VectorWeaver: Transformers-Based Diffusion Model for Vector Graphics Generation

Ivan Jarsky[a], Maxim Kuzin[b], Valeria Efimova[c], Viacheslav Shalamov[d]
and Andrey Filchenkov[e]
ITMO University, Kronverksky Pr. 49, St. Petersburg, Russia
fi *fi*

Keywords: Vector Graphics, Image Generation, Diffusion Models, Transformer.

Abstract: Diffusion models generate realistic results for raster images. However, vector image generation is not so successful because of significant differences in image structure. Unlike raster images, vector ones consist of paths that are described by their coordinates, colors, and stroke widths. The number of paths needed to be generated is unknown in advance. We tackle the vector image synthesis problem by developing a new diffusion-based model architecture, that we call VectorWeaver, including two transformer-based stacked encoders and two transformer-based stacked decoders. For training the model, we collected a vector images dataset from public resources, however, its size was not enough. To enrich and enlarge it we proposed new augmentation operations specific for vector images. To train the model, we designed a specific loss function, which allowed the generation of objects with smooth contours without artifacts. Qualitative experiments demonstrate the superiority and computational efficiency of the proposed model compared to the existing vector image generation methods. The vector image generation code is available at https://github.com/CTLab-ITMO/VGLib/tree/main/VectorWeaver.

1 INTRODUCTION

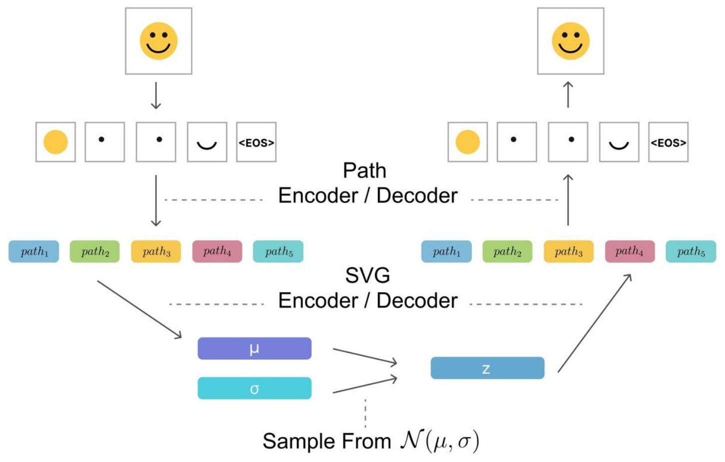
Image generation methods have been developing rapidly in recent years. New diffusion-based (Saharia et al., 2022; Xu et al., 2022; Wu et al., 2024; Du et al., 2023) and transformer-based (Chang et al., 2022; Yu et al., 2022) approaches have achieved excellent generation quality. However, training of these models requires hundreds of GPU days to obtain an image of 1024x1024 pixels.

An alternative to raster images is vector graphics, whose scalability allows to achieve a sufficient image resolution. However, there was no progress in generating vector images until recently. It was possible to generate vector images containing only points or thin curves (Frans et al., 2021; Li et al., 2020). In contrast, artists usually draw vector images with color-filled shapes. Despite serious differences between vector graphics and raster graphics, VectorFusion (Jain et al., 2022) based on Stable Diffusion (Rombach et al., 2022) was proposed to generate a raster image and then vectorize it using Layer-wise Image Vectorization (LIVE) approach (Ma et al., 2022). Main disadvantages of using LIVE vectorization are its long generation time and requirement of hyperparameters tuning for each case. At the same time, other vectorization approaches also have serious disadvantages, such as non-optimal shape generation – inappropriate shape number and their redundant complexity (Dziuba et al., 2023). Therefore, we choose another approach of vector image generation without referring to raster domain.

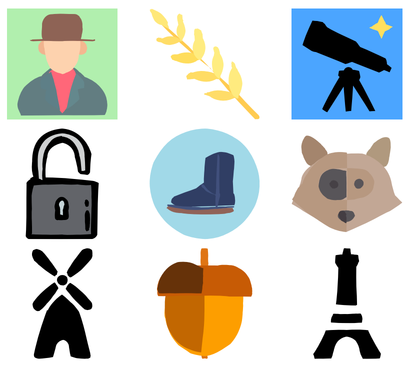
Vector image generation and text-to-image synthesis in vector format is challenged by the complete difference in the structure of images in the raster and vector domains. While raster images are a matrix of pixels, vector images contain a set of shapes that need to be displayed on the canvas. One of the most popular vector graphics formats is SVG format, which is based on the XML text format. However, generating such images directly in text format, for example, using LLM, is a very difficult task (Timofeenko et al., 2023), especially because of the requirement to conform to a regular file structure with

[a] https://orcid.org/0000-0003-1107-3363
[b] https://orcid.org/0009-0001-0982-1740
[c] https://orcid.org/0000-0002-5309-2207
[d] https://orcid.org/0000-0002-5647-6521
[e] https://orcid.org/0000-0002-1133-8432

Jarsky, I., Kuzin, M., Efimova, V., Shalamov, V. and Filchenkov, A.
VectorWeaver: Transformers-Based Diffusion Model for Vector Graphics Generation.
DOI: 10.5220/0013185100003912
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2025) - Volume 2: VISAPP*, pages 184-195
ISBN: 978-989-758-728-3; ISSN: 2184-4321
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.



(a)



(b)

Figure 1: (a) The general scheme of VectorWeaver. Four transformer models are trained, two for encoding and two for decoding. We denote embeddings with the colored rectangles in the scheme. Note that in our model diffusion is used in the innermost latent space. (b) Images unconditionally generated by the proposed custom-trained VectorWeaver model. Generation of random latent vector is performed using the diffusion model, the entire image is synthesised using VAE Decoder (SVG and Path Decoders).

many shapes and their attributes. Currently there is no such high-quality model, which could generate diverse and high-quality images in terms of structure and content. Therefore, in our research we focus on generating images consisting only of shapes formed from Bézier curves with their attributes. A non-trivial issue remains both the selection of the number of shapes applied to the canvas and the number of curves they should consist of. As a result, the existing excessive variability and uncertainty create instability when generating images. Another problem is the low prevalence of vector graphics, which results in a small amount of qualitative and detailed vector images in the public domain

In this paper we solve aforementioned problems and make the following contributions:

- We propose a new computationally efficient architecture for vector image generation based on diffusion process and transformer-based VAE capable to process path sequences of an arbitrary length, which we call VectorWeaver. The model achieved competitive performance on the task of unconditional vector image generation.
- We do not use vectorization approaches and differentiable rasterization. Instead, we adapt existing loss functions for vector graphics domain. We also involve the adversarial loss to improve the smoothness of the generated images.
- We propose augmentation operations specific for vector graphics.

