

# VecFusion: Vector Font Generation with Diffusion

ANONYMOUS AUTHOR(S)

SUBMISSION ID: 212

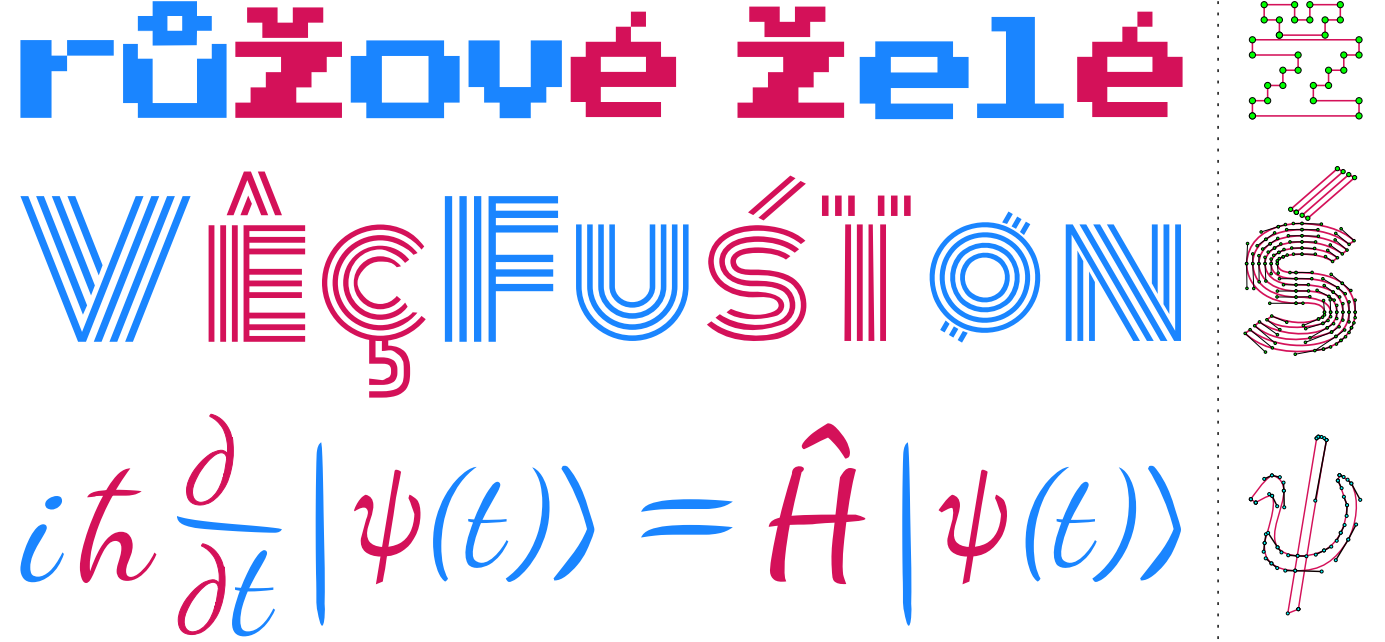


Fig. 1. We present VecFusion, a generative model for vector fonts. VecFusion can automatically generate missing glyphs from a font, shown here. **Blue glyphs** are from the training set and **red glyphs** are generated by our method. On the right, we show generated control points as circles, and curve handles (black) on selected glyphs. Our method generates precise, editable vector fonts whose geometry and control point locations are learned to match the target font style.

We present VecFusion, a new neural architecture that can generate vector fonts with varying topological structures and precise control point positions. We propose a cascaded diffusion model which consists of a raster diffusion model and a vector diffusion model. The raster diffusion model generates low-resolution, rasterized fonts with auxiliary control point information, capturing the global style and shape of the font. The vector diffusion model then synthesizes vector fonts conditioned on the low-resolution raster fonts. To synthesize long and complex curves we propose a transformer-based vector diffusion model and a novel vector representation that enables the modeling of diverse vector geometry and the precise prediction of control points. Our experiments show that, in contrast to previous generative models for vector graphics, our new cascaded vector diffusion model can allow us to generate higher quality of vector fonts with complicated structures and diverse styles.

CCS Concepts: • **Computing methodologies** → **Parametric curve and surface models**; **Neural networks**.

Additional Key Words and Phrases: vector graphics, font generation, denoising diffusion, transformers, neural networks

## 1 INTRODUCTION

Vector fonts are extensively used in graphic design, arts, publishing, and motion graphics. As opposed to raster fonts, vector fonts can be rendered at any resolution without quality degradation, and can be edited by users through intuitive control point manipulation. However, the process of creating high-quality vector fonts

remains challenging and requires significant effort and expertise from designers. Recent approaches [Carlier et al. 2020; Lopes et al. 2019; Wang and Lian 2021] employ VAE or autoregressive models to automatically synthesize vector fonts, but effectively modeling a diverse range of topology structures, glyph variations, and the inherent ambiguity of vector curves pose considerable challenges. Consequently, these approaches frequently generate artifacts and imprecise control point positions, compromising the overall quality and editability of the synthesized fonts.

In this work, we investigate how we can leverage recent advances in *raster* generative models, to design a generative model for *vector* fonts. Such a generative model has a number of real world applications, such as glyph completion, few-shot style transfer, and font style interpolation. However, training vector domain generative models is challenging, as the irregular data structure of vector graphics prevents naive application of commonly used CNN-based architectures. Furthermore, there is an inherent ambiguity in how vector fonts are constructed, in that control points can be placed arbitrarily but still produce the same glyphs. However, designers carefully create control points so that the font can be intuitively edited, and any generated vector fonts should follow similar design.

