (Cao et al., 2023). However, these "grid" token representations can disrupt visual semantics as the grids are artificially designed, which lacks inherent semantic awareness, and are easily subject to the visual module's intrinsic limitations like disturbances and tampering (Hu et al., 2023). Our study is a pioneering effort to explore a better visual representation by proposing the concept of the "stroke" token.

*Table 1.* Overview of basic SVG commands, including `M`, `L`, and `C`, where each command contains one beginning point $(x_0, y_0)$ and one end point $(x_1, y_1)$. For Cubic Bézier command, it contains two extra control points $(c_0^x, c_0^y)$ and $(c_1^x, c_1^y)$.

| Name | Symbol | Argument | Example |
|---|---|---|---|
| Move To | `M` | $(x_0, y_0)$ $(x_1, y_1)$ |  |
| Line To | `L` | $(x_0, y_0)$ $(x_1, y_1)$ |  |
| Cubic Bézier | `C` | $(x_0, y_0)$ $(x_1, y_1)$ $(c_0^x, c_0^y)$ $(c_1^x, c_1^y)$ |   $(c_0^x, c_0^y)$<br>$(x_1, y_1)$<br>$(x_0, y_0)$<br>$(c_1^x, c_1^y)$ |

Different from the 'grid" tokens, the "stroke" token is inherently defined by contextually associated coded language commands that offer strong semantic integrity, potentially mitigating the aforementioned issues.

### 2.2. SVG Generation

SVG generation employs a method of structured code generation for producing graphics, which offers better interpretability, flexibility, and scalability in image representation. The current mainstream approach of SVG generation is optimization-based methods (Su et al., 2023; Jain et al., 2023; Xing et al., 2023), which share a similarity with traditional raster image generation, involving iteratively refining randomly initialized SVG paths to fit a target raster image with a differentiable rasterizer (Li et al., 2020). However, the optimization process is both time-consuming and computationally intensive, e.g., creating an SVG graphic comprised of 24 SVG paths can exceed 20 minutes[1]. Alternatively, some recent approaches have begun to adopt auto-regressive models to directly generate code for SVG synthesis (Wang et al., 2022; Wu et al., 2023a). However, due to the inherent extensive length nature of SVGs and a lack of effective SVG representation, these methods constrain LLMs to generate complex SVGs. To address these challenges, we introduce VQ-Stroke and present the concept of "stroke" tokens. By transforming SVGs into stroke tokens, our approach enables LLMs to produce intricate SVGs with significantly improved inference speed.

## 3. Methodology

### 3.1. Problem Formulation

SVG code provides a suite of command and syntax rules, e.g., the "`<rect>`" command defines a rectangle shape with its position, width, and height, which can

[1]We test with LIVE (Ma et al., 2022) and VectorFusion (Jain et al., 2023) on one NVIDIA V100 GPU.

be written as `<rect x="10" y="20" width="50" height="80"/>`. However, considering the multitude of SVG command types, creating such a system not only requires a complex data structure, but without a massive dataset, LLMs would struggle to model the diverse range of commands effectively. Therefore, as shown in Tab. 1, we can simplify each SVG using just three basic commands: “Move To”, “Line To”, and “Cubic Bézier” by following Iconshop (Wu et al., 2023a) and DeepSVG (Carlier et al., 2020). For instance, intricate commands like “`<rect>`” can be constructed by those three basic commands. After simplification, an SVG $\mathcal{G} = \{\mathcal{P}_i\}_{i=1}^{N}$ can be described with $N$ SVG paths, with each SVG path $\mathcal{P}_i$ consists of $M_i$ basic commands: $\mathcal{P}_i = \{\mathcal{C}_i^j\}_{j=1}^{M_i}$, where $\mathcal{C}_i^j$ is the $j$-th command in the $i$-th path. Eventually, each basic command $\mathcal{C} = (T, \mathcal{V})$ is consist of command type $T \in \{\texttt{M}, \texttt{L}, \texttt{C}\}$, and the corresponding position argument $\mathcal{V}$.

## 3.2. StrokeNUWA

StrokeNUWA contains three core components: a Vector Quantized-Stroke (VQ-Stroke) for SVG compression, an Encoder-Decoder-based LLM (EDM) for SVG generation, and an SVG Fixer (SF) for post-processing. Firstly, VQ-Stroke compresses the SVG into stroke tokens, which enables a transformation between the SVG code and the discrete stroke tokens. Then, EDM utilizes the stroke tokens produced from VQ-Stroke to generate SVG code. Finally, SF is a post-processing module designed to refine the quality of the generated SVGs, given that the output generated from the EDM or VQ-Stroke may not always conform to the stringent syntactical rules of SVG code. Below, we will introduce the details of each component.

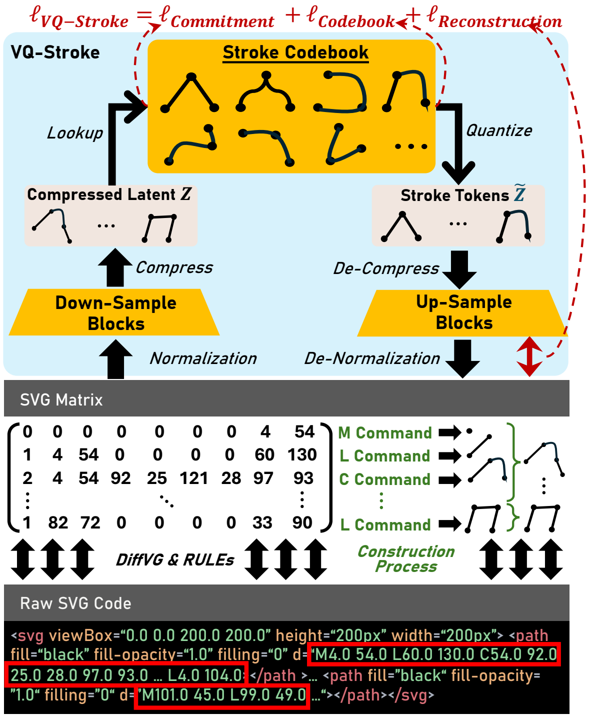
### 3.2.1. Vector Quantized-Stroke

VQ-Stroke encompasses two main stages: “Code to Matrix” stage that transforms SVG code into the matrix format suitable for model input, and “Matrix to Token” stage that transforms the matrix data into stroke tokens.

**Code to Matrix** As depicted in Fig. 3, we first transform the simplified SVG code (Sec. 3.1) into SVG matrix format by converting each basic command $\mathcal{C}_i^j$ to the individual vector $\mathcal{K}_i^j \in \mathbb{R}^9$ with rules $f$:

$$\mathcal{K}_i^j = f(\mathcal{C}_i^j) = (T, x_0, y_0, c_0^x, c_0^y, c_1^x, c_1^y, x_1, y_1)_i^j, \quad (1)$$

where $T$ denotes the basic command type, $(x_0, y_0)$ and $(x_1, y_1)$ represent the beginning and the end points, with $(c_0^x, c_0^y)$ and $(c_1^x, c_1^y)$ as the control points of each basic command. Then, to establish interconnections among the adjacent commands, we set the end point of $j$-th command $(x_1, y_1)_i^j$ equal to the beginning point $(x_0, y_0)_i^{j+1}$ of the subsequent $(j+1)$-th command in each individual path.



```
<svg viewBox="0.0 0.0 200.0 200.0" height="200px" width="200px"> <path fill="black" fill-opacity="1.0" filling="0" d="M4.0 54.0 L60.0 130.0 C54.0 92.0 25.0 28.0 97.0 93.0 … L4.0 104.0"</path >… <path fill="black" fill-opacity="1.0" filling="0" d="M101.0 45.0 L99.0 49.0 …"></path></svg>
```

*Figure 3.* Overview of VQ-Stroke.