2 RELATED WORK

2.1 Vector Graphics and Images Vectorization

The basic idea of vector graphics is to store images as a sequence of shapes to be drawn on the canvas. SVG is the most popular vector graphics format, which defines the image using the XML markup. The main tags describing figures are <circle>, <ellipse>, <rect>, <polygon>, but the most universal tag covering all these types of shapes is <path>, an arbitrary line. To draw such a shape, commands *moveTo*, *lineTo*, *cubicCurveTo*, and others with the corresponding coordinates are used. Thus, the task of generating a vector image is reduced to generating a sequence of path tags, each of which is a sequence of these commands.

The main purpose of vector graphics is to sketch simple, often abstract, images; mostly demanded in design, for example, are logos, fonts, and icons. The key feature and advantage of vector images over bitmaps is the preservation of resolution quality regardless of scale. However, the process of rendering vector graphics suffers in the presence of a large number of details and figures, therefore, generated images should be quite compact and not over complicated.

An important milestone in the vector graphics domain was DiffVG (Li et al., 2020), a library for the differentiable rasterization of vector images. The main contribution of DiffVG is transparent – after vector images rasterization, we can use bitmap losses to process them. DiffVG proves its efficiency by offering the VAE and GAN models, both generating the parameters of a vector image, whose rasterization is similar to the input image. However, these two models allow generating only simple vector fonts, generating more complex images might require additional losses and approaches.

One of the first works using DiffVG was the Im2Vec vectorizer (Reddy et al., 2021). The proposed VAE is a modification of the model proposed earlier in DiffVG. The authors paid special attention to the generation of closed figures and implemented a new method that samples and sequentially deforms the base circle. The authors claimed that collecting large datasets of vector images ceased to be a necessity, however, the experiments show that the thesis can hold true only for simple images consisting of a very small number of shapes.

The LIVE vectorizer (Ma et al., 2022) also extended opportunities of DiffVG. It iteratively adds a new path to the image, which is further sequentially optimized. It became possible to obtain vector analogs of raster images containing the specified number of paths. However, this approach is very time-consuming and vectorization even of a simple image can be performed in more than tens of minutes.

Another iterative approach is used in the CLIPGen vectorizer (Shen and Chen, 2021). The idea is to train a class-guided model to predict if a new layer with generated parameters should be added to the current vector image. However, this approach only works for very simple images and performs vectorization very slowly.

An overview of vectorization methods is provided in the article (Dziuba et al., 2023). The authors find that modern approaches are either highly time-consuming or not versatile. Therefore, we conclude that vectorization is undesirable to use when generating vector images.

2.2 Vector Graphics Generation

Variational autoencoders (VAE) (Kingma and Welling, 2019) were among the first popular architectures for generating raster graphics. To memorize and sample objects, it was originally proposed to predict the mean and standard deviation, which defines the continuous space of normal distribution. However, the VQ-VAE (Van Den Oord et al., 2017) work showed that additional quantization helps to improve convergence without losing the diversity of generated images.

The advent of techniques for generating raster images was followed by the emergence of methods for synthesizing vector graphics. Among them, VAEs were chosen as the main architecture for generating vector graphics.

One of the first simplest models is the SVG-VAE (Lopes et al., 2019), designed to generate fonts. A character is represented as exactly one <path>, and thus the task is to generate only one sequence of path commands. To memorize the drawing style of the characters, the authors trained a conditional VAE. The second model generates the commands sequence of the output symbol, taking the previously predicted style of the reference character and the label of the new character as input.

To generate images containing several shapes, the DeepSVG architecture was proposed (Carlier et al., 2020); it is also a VAE using a two-step method of encoding and decoding an image. The first step encodes each image path, then the second step encodes the set of previously encoded paths. After that, decoding is performed twice in the reversed way. DeepSVG allowed generating images containing several shapes as well memorizing and synthesizing various parameters of shapes, for example, color, opacity, and width of the strokes. A serious disadvantage of these approaches is that they strongly depend on the size of the training dataset, the collection of which is a non-trivial task. The authors train an autoencoder that translates vector parameters into a latent space without using rasterization. They represent each segment as a token in a transformer (Vaswani et al., 2017). However only small vector images (with maximum of five paths) generation was presented, therefore, the model is poorly applicable to the generation task.

In DeepVecFont (Wang and Lian, 2021), the authors proposed to generate an additional raster image by carrying out iterative optimization of the resulting vector image at the end, bringing it closer to the raster analog. This approach helps straighten the unevenness of the vector image lines.

One of the attempts to use GAN for vector graphics synthesis was the CoverGAN (Efimova et al., 2022), designed to generate a vector image for a cover based on the user's music track and given emotions. The authors developed the idea of Im2Vec and used circles as well as ovals, triangles, and other regular shapes as basic shapes for the variety of the results.

Iterative optimization has become another area of developing models for generating vector images. CLIPDraw (Frans et al., 2021) and StyleCLIPDraw (Schaldenbrand et al., 2022) consider the generation and the style transfer according to a given text description as a sequential process of changing the current vector image, starting with a randomly generated one. The CLIP model (Radford et al., 2021) is used as a loss function, which allows evaluating the text-to-image relationship. Generated images are drawn with "strokes" and are having a lot of shapes, which is not acceptable for vector graphics.

There is also a paper (Yan, 2023) that attempts to combine ideas from the works of DeepSVG and DeepVecFont. It proposes to generate an image not only with a two-level VAE, but also at the same time generate a raster copy of it and finally make a fine tuning of the vector image referring to the raster one. However, qualitative results were obtained only for simple font images.

The models discussed in this section lack sufficient quality. Furthermore, the use of DiffVG imposes significant limitations due to its instability and the inability to optimize image batches simultaneously. However, we find the concept of DeepSVG, which employs a two-tier VAE, to be quite intriguing, and we incorporate it into our work.

2.3 Image Generation with Diffusion Models

Diffusion models (Ho et al., 2020) are the state-of-the-art for generating raster images (Saharia et al., 2022). Its idea is to train a model to remove noise added to images. The training consists of two stages. At the first stage, called the forward pass, noise is added to a slightly forgotten image. This operation is repeated a large number of times (usually $T = 1000$). The second stage, called the backward pass, is predicting the original image from the noisy one.

After training, we can feed the model with random noise and after T steps of denoising obtain realistic images.

The idea behind the Latent Diffusion (Rombach et al., 2022) is that it is unnecessary to train the model to remove noise from the original image. Instead, we can train an autoencoder once and then operate not with the image space, but with the latent space.

This approach has been applied to generate vector images in VectorFusion (Jain et al., 2022). It uses the Stable Diffusion to generate a raster image, then vectorizes it using LIVE (Ma et al., 2022), and finally fine-tunes it using the Sample Distribution Loss. Despite the increased generation quality, the problems remain the same as in CLIPDraw, because the Stable Diffusion model is also trained on raster images, which results in an image drawn "with strokes"" (homogeneous areas are covered with several shapes, which is considered low-quality in vector graphics). Moreover, this method inherits all drawbacks of the LIVE method, for example, such as high image generation time and the requirement for pre-setting hyperparameters.

Another work (Zhang et al., 2023) applies Stable Diffusion with vectorization methods, in order to change the input vector image according to the transmitted text description. Using Stable Diffusion, the method first generates a target bitmap based on the text condition, then, based on dual features extracted from deep networks, the input vector image is adapted to the target bitmap and the final optimization of this vector image is performed. In addition to the fact that this method is too narrowly focused and does not allow generating vector images unconditionally or by textual condition, the use of a pre-trained diffusion raster model leads to problems similar to those that arise with vectorizing approaches.

One of the latest works, which uses the diffusion process, is SVGDreamer (Xing et al., 2024) - an optimization-based method for creating vector images on text prompts. The authors introduce semantic-driven image vectorization method (SIVE), which aim is to distinguish well the background and foreground objects and to optimize them with the guidance of attention maps, obtained from Text-to-Image Diffusion model. After that Vectorized Particle-based Score Distillation (VPSD) is used to finetune previously generated image. The authors show pleasant results, however, the resulting images contain too many shapes and the whole process is quite highly loaded. The main problem with the approach is the use of vectorization, which is difficult to correctly configure for automatic use.

Nevertheless, the use of diffusion approaches has significantly improved the quality of vector image generation. We also use a diffusion model in our work.

2.4 Vector Image Generation with LLMs

A promising approach to generating vector graphics may be the use of large language models (LLMs). Since a vector image in SVG format is a text file, it would be possible to train a language model to generate such files. There are studies (Nishina and Matsui, 2024; Zou et al., 2024; Xu and Wall, 2024; Cai et al., 2024) devoted to evaluating the quality of SVG image generation using open, well-known pre-trained chat models. The conclusion of them is that the quality of image generation is very strongly influenced by prompt engineering, but at the same time it is almost impossible to achieve high-quality results without serious finetuning of models for generating vector images.

We found only one paper (Wu et al., 2023) dedicated to SVG generation using LLM. The authors trained a Transformer-Decoder model to generate SVG text tokens in an autoregressive way based on a text condition. Although the authors demonstrate good results in their work, we observed that the model likely memorized the dataset on which it was trained and it shows poor results for unseen data. As a result, even for simple text queries like ”horse”, ”flower” or ”dog” we obtained distorted images. The model struggles to comprehend complex text queries and generate diverse images. Nevertheless, we believe that this approach holds great promise; however, it requires a substantial high-quality dataset, which is currently lacking.

3 METHOD

Our approach aims to avoid vector image rasterization to solve the problem of “drawing with strokes” (when vector shapes are unclosed or a lot of them form one homogeneous area) by using transformers to train an autoencoder and a diffusion model to generate image embedding in a latent space. The main task here is to train a highly qualitative autoencoder because the task of training a diffusion model to generate embeddings in a latent space has already been solved (Rombach et al., 2022). To train the autoencoder, we need to collect a dataset large enough to train a model, create augmentations and loss functions for the vector graphics domain and develop an optimal architecture for fast learning.

In Section 3.1, we provide a detailed overview of our data processing. We then discuss in Section 3.2 our approach as a whole, followed by an in-depth examination of its components: the encoding of paths and entire images in Section 3.3, as well as the prediction and decoding of paths based on the internal representation in Section 3.4. Subsequently, we delve into the details of applying the latent diffusion model in Section 3.5 and in Section 3.6 discuss the loss functions implemented in our study.

3.1 Dataset Collection

One of the main problems in the vector graphics domain is the lack of large open collections of data. While Stable Diffusion was trained on the LAION dataset (Schuhmann et al., 2022) consisting of five billion images, papers on the generation of vector images use less than one million images. For example, DeepSVG was trained on the SVG-Icons8 dataset (Carlier et al., 2020) consisting of $100,000$ images.

To train the initial version of our model, we need images with a maximum of 20 paths, because almost all typical vector icons satisfy this condition. We cannot use vectorizers due to the following reasons: (1) they generate too many paths (Dziuba et al., 2023) for simple images (software algorithms (Tian and Günther, 2022)); (2) they are extremely time-consuming (LIVE, ClipGen); (3) our experiments with Im2Vec showed that it is very unstable and is unable to memorize and vectorize images containing more than 10 shapes.