We then decompose all the paths within the SVG $\mathcal{G}$ into distinct basic commands and combine their corresponding vectors into a matrix form:

$$\begin{aligned} f(\mathcal{G}) &= (f(P_i))_{i=1}^{N} = \left(\left(f(\mathcal{C}_i^j)\right)_{j=1}^{M_i}\right)_{i=1}^{N} \\ &= \begin{pmatrix} (\mathcal{K}_1^1; & \mathcal{K}_1^2; & \cdots; & \mathcal{K}_1^{M_1}) \\ \vdots & \vdots & \ddots & \vdots \\ (\mathcal{K}_N^1; & \mathcal{K}_N^2; & \cdots; & \mathcal{K}_N^{M_N}) \end{pmatrix}, \end{aligned} \quad (2)$$

where “;” denotes the stack operation, and each matrix row represents an individual command. Thus, we can obtain a structured SVG matrix $f(\mathcal{G}) \in \mathbb{R}^{(\sum_{i=1}^{N} M_i) \times 9}$ to represent an SVG that contains $\sum_{i=1}^{N} M_i$ individual basic commands.

**Matrix to Stroke** After obtaining the SVG matrix $f(\mathcal{G})$, we aim to compress the matrix into discrete stroke tokens via latent representation, with which one can reconstruct the $f(\mathcal{G})$. As shown in Fig. 3, the VQ-Stroke model is composed of Down-Sample blocks, a Stroke Codebook $\mathcal{B}$, and Up-Sample blocks. The SVG matrix $f(\mathcal{G})$ is first encoded by the Down-Sample blocks to obtain the compressed representations, which entails increasing the number of representation channels (column of $f(\mathcal{G})$) while concurrently compressing the spatial dimensions (row of $f(\mathcal{G})$) to yield a more compact representation, i.e. compressing the number of commands into $T$ s.t. $T < \sum_{i=1}^{N} M_i$. Then, the Codebook $\mathcal{B}$ simultaneously conducts $d$ levels of compression with residual vector quantization (Barnes et al., 1996;

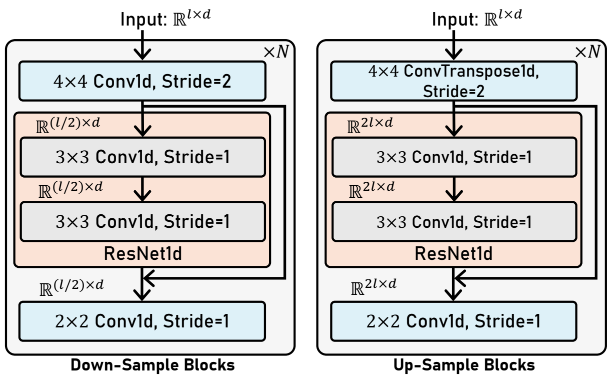


*Figure 4.* Architecture of Down-Sample and Up-Sample Blocks.

Wang et al., 2023), enabling VQ-Stroke to better model the compressed representations. We depict the detailed architecture of Down-Sample blocks and Up-Sample blocks in Fig 4, wherein both blocks first utilize a Conv1d or ConvTranspose1d model to compress or expand the features, succeeded by a ResNet1d module and an additional Conv1d module for feature extraction. It is worth mentioning that a low compression rate allows the VQ-Stroke to learn the fine details of SVGs (the first and second columns), while more aggressive compression (the third column) enables the VQ-Stroke to capture the overall contours of the SVGs, As illustrated in Fig.5, a low compression rate allows the VQ-Stroke to learn the fine details of SVGs (the first and second columns), while more aggressive compression (the third column) enables the VQ-Stroke to capture the overall contours of the SVGs. We have more discussion in Sec. 4.2. Finally, the Down-Sample blocks reconstruct the SVG latent representation output from the Codebook $\mathcal{B}$.

Following Dhariwal et al., the training objective of VQ-Stroke consists of commitment loss, codebook loss, and reconstruction loss, which can be written as:

$$\begin{aligned}\ell_{VQ-Stroke} &= \alpha\left(\ell_{codebook} + \ell_{commit}\right) + \ell_{recon}\\ &= \alpha\left(\|\, \mathcal{Z} - \mathrm{sg}[\tilde{\mathcal{Z}}]\,\|_2^2 + \|\,\mathrm{sg}[\mathcal{Z}] - \tilde{\mathcal{Z}}\,\|_2^2\right)\\ &+ \mathrm{MSE}(\widetilde{f(\mathcal{G})}, f(\mathcal{G})),\end{aligned} \tag{3}$$

where $\alpha$ is the hyper-parameter, $\mathcal{Z}$ is the compressed latent output from down-sample blocks, $\tilde{\mathcal{Z}}$ is the latent looked up from codebook $\mathcal{B}$, and $\mathrm{sg}[\cdot]$ is the gradient clipping operation. Besides, we pre-normalize the input data into the $[-1, 1]$ range to stabilize the training process.

### 3.2.2. Encoder-Decoder-based LLM

We employ an Encoder-Decoder LLM (EDM) to predict the stroke tokens obtained from the codebook $\mathcal{B}$. Considering LLM's inherent textual instruction capability, we freeze the EDM encoder to leverage its inherited textual knowledge. Subsequently, we fine-tune the EDM decoder

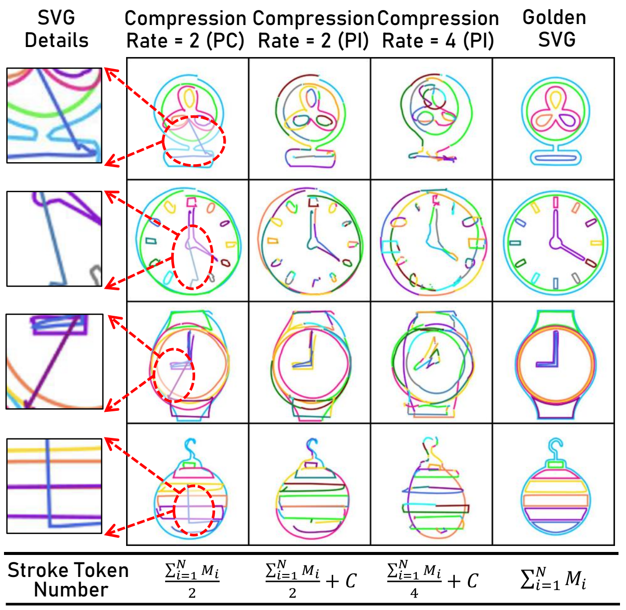

SVG Details
Compression Rate = 2 (PC)
Compression Rate = 2 (PI)
Compression Rate = 4 (PI)
Golden SVG
Stroke Token Number
$\frac{\sum_{i=1}^{N} M_i}{2}$
$\frac{\sum_{i=1}^{N} M_i}{2} + C$
$\frac{\sum_{i=1}^{N} M_i}{4} + C$
$\sum_{i=1}^{N} M_i$

*Figure 5.* Analysis of SVG reconstruction, where $C$ is a constant representing the number of inserted `<M>` command in PI setting. To facilitate clear observation of the SVG composition, we represent each basic command with a distinct color.

to learn the stroke token prediction task. Due to the discrepancy between the vocabulary of stroke tokens and the original LLM's vocabulary, we extend EDM with an additional stroke embedding layer and a stroke predictor. Consequently, given the trainable model parameters $\theta$ and the textual prompt $\boldsymbol{K}$, we maximize the log probability $\mathrm{argmax}_\theta \prod_{i=1}^{T} P(t_i \mid t_{<i}, \boldsymbol{K})$ with the cross-entropy loss.