To collect the largest dataset, we searched for images from open sources on the Internet. The most of search results are giant maps, diagrams, and other images containing thousands of paths. They are too complicated and are not of interest to generate. We managed to collect 10M images, most of which were unsuitable for high-quality image generation as they have curved lines, fonts, diagrams, and other clipart. Therefore, we train our autoencoder model in two stages: at first, we use all the collected data (with augmentations) to train the model to operate with vector shapes in a basic way, and on the second stage we employ only high-quality data for the finetuning.

However, the amount of collected data is still not enough to train such a model. To overcome this, we created the following augmentations to expand our dataset:

1. Images mixing (one path from each of the 20 images),
2. Path random shuffling within an image,
3. Adding anchor points on existing paths,
4. Reversing the order of anchor points in paths,
5. Cyclic shift of anchor points in cyclic paths,
6. Random colors change,
7. Adding random noise to coordinates of anchor points.

Most of these augmentations spoil the images but it should help the model to better understand the structure of vector images and avoid overfitting. In addition, we used randomly generated images to enlarge the dataset; their effect is considered in the Discussion section.

The proportions of data augmentations in the dataset were as follows: approximately 80% consisted of augmented images, 5% were random images, and the remaining were original vector images. This balanced approach helps enhance the model’s robustness by providing a diverse set of training examples, which is crucial for improving performance and reducing overfitting.

To fine-tune the model, we additionally collected 0.5M of images from open trusted sources[1]. At the fine-tuning stage, we used only such augmentations that did not change the final image, such as (3), (4), and (5).

3.2 Autoencoder

We follow the DeepSVG (Carlier et al., 2020) double staged encoding and decoding approach. At first, we transform each path of the image to corresponding embeddings using Path Encoder. Then we combine these embeddings using SVG Encoder and obtain the representation of the entire image. The decoding process is made in reverse order. All paths at the encoding and decoding stages are processed independently. We train two transformers, one for encoding and decoding paths (Path Encoder/Decoder) and one for processing the entire image representation (SVG Encoder/Decoder). The general scheme of our model training is presented in Fig. 1a. We call it **VectorWeaver** because the second transformer "weaves" the resulting vector images from curve embeddings.

3.3 Encoding Process

Path Encoder. To encode a path, we must first represent all its anchor point coordinates as tokens. Although the coordinates are float values, a common practice is to discretize them and use with a special dictionary as in (Nash et al., 2020; Esser et al., 2021). Inspired by this approach, we divided each axis by $N = 64$ equal segments. Following this discretization, we trained the embedding dictionary. Namely, we employ two dictionaries, the first one for x coordinates and the second for y coordinates. Besides, we use the float value of the actual coordinate position within its segment not to lose information about the real location of the point (float in-block coordinates).

To create a token, we use the positions (indices) of the anchor points inside its path and the position (index) of this path in the SVG file. These discrete indices are mapped to continuous learnable vectors using dictionaries. All features (discretized coordinates tokens, float in-block values and their positional encodings) are passed through linear layers resulting in the final token.

Additionally, we provide the transformer with information about the filling attribute of this path. To do this, we trained three dictionaries of tokens, each consisting of 255 embeddings. Using them, we translate the *RGB* color into a set of three tokens for the transformer. The overall path encoding scheme can be seen in Fig. 2. Summarizing, in path encoder we use these types of tokens: tokens of segment coordinates, positional encodings of segments, colour encodings (RGB), token of the path position in the whole image.

We use the transformer encoder here to enrich these tokens with information about each other. Therefore, after enriching all the embeddings via the transformer encoder, we combine them into one common embedding path using concatenation and a small multi-linear neural network.

SVG Encoder. Having passed all n paths through the Path Encoder, we can obtain embeddings of all paths. It remains to combine them into a common embedding for the entire image. To do this, we use the transformer decoder (using of decoder is simply an implementation detail for generating two tokens, we could also use an encoder instead of decoder), having previously added positional embeddings to the tokens, as can be seen in Fig. 3.

The SVG encoder predicts two embeddings - μ and σ - that are parameters of the normal distribution from which the final embedding of the image is sampled. We use a standard reparameterization trick, which also helps the diffusion model perform better. The diffusion model removes noise from the embedding of the image. Without reparametrization trick it may not do it perfectly and miss the real latent embedding. However, the reparameterization trick guarantees that some area around each embedding μ is also decoded into the image we need. Therefore, during the training of the diffusion model, we set $\sigma = 0$ in order to predict better central vectors of the latent distribution.

3.4 Decoding Process

SVG Decoder. First of all, we need to obtain information about each path from the embedding of the entire image (sampled from the latent space). Moreover, we need to understand how many paths the original image has.

To solve this problem, the transformer predicts not only the embeddings of the paths but the probability of their existence (technically, we use and train a small network on each generated path embedding for that). We impose a large loss on this part, which severely penalizes the model for throwing out real paths or drawing non-existent ones.

The SVG decoder is a regular transformer decoder, which should solve a one-to-many problem, as presented in Fig. 4a.

Path Decoder. The remaining part is the prediction of the vector image parameters. The embedding