### 3.2.3. SVG Fixer

A critical issue arises in the generation results from both SDM and EDM, as they fail to guarantee Equ. 1 due to the discrepancies of the interconnection points among adjacent commands in each individual SVG path, i.e., $(x_1, y_1)_i^j \neq (x_0, y_0)_i^{j+1}$ in $i$-th path. To address this issue, we introduce the SVG Fixer (SF) as a post-processing module for the generated results. It encompasses two strategies: Path Clipping (PC) and Path Interpolation (PI). Specifically, PC involves the direct substitution of each SVG command's beginning point with the endpoint of adjacent SVG commands: $(x_0, y_0)_i^{j+1} := (x_1, y_1)_i^j$. On the other hand, PI entails the addition of `M` commands between each pair of adjacent, yet non-interconnected SVG commands to bridge the discrepancy, i.e., if $(x_1, y_1)_i^j \neq (x_0, y_0)_i^{j+1} \implies$ adding an extra command $\left(\mathtt{M}, (x_1, y_1)_i^j, 0, 0, 0, 0, (x_0, y_0)_i^{j+1}\right)$ to force the previous command's end point to move to the beginning point of the next adjacent command. As shown in Fig. 5, PC can streamline the overall paths of SVGs, making them more succinct, but may lead to some inaccuracies in the details. On the other hand, PI tends to reveal more generated stokes' details, but it may introduce more curves. Each strategy has its own applicable scenarios.

## 4. Experiment

### 4.1. Experimental Settings

**Dataset** We construct the training and evaluation data with FIGR-8-SVG dataset (Clouâtre & Demers, 2019), which consists of massive monochromatic (black-and-white) SVG icons. We pre-process the SVG data by transforming each SVG sample into standardized representations, eliminating the redundant SVG paths, dropping the outer black box, and filtering the data by applying a threshold of 1,024 basic commands in length. We filter the instance with less than two annotated discrete keywords and apply a template "`Generating SVG according to keywords:`{$\cdots$}" to build the text prompt. After preprocessing, we sample 2,000 instances with varying SVG code lengths as the testing set, 8,000 samples for validation, and apply the remaining 740K samples for training.

**Evaluation Metrics** For VQ-Stroke, we primarily consider the reconstruction quality and the compression effectiveness. We evaluate the reconstruction quality with the Fréchet Inception Distance (FID)[2] (Heusel et al., 2017) and the CLIPScore (Hessel et al., 2021). Given that the generated SVG graphics consist solely of lines, we set the background color to white to mitigate the potential biases for FID and CLIPScore brought by the background (Wu et al., 2023a). Additionally, we calculate the Edit Score (EDIT) between the reconstructed SVG code and the Golden SVG (ground truth) code to reflect the fidelity of the reconstructed SVG graphics in replicating fine details. To reflect the practical compression effectiveness of VQ-Stroke, we calculate the Compression Ratio (CR) score between the tokenized SVG code and the stroke tokens, i.e., $\text{CR} = \text{Len}(\text{tokenized SVG code})/\text{Len}(\text{stroke tokens})$. For StokeNUWA, apart from utilizing the metrics mentioned above, we supplement evaluation with Human Preference Score (HPS) (Wu et al., 2023b) and Recall Score to reflect the quality of the generated SVG graphics and their degree of overlap with the Golden SVG code. Additionally, we also report the time required to generate each SVG and conduct the qualitative evaluation. More details about evaluation are illustrated in Appendix A.

**Tasks and Baselines** We evaluate VQ-Stroke and StrokeNUWA with the SVG reconstruction and the text-guided SVG generation tasks, respectively. For VQ-Stroke, considering the absence of works in the field of SVG representation, we focus on comparing the performance of two SF methods, i.e., PI and PC. Additionally, we evaluate the reconstruction performance of two different compression rates, i.e., compression rates of 2 and 4. For StrokeNUWA, we compare with the optimization-based methods, including Vector Fusion (Jain et al., 2023) and the Stable Diffusion (Rombach et al., 2022) combined with LIVE method (Li et al., 2020). Given that optimization-based methods are notably time-intensive, i.e., requiring more than 20 minutes to generate a single SVG on one NVIDIA V100 GPU, we randomly sample 500 instances from the testing set for evaluation to ensure a feasible timeframe. Additionally, we also compare with the LLM-based method Iconshop (Wu et al., 2023a). We re-implement Iconshop with the same Flan-T5 backbone as in StrokeNUWA and use T5 tokenizer to encode the numerical values built in Iconshop. Notably, the primary distinction between Iconshop and StrokeNUWA lies in their approaches to handling visual representation. While Iconshop directly treats SVG code as visual tokens, StrokeNUWA converts SVGs into stroke tokens with VQ-Stroke. We set the maximum model length to 1,500 for IconShop to ensure the SVG completeness.

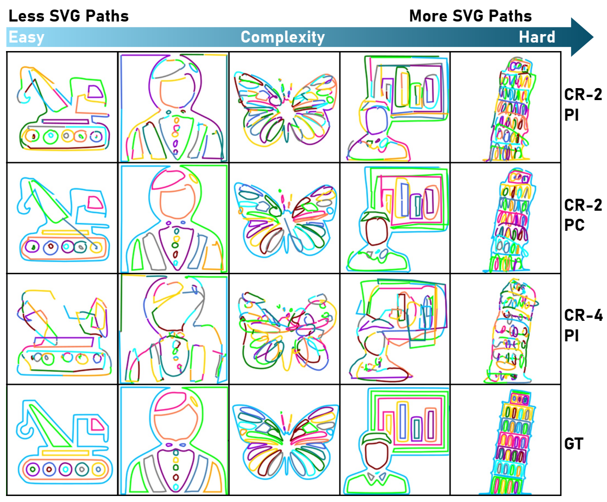


*Figure 6.* Reconstruction cases generated by VQ-Stroke, difficulty (reflected by path numbers) increases from left to right.

**Implementation Details** For VQ-Stroke, we set the depth of the residual vector quantization $d$ to 2, corresponding to compression rates of 2 and 4. Then, we set the codebook size $|\mathcal{B}|$ as 4096, with each code corresponding to a latent representation of 512 dimensions. We set $\alpha = 1$ in Equ. 3 during the training process. For EDM, we utilize the 3B Flan-T5 model (Chung et al., 2022) as the backbone. We utilize DeepSpeed Library (Rajbhandari et al., 2020) to implement models on 64 NVIDIA V100 GPUs and set the maximum model length as 512.

[2]Specifically, we obtain the image features of rendered SVG graphics with the CLIP image encoder (Radford et al., 2021)

*Table 2.* Performance of StrokeNUWA, where "Optim/Pred Length" denotes the actual predicted or optimized number of paths.

| Methods | Visual Performance | | | SVG Code Quality | | | Generation Speed (↓) (*per* SVG) |
|---|---|---|---|---|---|---|---|
| | FID (↓) | CLIPScore (↑) | HPS (↑) | Recall (↑) (Stoke Token) | EDIT (↓) | Optim / Pred Length (*Avg*) | |
| SD & LIVE | 14.236 | 12.908 | 11.210 | 0.028 | - | 160 (32 Path) | ≈ 28.0 min |
| VectorFusion | <u>7.754</u> | <u>17.539</u> | <u>15.901</u> | 0.079 | - | 2,048 (128 Path) | ≈ 30.0 min |
| Iconshop | 17.828 | 8.402 | 8.234 | <u>0.114</u> | <u>24,792.476</u> | 993.244 | ≈ <u>63.743</u> sec |
| StrokeNUWA (PC) | 6.607 | 17.852 | 16.134 | **0.239** | **9,092.476** | 271.420 | ≈ **19.128** sec |
| StrokeNUWA (PI) | **6.513** | **17.994** | **16.801** | 0.207 | 12,249.091 | 271.420 | ≈ **19.128** sec |

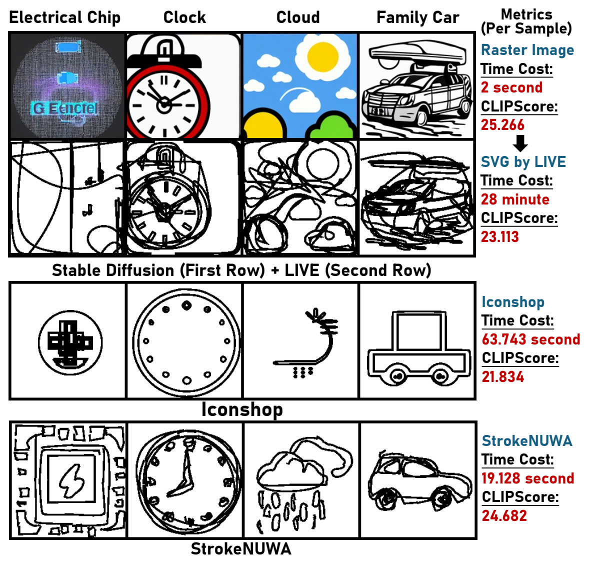
*Table 3.* Performance of VQ-Stroke on SVG reconstruction task, where C-2 and C-4 denote the Compression Rate 2 and 4.

| Methods | FID (↓) | CLIPScore (↑) | EDIT (↓) | CR (↓) |
|---|---|---|---|---|
| SQM (C-2) | - | - | 1,114.791 | 8.549% |
| SQM (C-2) + SF (PC) | 3.751 | 19.861 | **1,096.313** | 8.786% |
| SQM (C-2) + SF (PI) | **3.518** | **20.290** | 1,315.137 | 13.780% |
| SQM (C-4) + SF (PI) | 4.943 | 15.192 | 2,100.671 | **6.890%** |
| Golden SVG | - | - | - | 100% |

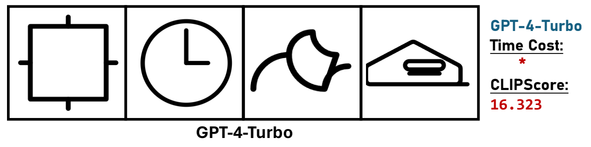
## 4.2. Quantitative Evaluation

**VQ-Stroke** We report the reconstruction quality of VQ-Stroke in Tab. 3. without SF, VQ-Stroke fails to generate results that conform to SVG syntax. After equipping VQ-Stroke with SF, PI facilitates a more faithful approximation of the original SVG graphics by achieving the lowest FID score and demonstrating a higher concordance with the given text prompts, as evidenced by the lowest CLIP score. In contrast, the PC method yields better alignment results with the original SVG code as it achieves the lowest EDIT score. Utilizing compression level 2 (C-4), VQ-Stroke attains a notable Compression Ratio (CR) of 6.9%, maintaining performance on par with that of C-2 as evidenced by comparable CLIPScore and FID. This suggests that VQ-Stroke preserves the semantic integrity of the original SVG graphics despite the substantial path compression.

**StrokeNUWA** As illustrated in Table 2, StrokeNUWA outperforms other methods by achieving superior results. Specifically, in terms of visual performance, StrokeNUWA is capable of generating graphics that more closely resemble the Golden SVG—evidenced by the lowest FID score (6.513) and the highest HPS (16.801). This indicates that our Stroke Tokens offer great compatibility with the LLMs. Moreover, StrokeNUWA has attained the highest CLIPScore (17.994), surpassing optimization-based methods. This suggests that stroke tokens encapsulate visual semantics effectively. In terms of the quality of the SVG code and the efficiency of generation, the Stroke Token not only aligns closely with the Golden standard but also markedly enhances the generation speed, i.e., around 19 seconds of StrokeNUWA V.S. around 30 minutes of optimization-based method LIVE, which underscores the outstanding compressive capability of the stroke token on the original SVG code.



(a) Comparison between StrokeNUWA and other baselines.



(b) Cases generated by GPT-4-Turbo with same keywords. As GPT-4 is not open-source, we cannot get the generation time.

*Figure 7.* Sampled cases from different models in SVG generation task, where the CLIPScore is the average score calculated across four generated cases for each method.

## 4.3. Qualitative Evaluation

**Case Study** We show the reconstruction results of VQ-Stroke with varying complexity levels in Fig. 8 and present a qualitative comparison between StrokeNUWA and other baselines in Fig. 7(a). It is impressive that VQ-Stroke can reconstruct complex SVGs with a limit of only 4,096 codebook size. Then, at the Compression Rate of 2 (CR-2), VQ-Stroke successfully outlines the edge of objects within the graphics, demonstrating that stroke tokens can be highly compressed with a dense representation and inherently incorporate semantic segmentation, which is essential for

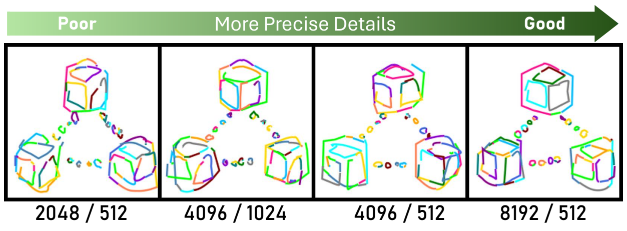


*Figure 8.* Reconstruction performance of difference VQ-Strokes.

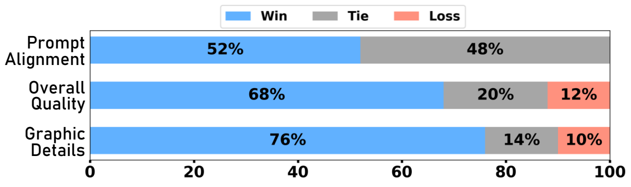


*Figure 9.* Human evaluation between StrokeNUWA and LLM-based method—Iconshop.

retaining visual semantics. Regarding the comparison of StrokeNUWA, we note that employing LLM-based generation methods can result in incomplete SVGs (Iconshop). This is attributed to the excessive SVG code lengths and LLMs struggling to capture the key information embedded within SVG graphics. However, the use of stroke tokens can mitigate these issues by compressing the paths and being compatible with LLMs. Furthermore, we find that the performance of the optimization-based method heavily relies on the outputs generated by the stable diffusion model, which is subject to the limitations of grid tokens mentioned in Sec. 1, e.g., it is hard to capture the visual semantics and tends to generate extra visual information that is not aligned with the text prompt. Besides, the optimization process is extremely slow. In contrast, StrokeNUWA, which utilizes stroke tokens, inherently contains visual semantic segmentation. As a result, the content generated is more aligned with the textual semantics, providing a more coherent and semantically accurate graphic.