[1]https://freesvg.org/ and https://www.svgrepo.com/collections/

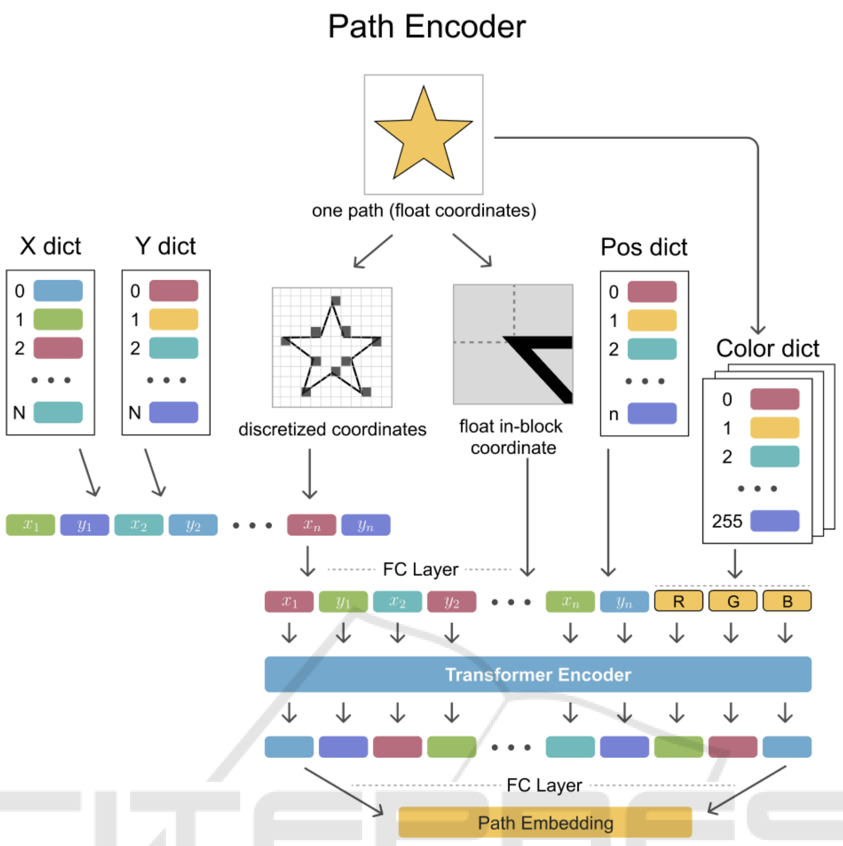


Figure 2: Path encoding scheme. The figure shows which characteristics of the shape we use and how we transform them into an embedding of the entire shape. Namely, we encode to tokens discretized shape coordinates, their float in-block coordinates, shape color, shape position in the entire image. Additionally, we utilize positional encoding for the coordinates.

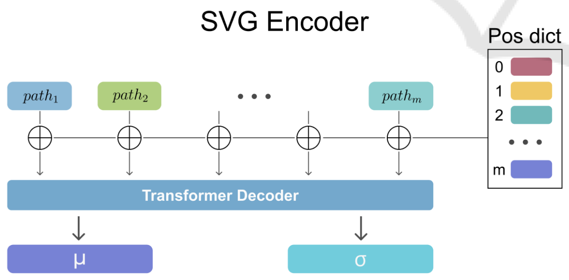


Figure 3: SVG encoding scheme. This is a single transformer that combines several shapes embeddings into parameters of the normal distribution that defines the entire image.

obtained from SVG Decoder is transformed to a fixed set of tokens. We also utilize positional encoding for these tokens. The sequence of tokens is passed to Transformer Encoder in order to enrich them with neighboring tokens. In the end we use a pair of linear layers to obtain information about initial vector image attributes.

3.5 Latent Diffusion

After training the autoencoder, we separately train an unconditional diffusion model which operates in the latent space of the trained VAE. We generated and sampled target embeddings of the images from our dataset using VAE Encoder (Path and SVG Encoders) and trained the diffusion model to predict these embeddings from their noised equivalents. We used a 15-layer DenseNet for this task.

Noteworthy, all diffusion models allow adding generation by prompts easily. For instance, for the purpose of text-based generation, it is enough just to feed text to the input of a network that removes noise from the vector embeddings considering text-features. However, due to the fact that our dataset does not have high-quality text descriptions, text generation using our model remains unresolved. We plan to solve this issue, for example, by expanding our dataset using text descriptions derived from raster Vision-Language models. Therefore, at the moment the generation with our model is unconditional and only one noisy vector is supplied.

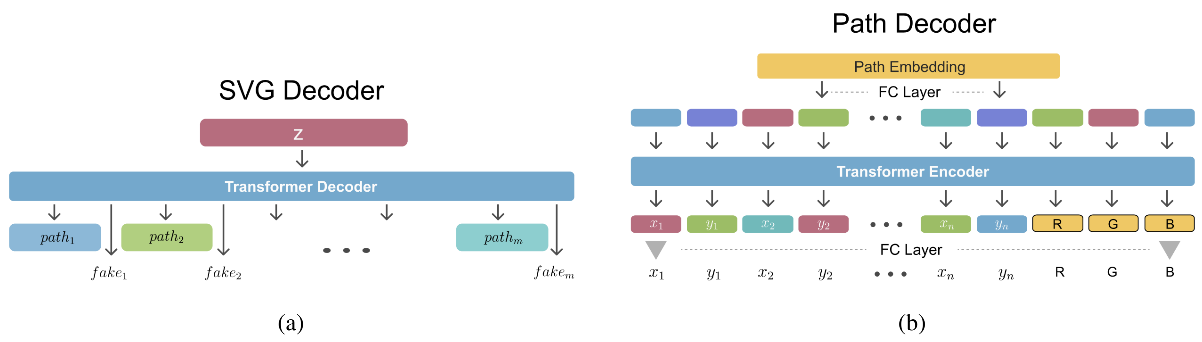


Figure 4: Blocks of Decoding layer. (a) SVG decoding scheme. This block transforms latent code to paths embeddings with flags which indicates if a path has to be visible or not. (b) Path decoding scheme. This layer follows the SVG Decoder block and predicts path parameters (coordinates and colors) of each generated path embedding.

3.6 Losses

As mentioned above, we impose cross-entropy loss $\mathcal{L}_{CE}$ with a large weight on determining the number of paths in the decoder. In fact, this is a very simple task for the model and this loss converges already at initial steps.

To preserve the image structure, we impose MSE loss $\mathcal{L}_{MSE_{x,y}}$ on the coordinates, thereby, forcing the model to give the same output as it receives at the input. However, we found that this approach alone is not good enough and requires improvement, such as the use of other loss functions, because the lines in the generated images were not precise enough.

This effect is simpler to illustrate on raster images. Training a raster convolutional autoencoder with the MSE loss results in blurry images. It occurs because the model is easier to learn a smooth transition instead of sharp color change at the objects boundaries, while the MSE loss remains quite low. Similarly, in the case of vector images, long straight lines can be made a little more crooked, without affecting the loss greatly. Such images are discussed in the Discussion section.

To solve the problem with crooked contours, we used a standard trick, adding an adversarial discriminator. The idea is to add a loss $\mathcal{L}_{adv}$ from the scoring model to the overall loss, discriminator is adversarially trained to determine the input of the autoencoder from its output. The discriminator can easily determine smooth transitions that a given raster image is fake, motivating thus the autoencoder to generate sharper images. Such an approach was also applied in the original Stable Diffusion.

The discriminator architecture is similar to the introduced VAE Encoder (Path and SVG Encoders). The only difference is that it lacks discretization modules to maintain backpropagation through the outputs of the model. Such an encoder has a weaker generalizing ability and learns worse, but it easily detects errors that the MSE loss is blind to.

We also added an MSE loss for color detection, $\mathcal{L}_{MSE_{color}}$, and the KL loss for VAE, $\mathcal{L}_{KL}$. As a result, the total loss is:

$$\mathcal{L} = \lambda \cdot \mathcal{L}_{CE} + \mathcal{L}_{MSE_{x,y}} + \mathcal{L}_{adv} + \mathcal{L}_{MSE_{color}} + \mathcal{L}_{KL}. \quad (1)$$

4 EXPERIMENTS AND RESULTS

4.1 Experimental Evaluation

The training our model, VectorWeaver, was performed on 8 NVIDIA Tesla A100 GPU with 80 GB of memory during 14 days. The fine-tuning was performed using 2 NVIDIA Tesla A100 with 32 GB of memory for one day. The AdamW optimization algorithm was chosen for training with learning rate $\cos(5e^{-6})$. The generation results are presented in Fig. 1b and Fig. 5. The diffusion process is presented in Fig. 7.

We compared our model with DeepSVG trained on our vector images dataset and VectorFusion, which needs no training. We measured basic generation metrics FID and Inception Score (IS). We conducted a survey asking participants to rate the vector images generated by the proposed model, DeepSVG, and VectorFusion on a scale of 1 to 5 (1 stands for completely inappropriate, 5 stands for the perfect fit). 100 assessors took part in the survey. The results of the vector image generation with the three methods are presented in Fig. 6. Moreover, we compared the execution time of these methods. The comparison results are presented in Tab. 1. Additionally, we present examples of image interpolation 8 provided by our model.

Due to the lack of colors and crooked contours, DeepSVG was always considered by assessors as the worst option. Since VectorFusion generates realistic and complex images, assessors sometimes rate it better than the proposed method. However, its gener-

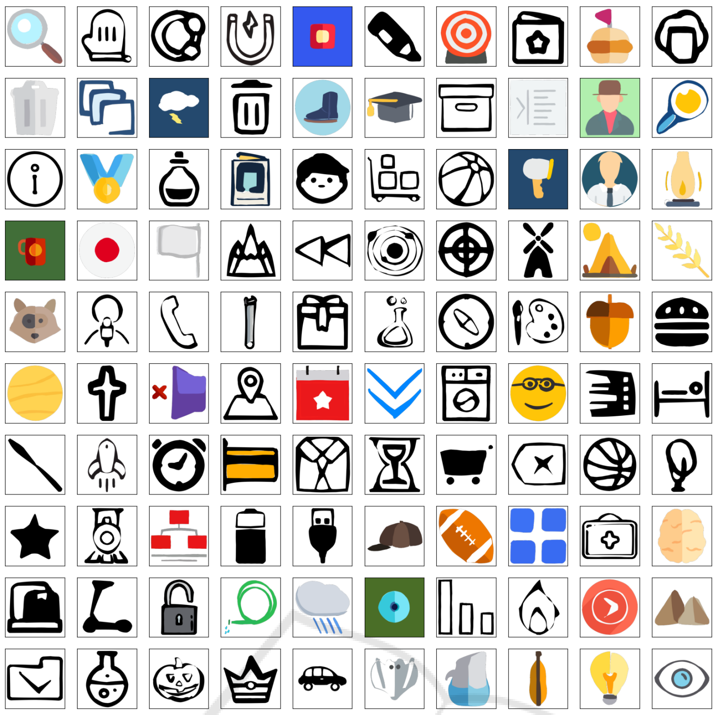


Figure 5: 100 randomly selected images generated by VectorWeaver.

Table 1: Comparison of DeepSVG, VectorFusion, and our method using IS and FID metrics, normalized survey results and algorithms running times.

Method	IS↑	FID↓	Assessors' score↑	Avg time per img, s
DeepSVG	1.53	30.7	0.19±0.03	**0.06**
VectorFusion	1.2	15.1	0.48±0.05	1800
VectorWeaver	**1.8**	**4.3**	**0.79**±0.03	3

ated images often contain artifacts, and the drawing is obtained "with strokes". In 90% of cases, assessors called our generation the best. We also measured the Inception Score. The results of all three models turned out to be quite low because the Inception network was trained on a raster dataset. However, our result was the best apparently due to a smaller number of artifacts and greater simplicity of the images. We obtained similar results with the FID metric.

In comparison with DeepSVG the main advantage of our model is the quality of generation. VectorFusion generates complex vector images with a large number of "strokes", but it takes much time. In paper (Jain et al., 2022), the authors stated that it takes 30 minutes to generate one image. In our case, the diffusion model inference and decoder application requires only 3 seconds on average.

4.2 Discussion

Training Without the Discriminator Loss. At the initial stage, we trained our model without adversarial loss. As seen in Fig. 9, the results are much awrier compared with the ones generated with adversarial loss use. The discriminator sees uneven joints on the lines and detects the image as synthetic.

However, such images seem to be more human-made, they look as if a person drew them in an image editor. This can be used as a generator of images in such a specific style.

Training on Random Images. In Subsection 3.1, we mentioned adding random images to the dataset (the coordinates of the paths are random noise). They help prevent the model's overfitting because it is forced to remember them as well. If the model starts to generalize random images poorly, then the loss on this part of the dataset grows. In practice, 5 percent of the images in our dataset were generated randomly.