**Human Evaluation** Furthermore, we conduct a human evaluation to compare the generated SVG outputs from StrokeNUWA with those produced by the LLM-based method, Iconshop. We select 50 different textual prompts and guide the model to generate corresponding SVGs for evaluation. As depicted in Figure 9, our comparison is founded on three criteria: Prompt Alignment (consistency between the generated result and the text prompt), Overall Quality (the general caliber of SVGs), and Graphic Details (intricacies such as curves). We observe that StrokeNUWA, compared to Iconshop, which regards code as visual representation, not only yields more complete content (better Overall Quality) but produces results more closely aligned with the textual prompts (better Prompt Alignment)[3]. Given that stroke tokens compress the details of SVG, it is natural that StrokeNUWA excels in generating Graphic Details.

[3]The main reason for low Prompt Alignment in Iconshop is also due to the incompleteness of the generated SVGs.

*Table 4.* Comparison among different VQ-Stroke Settings.

| Settings | | FID (↓) | CLIPScore (↑) | EDIT (↓) |
|---|---|---|---|---|
| $\mid\mathcal{B}\mid$ | Dim | | | |
| 2048 | 512 | 5.702 | 19.365 | 2,323.810 |
| 4096 | 512 | 3.518 | 20.290 | 1,315.137 |
| 4096 | 1024 | 3.901 | 20.159 | 1,793.008 |
| 8192 | 512 | **2.639** | **21.014** | **907.106** |

## 5. Ablation Study

### 5.1. Analysis of VQ-Stroke Model Architecture

To investigate the impact of VQ-Stroke architecture configurations on the stroke token performance, we experiment with different codebook sizes $\mid\mathcal{B}\mid$ and codebook dimension $\mathrm{Dim}$. As shown in Tab. 4, we can observe that by increasing the codebook size while simultaneously reducing the dimension of each stroke token, the VQ-Stroke achieves superior performance across multiple metrics. We sample a set of reconstruction cases to showcase the trend of changes in Fig. 8, which indicates that, with a larger codebook size and smaller dimension, the VQ-Stroke can delineate details with greater accuracy, e.g., straighter lines.

### 5.2. Comparison with GPT-4

We compare the generation results with GPT-4 (Achiam et al., 2023) by employing the following template to guide GPT-4 in producing the corresponding SVG code: `Generate SVG codes in icon style based on keywords:{KEYWORDS}`. We show the rendered SVGs in Fig. 7(b), where we can observe that GPT-4 can only generate simple SVGs, which is consistent with LLM-based methods. Moreover, GPT-4 often yields SVGs that are incongruent with the associated text. On the contrary, StrokeNUWA can generate complex SVGs that are consistent with textual prompts.

## 6. Conclusion and Future Work

This paper presents StrokeNUWA, a pioneering study that explores a superior visual representation—"stroke" tokens, as an alternative method for expressing images through vector graphics. Stroke tokens not only preserve the semantic integrity of the images but are also conducive to processing by LLMs. Moreover, strokes in vector graphics can be highly compressed. Experiments indicate that, equipped with stroke tokens, LLMs can achieve superior results across various metrics in the SVG synthesis task. Moving forward, we aim to continue improving the quality of stroke tokens through advanced visual tokenization methods for LLMs.

## Impact Statement

The implications of this work are manifold, potentially revolutionizing the visual synthesis from another format of image, vector graphics. As stroke tokens refine the interplay between visual representation and LLMs, future advancements in visual tokenization techniques designed for LLMs are anticipated. Moving forward, the community can extend stroke token application into wider tasks and domains, including SVG comprehension and open-domain SVG synthesis for images from the real world. As we pioneer this nascent field, we are conscious of the profound societal impact that such advancements in machine learning and graphical representations hold. The capabilities for automated graphic design, scalable vector graphics production, and enhanced digital artistry foreshadow considerable shifts in industries reliant on visual content. By forging new pathways for artistic expression and visual communication, our work stands to not only contribute to the scientific community but also to catalyze transformations in creative, technological, and educational sectors. We recognize the importance of our work and our responsibility to ensure that our contributions to the field are conducted ethically, aiming to benefit society as a whole, democratize the visual landscape, and enrich it through responsible and judicious innovation.

## References

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

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Barnes, C. F., Rizvi, S. A., and Nasrabadi, N. M. Advances in residual vector quantization: A review. *IEEE transactions on image processing*, 5(2):226–262, 1996.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Cao, S., Yin, Y., Huang, L., Liu, Y., Zhao, X., Zhao, D., and Huang, K. Efficient-vqgan: Towards high-resolution image generation with efficient vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7368–7377, 2023.

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

Chang, H., Zhang, H., Jiang, L., Liu, C., and Freeman, W. T. Maskgit: Masked generative image transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11315–11325, 2022.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

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

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Esser, P., Rombach, R., and Ommer, B. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

Ghosal, D., Majumder, N., Mehrish, A., and Poria, S. Text-to-audio generation using instruction-tuned llm and latent diffusion model. *arXiv preprint arXiv:2304.13731*, 2023.

Hessel, J., Holtzman, A., Forbes, M., Bras, R. L., and Choi, Y. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Hu, Q., Zhang, G., Qin, Z., Cai, Y., Yu, G., and Li, G. Y. Robust semantic communications with masked vq-vae enabled codebook. *IEEE Transactions on Wireless Communications*, 2023.

Jain, A., Xie, A., and 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.

Kim, Y., Mo, S., Kim, M., Lee, K., Lee, J., and Shin, J. Bias-to-text: Debiasing unknown visual biases through language interpretation. *arXiv preprint arXiv:2301.11104*, 2023.

Kim, Z. M., Du, W., Raheja, V., Kumar, D., and Kang, D. Improving iterative text revision by learning where to edit from other revision tasks. *arXiv preprint arXiv:2212.01350*, 2022.

Kondratyuk, D., Yu, L., Gu, X., Lezama, J., Huang, J., Hornung, R., Adam, H., Akbari, H., Alon, Y., Birodkar, V., et al. Videopoet: A large language model for zero-shot video generation. *arXiv preprint arXiv:2312.14125*, 2023.

Lee, D., Kim, C., Kim, S., Cho, M., and HAN, W. S. Draft-and-revise: Effective image generation with contextual rq-transformer. *Advances in Neural Information Processing Systems*, 35:30127–30138, 2022.

Li, T.-M., Lukáč, M., Gharbi, M., and Ragan-Kelley, J. Differentiable vector graphics rasterization for editing and learning. *ACM Transactions on Graphics (TOG)*, 39 (6):1–15, 2020.

Ma, X., Zhou, Y., Xu, X., Sun, B., Filev, V., Orlov, N., Fu, Y., and Shi, H. Towards layer-wise image vectorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16314–16323, 2022.

Martinez, J., Hoos, H. H., and Little, J. J. Stacked quantizers for compositional vector compression. *arXiv preprint arXiv:1411.2173*, 2014.

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.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Reddy, M. D. M., Basha, M. S. M., Hari, M. M. C., and Penchalaiah, M. N. Dall-e: Creating images from text. *UGC Care Group I Journal*, 8(14):71–75, 2021.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Singhal, K., Tu, T., Gottweis, J., Sayres, R., Wulczyn, E., Hou, L., Clark, K., Pfohl, S., Cole-Lewis, H., Neal, D., et al. Towards expert-level medical question answering with large language models. *arXiv preprint arXiv:2305.09617*, 2023.