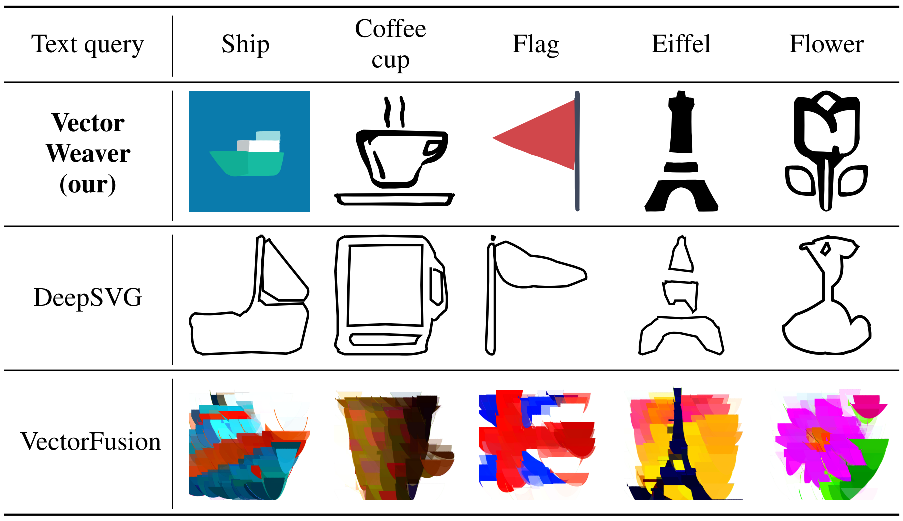


Figure 6: Qualitative comparison of vector image generation results using our VectorWeaver model, DeepSVG, and VectorFusion.

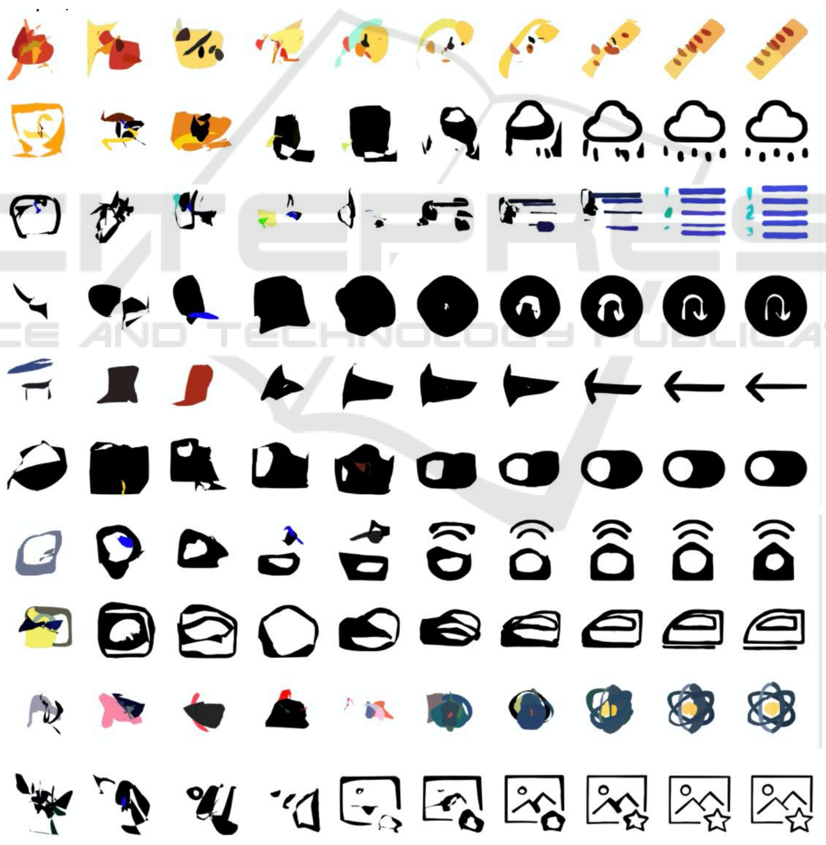


Figure 7: The diffusion process. Each line corresponds to a generation of a single image. We start with random noise in a latent space and remove it until we come to a good result. We produce 1000 steps and show every hundredth.

Limitations. At the moment, our method operates with images consisting of no more than 20 paths defined by no more than 20 anchor points. The selection of the number of paths was dictated by the limited computational resources. However, we believe that our approach would generalize to more compli-

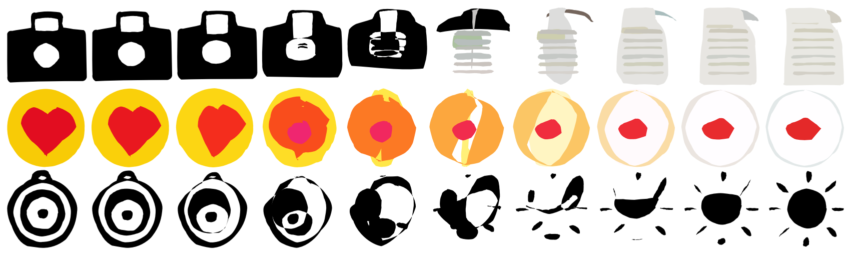


Figure 8: Example of interpolation with our model between two randomly generated images from left to right.

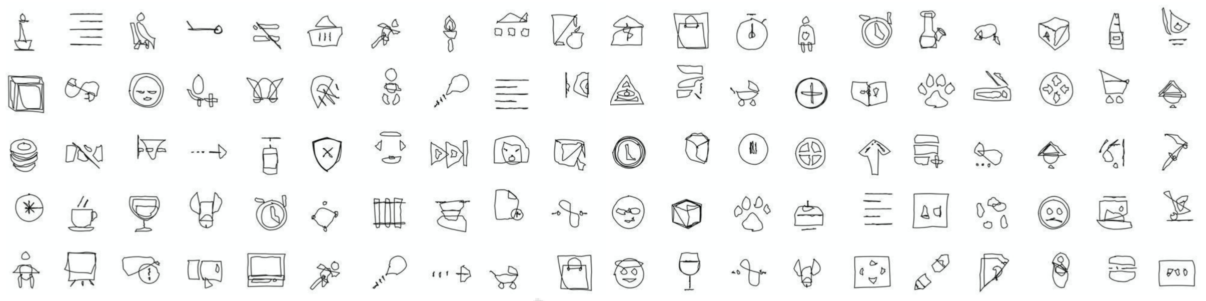


Figure 9: 100 examples of generation without adversarial loss.

cated images if we train the model using more computational resources.