Su, H., Liu, X., Niu, J., Cui, J., Wan, J., Wu, X., and Wang, N. Marvel: Raster gray-level manga vectorization via primitive-wise deep reinforcement learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

Sun, Q., Yu, Q., Cui, Y., Zhang, F., Zhang, X., Wang, Y., Gao, H., Liu, J., Huang, T., and Wang, X. Generative pretraining in multimodality. *arXiv preprint arXiv:2307.05222*, 2023.

Tang, Z., Zhou, K., Wang, P., Ding, Y., Li, J., et al. Detoxify language model step-by-step. *arXiv preprint arXiv:2308.08295*, 2023.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Wang, C., Chen, S., Wu, Y., Zhang, Z., Zhou, L., Liu, S., Chen, Z., Liu, Y., Wang, H., Li, J., et al. Neural codec language models are zero-shot text to speech synthesizers. *arXiv preprint arXiv:2301.02111*, 2023.

Wang, Y., Pu, G., Luo, W., Wang, Y., Xiong, P., Kang, H., and Lian, Z. Aesthetic text logo synthesis via content-aware layout inferring. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2436–2445, 2022.

Wu, C., Liang, J., Ji, L., Yang, F., Fang, Y., Jiang, D., and Duan, N. Nüwa: Visual synthesis pre-training for neural visual world creation. In *European conference on computer vision*, pp. 720–736. Springer, 2022.

Wu, R., Su, W., Ma, K., and Liao, J. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *ACM Transactions on Graphics (TOG)*, 42(6): 1–14, 2023a.

Wu, X., Sun, K., Zhu, F., Zhao, R., and Li, H. Human preference score: Better aligning text-to-image models with human preference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2096–2105, 2023b.

Xing, X., Zhou, H., Wang, C., Zhang, J., Xu, D., and Yu, Q. Svgdreamer: Text guided svg generation with diffusion model. *arXiv preprint arXiv:2312.16476*, 2023.

Yu, L., Lezama, J., Gundavarapu, N. B., Versari, L., Sohn, K., Minnen, D., Cheng, Y., Gupta, A., Gu, X., Hauptmann, A. G., et al. Language model beats diffusion–tokenizer is key to visual generation. *arXiv preprint arXiv:2310.05737*, 2023.

Zhang, T., Liu, H., Zhang, P., Cheng, Y., and Wang, H. Beyond pixels: Exploring human-readable svg generation for simple images with vision language models. *arXiv preprint arXiv:2311.15543*, 2023.

## A. Evaluation Metrics

Here we will elaborate more meticulously on the setup of the evaluation.

### A.1. Automatic Evaluation Metrics

**EDIT Score** The EDIT Score calculates how many insertions, substitutions, and deletions are needed to make two strings identical [4]. In the field of NLP, the Edit Score measures the similarity between two strings (Kim et al., 2022; Tang et al., 2023). In the task of SVG generation, we evaluate the quality of the generated SVG from the code perspective, i.e., the closer the generated SVG code is to the ground truth SVG, the higher the quality of the SVG path, indicating fewer repeated paths and missing paths. It is worth noting that we only test the direct SVG generation methods, including Iconshop and StrokeNUWA. For optimization-based methods, e.g., SD & LIVE and VectorFusion, which continuously optimize raster images and do not have a fixed ground truth SVG, it is not possible to evaluate with EDIT score. We convert all the SVGs into the code format, treat the codes as strings, and calculate the corresponding EDIT scores.

**Recall Rate** Based on our observations, we have noticed that for the majority of SVGs, some key paths and keywords match. For example, given the keyword "clock", the corresponding SVG should contain key SVG paths resembling clock hands. By identifying these key paths, one can roughly understand that the SVG represents a clock. Therefore, we design the Recall Score to measure how many key paths in the generated SVG can match the ground truth. Specifically, we convert both the generated SVG and the ground truth SVG into stroke tokens with VQ-Stroke and then calculate how many strokes in the generated SVG's corresponding stroke tokens appear in the ground truth using the following formula: $Recall = \frac{|P \cap \mathcal{G}|}{|\mathcal{G}|}$, where $|P \cap \mathcal{G}|$ represents the number of overlapping stroke tokens between the generated stroke tokens $P$ and the ground truth stroke tokens $\mathcal{G}$. We convert all SVGs into the stroke token format, subsequently computing the recall rate.

### A.2. Human Evaluation

Considering the potential bias and inaccuracies that automated metrics may introduce in evaluating SVGs, e.g., CLIP-Score (Kim et al., 2023), we organize a human evaluation process. In this human evaluation, we employ 10 evaluators and provide each with 100 evaluation samples. Each sample set included an SVG generated by StrokeNUWA and one generated by Iconshop. Evaluators are tasked with assessing the following aspects:

- **Prompt Alignment**: Whether the generated SVG aligns with the given prompt.
- **Overall Quality**: The general quality of the generated SVG, such as balance of content and clarity.
- **Graphic Details**: The presence of detailed elements in the generated SVG, such as intricacies outlined by curves.

Evaluators are instructed to evaluate StrokeNUWA against Iconshop in each metric, determining whether StrokeNUWA wins, ties, or loses against Iconshop. Before releasing the human evaluation cases, we conducted thorough checks to ensure there were no private information or problematic biases present in the cases. Additionally, we refrained from collecting personal information or inquiring about evaluators' private details during the annotation process.

## B. Additional Ablation Experiments

### B.1. Effectiveness of Different Textual Prompts

We study the effectiveness of different textual prompts on StrokeNUWA. Specifically, as shown in Fig. 10, we embed keywords into different textual prompt templates to guide SVG generation. These new prompts were then fed into the StrokeNUWA model to generate SVGs. It is worth mentioning that, for the sake of conciseness in illustrating the impact of different textual prompts, we employ the PI method as the SVG fixer and prompt the StokeNUWA with different textual prompts. We try five different textual prompts (including the one used in the main experiment). Notably, the first four templates place the keywords in different positions within the prompt, while the last one reconstructs the keywords into a natural language sentence using LLM. For instance, for the keywords "cloud" and "rain", we can rewrite them in natural language as: "Raindrops are falling from the cloud".

[4] https://en.wikipedia.org/wiki/Edit_distance

Templates for Different Textual Prompts

**Template 1 (Utilize in the main experiment)**
Generating SVG according to keywords: {Keywords}

**Template 2 (Shuffle the Keywords order)**
Generating SVG according to keywords: {Shuffled Keywords}

**Template 3**
Please generate an SVG that includes the following attributes: {Keywords}

**Template 4**
Here are some keywords: {Keywords}, generate the corresponding SVG:

**Template 5 (Re-organize the keywords into natural language**
{Natural language prompt including keywords}

*Figure 10.* Templates for different textual prompts.

*Table 5.* Effectiveness of Different Textual Prompts.

| **Prompt** | Visual Performance | | | SVG Code Quality | | |
|---|---|---|---|---|---|---|
| | FID (↓) | CLIPScore (↑) | HPS (↑) | Recall (↑) (Stoke Token) | EDIT (↓) | Optim / Pred Length (*Avg*) |
| Template 1 | 6.513 | 17.994 | 16.801 | 0.207 | 12,249.091 | 271.420 |
| Template 2 | **6.612** | **18.113** | **16.942** | **0.211** | **12,223.724** | 274.006 |
| Template 3 | 8.726 | 16.026 | 14.093 | 0.199 | 13,287.631 | 263.196 |
| Template 4 | 11.607 | 13.852 | 9.097 | 0.101 | 19,167.086 | 181.059 |
| Template 5 | - | - | - | - | - | - |

We report the StrokeNUWA performance with different textual prompts in Tab. 5. We can observe that if the textual prompt doesn't change much, i.e., it stays consistent with the training phase or has minor modifications (such as changing the order of keywords), the performance of the StrokeNUWA is not greatly affected and might even generate SVGs of higher quality (Template 2). However, if a new textual prompt is adopted, the position of the keywords becomes very important. If the keywords are still at the end of the textual prompt (Template 3), the model's performance shows a slight decrease across various metrics. However, suppose the keywords are not at the end (Template 4). In that case, StrokeNUWA fails to generate complex SVGs, resulting in failures on some complicated test samples and a significant drop in overall metrics. For the last case (Template 5), if keyword prompts are completely unused by StrokeNUWA during the training stage, i.e., the prompts during inference are entirely inconsistent with the prompts used during training, then StrokeNUWA can only generate a few simple SVGs, thus making the generated SVGs difficult to evaluate. We also provide some generated SVGs from StrokeNUWA under different textual prompts for reference in Fig. 12.

## B.2. Semantic Clusters of Strokes

We adopt the following configuration of VQ-Stroke (the depth of the residual vector quantization is 2, with compression rates of 2 and 4, respectively, and a vocabulary size of 4096, which is the model configuration used in the main experiment), and performed clustering on all stroke tokens in the stroke codebook.

We manually set the number of clusters to 10 categories and analyze them using the T-SNE method[5]. We present the results of vocabulary clustering under the condition of a compression rate of 2 in Fig. 11, where we divide the clustering results into two types. Specifically, the cases in the right column of the above figure indicate that the distribution of these stroke tokens is very concentrated. These Stroke Tokens have very distinct features, such as representing Dots, Vertical Lines, or direction-specific short lines. In contrast, the cases in the left column represent strokes with less distinct features. This type of strokes includes various different categories of stroke types (for example, short/long strokes, direction-specific hooks, etc.). We believe that these types of strokes (on the left) constitute complex SVGs, and are greatly influenced by the vocabulary size set in VQ-Stroke. A larger vocabulary may learn more complex and layered strokes.

[5] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

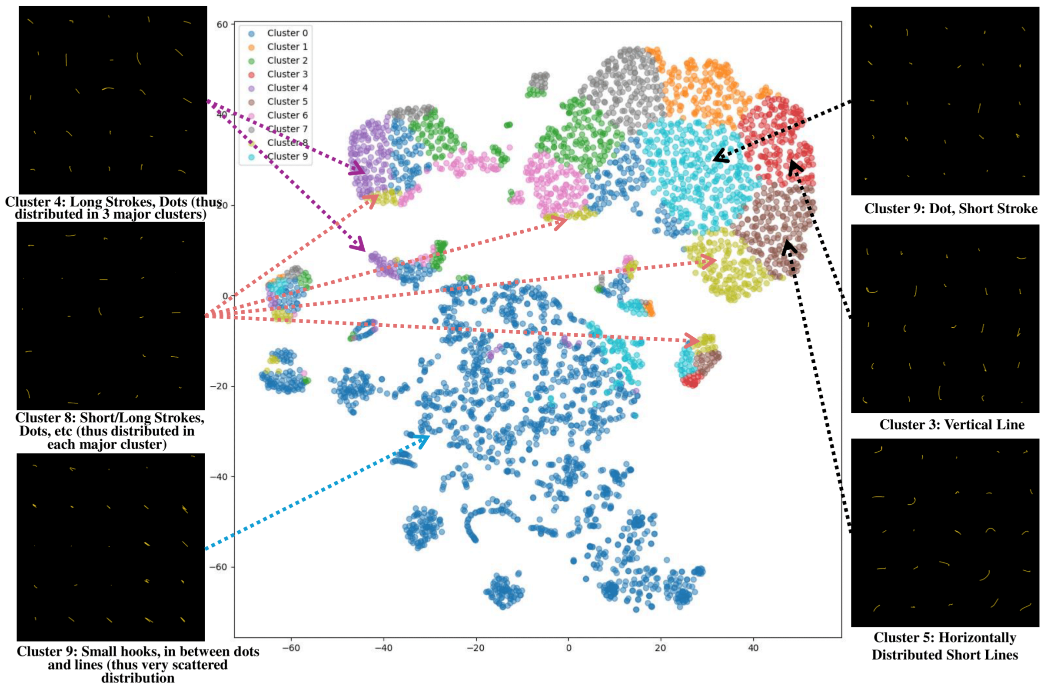


*Figure 11.* Semantic Clusters of Strokes

## C. SVG Generation with more Attributes

In order to incorporate more elements into the process of Stroke generation, we have expanded the Code to Matrix method (as described in section 3.2.1 of the original paper). Specifically, we have added three placeholders to the original matrix, so that Equation 1: $K_{ij} = f(C_{ij}) = (T, x_0, y_0, c_0^x, c_0^y, c_1^x, c_1^y, x_1, y_1)_{ij}$ becomes $(\mathcal{K}_i^j)' = f'(\mathcal{C}_i^j) = (T, color_i, width_i, opacity_i, x_0, y_0, c_0^x, c_0^y, c_1^x, c_1^y, x_1, y_1)_i^j$. It's worth noting that the placeholders ($color_i$, $width_i$, and $opacity_i$) here can be expanded to any number, which can be defined by the user and reflected to the SVG through rules. For simplicity of demonstration and validate the feasibility of adding more attributes into "stroke tokens", we insert three special placeholders: 1) $color_i$: to control the color of the SVG path; 2) $width_i$: to control the width of the SVG path; and 3) $opacity_i$: to control the transparency of the SVG path. Since the SVG dataset we used does not provide these parameters, we randomly generated some values for a toy experiment.

For clarity, we compare the predicted strokes from VQ-Stroke model under different settings in Fig. 13, where Raw predicted SVG only uses numerical coordinates as attributes, while Random Colored SVG fills the color attribute in the SVG Path using a rule-based method. The two sub-figures in the bottom of Fig. 13 show the results trained with **VQ-Stroke** with different vocabulary sizes and attributes, i.e., color, path width, and opacity.

We retrain the EDM model in the paper using the aforementioned stroke tokens which fused with 3 attributes. Finally, we show some generated cases in Fig. 14.