Unfortunately, it is currently difficult to assemble a high-quality and large dataset of text and image pairs, therefore, we made the generation unconditional. However, once such a dataset is assembled, one can easily add text generation by adding CLIP (Radford et al., 2021) model as a text feature extractor for a diffusion model.

5 CONCLUSION

In this work, we have proposed a method to generate vector images based on the diffusion process and stacked transfomer-based autoencoders. Our method, VectorWeaver, operates exclusively within the vector image domain. As a result, it avoids many of the challenges associated with transforming images between raster and vector formats. Moreover, to enlarge training dataset, we have proposed vector image augmentations. We have demonstrated that our model is capable of generating vector images consisting of an arbitrary number of paths and compared it with DeepSVG and VectorFusion. The comparison showed that our model generates smoother vector images and does it 600 times faster than VectorFusion. We trained the proposed model to generate colored icons, but it can be trained to generate more complex images.

In the future, we plan to add more augmentations, make the model text-conditional and to train it with more resources. We expect the proposed architecture to serve as a strong baseline for further research in the little-explored field of vector graphics.

ACKNOWLEDGEMENTS

The research was supported by the ITMO University, project 623097 ”Development of libraries containing perspective machine learning methods”

REFERENCES

Cai, M., Huang, Z., Li, Y., Ojha, U., Wang, H., and Lee, Y. J. (2024). Leveraging large language models for scalable vector graphics-driven image understanding.

Carlier, A., Danelljan, M., Alahi, A., and Timofte, R. (2020). Deepsvg: A hierarchical generative network for vector graphics animation. *CoRR*, abs/2007.11301.

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

Du, R., Chang, D., Hospedales, T., Song, Y.-Z., and Ma, Z. (2023). Demofusion: Democratising high-resolution image generation with no $$$.

Dziuba, M., Jarsky, I., Efimova, V., and Filchenkov, A. (2023). Image vectorization: a review. *arXiv preprint arXiv:2306.06441*.

Efimova, V., Jarsky, I., Bizyaev, I., and Filchenkov, A. (2022). Conditional vector graphics generation for music cover images. *arXiv preprint arXiv:2205.07301*.

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

Frans, K., Soros, L. B., and Witkowski, O. (2021). Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. *CoRR*, abs/2106.14843.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851.

Jain, A., Xie, A., and Abbeel, P. (2022). Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. *arXiv preprint arXiv:2211.11319*.

Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*.

Li, T.-M., Lukáč, M., Michaël, G., and Ragan-Kelley, J. (2020). Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 39(6):193:1–193:15.

Lopes, R. G., Ha, D., Eck, D., and Shlens, J. (2019). A learned representation for scalable vector graphics. *CoRR*, abs/1904.02632.

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

Nash, C., Ganin, Y., Eslami, S. A., and Battaglia, P. (2020). Polygen: An autoregressive generative model of 3d meshes. In *International conference on machine learning*, pages 7220–7229. PMLR.

Nishina, K. and Matsui, Y. (2024). Svgeditbench: A benchmark dataset for quantitative assessment of llm's svg editing capabilities.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020.

Reddy, P., Gharbi, M., Lukac, M., and Mitra, N. J. (2021). Im2vec: Synthesizing vector graphics without vector supervision. *arXiv preprint arXiv:2102.02798*.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*.

Schaldenbrand, P., Liu, Z., and Oh, J. (2022). Styleclipdraw: Coupling content and style in text-to-drawing translation. *arXiv preprint arXiv:2202.12362*.

Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., et al. (2022). Laion-5b: An open large-scale dataset for training next generation image-text models. *arXiv preprint arXiv:2210.08402*.

Shen, I.-C. and Chen, B.-Y. (2021). Clipgen: A deep generative model for clipart vectorization and synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):4211–4224.

Tian, X. and Günther, T. (2022). A survey of smooth vector graphics: Recent advances in representation, creation, rasterization and image vectorization. *IEEE Transactions on Visualization and Computer Graphics*.

Timofeenko, B. A., Efimova, V., and Filchenkov, A. A. (2023). Vector graphics generation with llms: approaches and models. *Zapiski Nauchnykh Seminarov POMI*, 530(0):24–37.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, Y. and Lian, Z. (2021). Deepvecfont: synthesizing high-quality vector fonts via dual-modality learning. *ACM Transactions on Graphics (TOG)*, 40(6):1–15.

Wu, H., Shen, S., Hu, Q., Zhang, X., Zhang, Y., and Wang, Y. (2024). Megafusion: Extend diffusion models towards higher-resolution image generation without further tuning.

Wu, R., Su, W., Ma, K., and Liao, J. (2023). Iconshop: Text-guided vector icon synthesis with autoregressive transformers.

Xing, X., Zhou, H., Wang, C., Zhang, J., Xu, D., and Yu, Q. (2024). Svgdreamer: Text guided svg generation with diffusion model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4546–4555.

Xu, X., Wang, Z., Zhang, E., Wang, K., and Shi, H. (2022). Versatile diffusion: Text, images and variations all in one diffusion model. *arXiv preprint arXiv:2211.08332*.

Xu, Z. and Wall, E. (2024). Exploring the capability of llms in performing low-level visual analytic tasks on svg data visualizations.

Yan, S. (2023). Redualsvg: Refined scalable vector graphics generation. In *International Conference on Artificial Neural Networks*, pages 87–98. Springer.

Yu, J., Xu, Y., Koh, J. Y., Luong, T., Baid, G., Wang, Z., Vasudevan, V., Ku, A., Yang, Y., Ayan, B. K., et al. (2022). Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*.

Zhang, P., Zhao, N., and Liao, J. (2023). Text-guided vector graphics customization.

Zou, B., Cai, M., Zhang, J., and Lee, Y. J. (2024). Vgbench: Evaluating large language models on vector graphics understanding and generation.