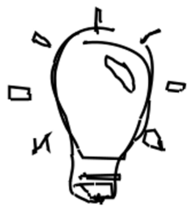
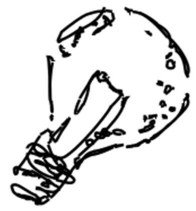
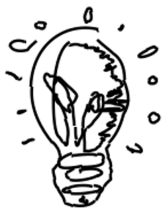
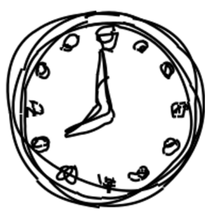
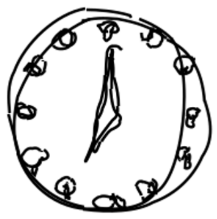
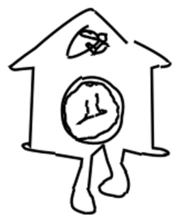
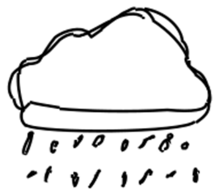
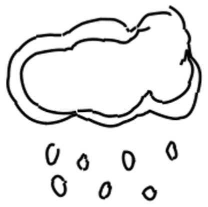
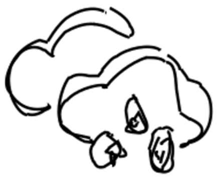
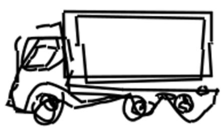
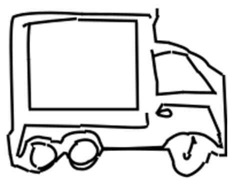
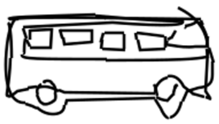
| | | |
|---|---|---|
|  |  |  |
| **Template 1:** Generate SVG according to keywords: Light Bulb, Emitting Light | **Template 3:** Please generate an SVG that includes the following attributes: Light Bulb, Emitting Light | **Template 4:** Here are some keywords: Light Bulb, Emitting Light, generate the corresponding SVG: |
|  |  |  |
| **Template 1:** Generate SVG according to keywords: Clock, Night | **Template 3:** Please generate an SVG that includes the following attributes: Clock, Night | **Template 4:** Here are some keywords: Clock, Night, generate the corresponding SVG: |
|  |  |  |
| **Template 1:** Generate SVG according to keywords: Rain, Weather | **Template 3:** Please generate an SVG that includes the following attributes: Rain, Weather | **Template 4:** Here are some keywords: Rain, Weather, generate the corresponding SVG: |
|  |  |  |
| **Template 1:** Generate SVG according to keywords: Car, Vehicle | **Template 3:** Please generate an SVG that includes the following attributes: Car, Vehicle | **Template 4:** Here are some keywords: Car, Vehicle, generate the corresponding SVG: |

*Figure 12.* SVG cases generated by StrokeNUWA with different textual prompts.

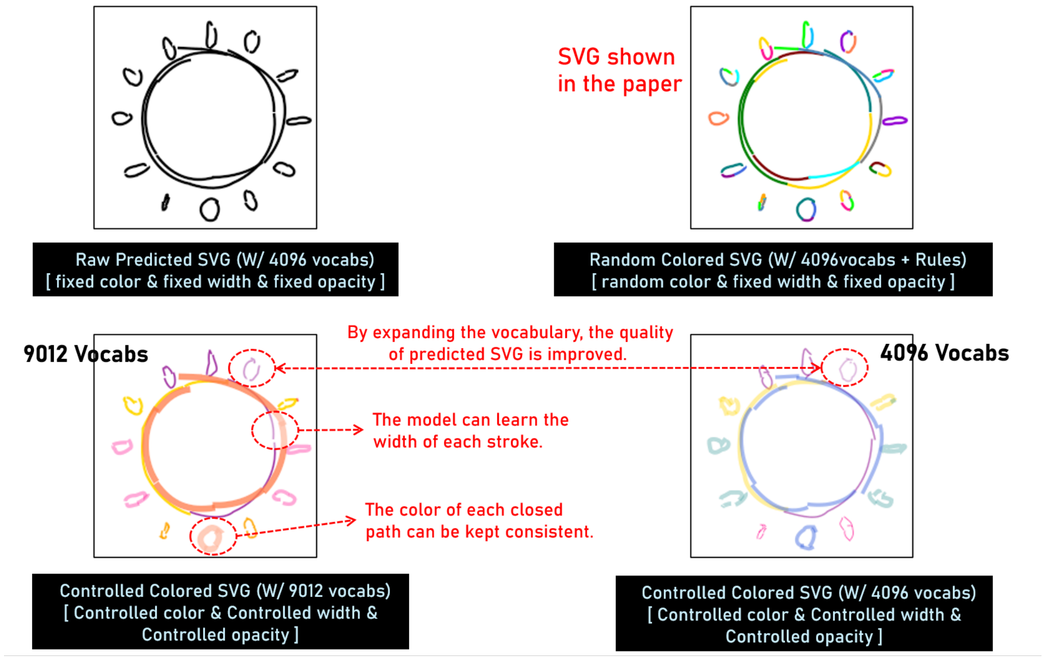

SVG shown
in the paper
Raw Predicted SVG (W/ 4096 vocabs)
[ fixed color & fixed width & fixed opacity ]
Random Colored SVG (W/ 4096vocabs + Rules)
[ random color & fixed width & fixed opacity ]
9012 Vocabs
4096 Vocabs
By expanding the vocabulary, the quality of predicted SVG is improved.
The model can learn the width of each stroke.
The color of each closed path can be kept consistent.
Controlled Colored SVG (W/ 9012 vocabs)
[ Controlled color & Controlled width & Controlled opacity ]
Controlled Colored SVG (W/ 4096 vocabs)
[ Controlled color & Controlled width & Controlled opacity ]

*Figure 13.* SVG reconstruction with more attributes.

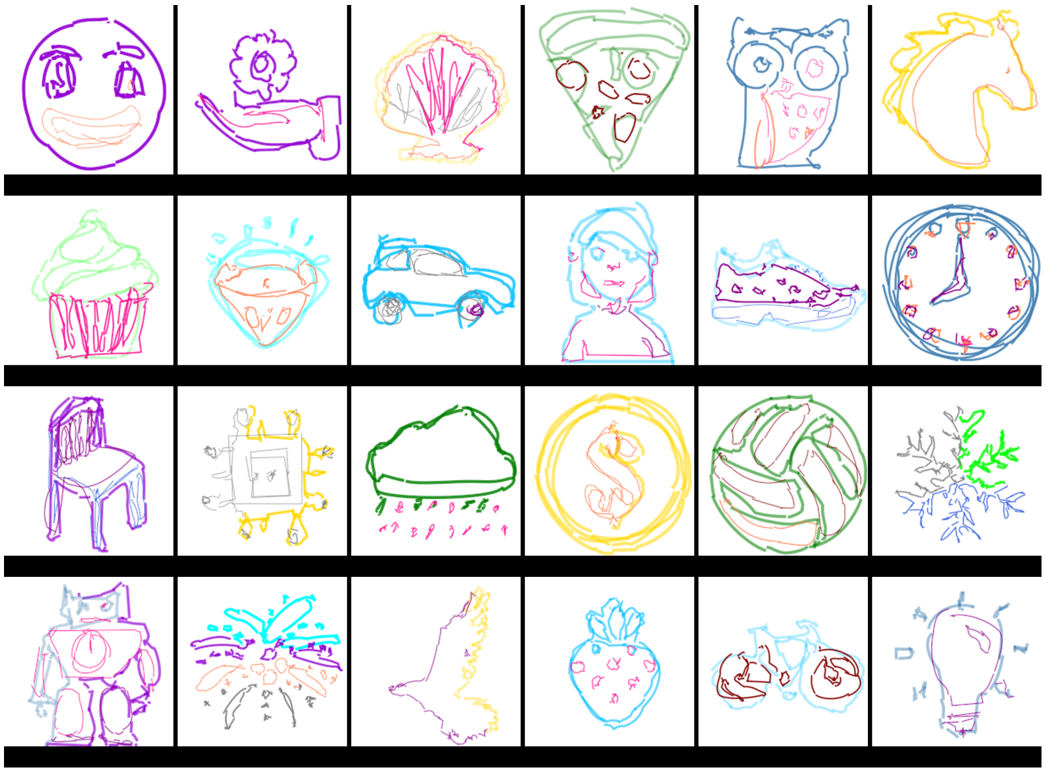


*Figure 14.* SVG generated by StrokeNUWA fused with more attributes.