To address the above-mentioned challenges, we propose a novel two-stage diffusion model, called VecFusion, to generate high-quality vector fonts. Our pipeline is a cascade of a raster diffusion model followed by a vector diffusion model. The raster diffusion model

gradually transforms a noisy image sampled from the Gaussian distribution to the target raster font at low resolution, conditioned on a target glyph and a target font style. It also generates auxiliary control point fields which will guide the next step. The vector diffusion model is conditioned on the output from raster diffusion model, and is trained to “denoise” a noisy vector font representation.

*Contributions.* We introduce a number of contributions. First, we present a two-stage diffusion model for generating high-quality vector fonts: the first stage is a raster diffusion model which generates critical conditioning fields for the vector diffusion model to generate precise vector results. This cascading process allows us to effectively “upsample” low-resolution raster outputs into a vector representation. We introduce a new mixed discrete-continuous representation for each control point, which allows the vector diffusion model can automatically predict the number of control points and paths using this representation, as well as control point position, and we show that diffusion models are able to accurately “denoise” in this representation space. Moreover, to capture long-range dependencies and accommodate the irregular nature of vector representations, we introduce the first transformer-based vector diffusion model and a novel vector representation. Finally, we show that method can synthesize fonts with significantly higher fidelity than state-of-the-art methods e.g., we improve the precision of control point location prediction by almost an order of magnitude when measured as the Chamfer distance between artist-specified control point locations and generated ones in a dataset of diverse Unicode glyphs and fonts.

## 2 RELATED WORK

*Raster font generation.* Generative models based on GANs have achieved impressive results in generating raster fonts. Some approaches [Gao and Wu 2020; Jiang et al. 2019] rely on using hundreds of reference glyphs to fine-tune their network for generating new glyphs. To address the time-consuming collection process of reference glyphs, many approaches learn to generate raster fonts from only a few reference images. The Multi-Content GAN [Azadi et al. 2018] and AGIS-Net [Gao et al. 2019] approaches decompose the artistic font generation into the synthesis of glyph shape and texture. Other approaches [Cha et al. 2020; Kong et al. 2022; Park et al. 2021; Tang et al. 2022] improve the quality of few-shot font generation by employing style representation at the glyph component level. In Srivatsan et al. [2021], the Unicode completion problem is approached using matrix factorization, where character shape and font style are embedded into separate manifolds. These methods can create visually appealing raster fonts across a range of styles. However, they fail to capture detailed curve geometry and do not produce vector outputs. Converting the generated glyph images into high-quality vector fonts still requires significant manual intervention for such approaches.

*Vector graphics generation.* Significant work has been invested in generative modeling of vector graphics. Suveeranont and Igarashi [2010] generated new fonts based on an example glyph by constructing a morphable model from a template font database. Campbell and Kautz [2014] proposed to learn a font manifold that enables interpolation and extrapolation of existing fonts. More recent methods use deep learning-based approaches to synthesize vector fonts.

EasyFont [Lian et al. 2018] learned the Chinese handwriting style by decomposing it into stroke layout and stroke shape. However, this method needs substantial manual annotation and requires hundreds of reference glyphs. Other approaches use VAE [Kingma and Welling 2013] or autoregressive models to generate vector graphics. SketchRNN [Ha and Eck 2017] and Sketchformer [Ribeiro et al. 2020] synthesize sketches with sequence-to-sequence models. SVG-VAE [Lopes et al. 2019] is the first method that attempts to estimate vector graphics parameters for generative tasks. DeepSVG [Carlier et al. 2020] proposed a hierarchical generative network for the generation and interpolation of SVG icons. Im2Vec [Reddy et al. 2021] can generate vector graphics with indirect supervision from raster training images. As shown in DeepVecFont [Wang and Lian 2021], these methods tend to generate vector glyphs with distortion and artifacts. In contrast, DeepVecFont utilizes features of both raster images and vector outlines of reference glyphs to synthesize target glyphs. In a concurrent work with ours, DeepVecFont-v2 [Wang et al. 2023b] improves the quality of synthesis with a transformer architecture. Although this method can synthesize visually pleasing vector fonts, effectively modeling a diverse distribution of glyphs and topologies remains a challenge. The method only supports a limited number of glyphs (52 characters). Moreover, during inference, as an additional ad-hoc step, it needs to generate several candidate vector glyphs until it find ones matching synthesized raster glyphs used as reference.

To address the challenges in vector field design, we leverage diffusion models [Ho et al. 2020] which excel at modeling diverse and complex data distributions. Concurrently with our work, ChiroDiff [Das et al. 2023] and SketchKnitter [Wang et al. 2023a] proposed to synthesize vector sketches with CNN-based and RNN-based vector diffusion models respectively. In contrast to these methods, our method utilizes a transformer-based vector diffusion model to handle long-range dependencies of complex vector glyphs. Furthermore, our two-stage approach and novel vector representation enable us to achieve precise Bezier curve prediction on challenging artist-designed font datasets. As we demonstrate in our results, compared to Chirodiff, we improve the prediction of control point location by almost an order of magnitude.

*Cascaded diffusion models.* Cascaded diffusion models [Ho et al. 2022b] have proven to be effective in achieving high-fidelity synthesis across various domains. In the image domain, Imagen [Saharia et al. 2022] and eDiff-I [Balaji et al. 2022] employ a cascade of diffusion models, starting with a standard diffusion model at the lowest resolution and progressively adding higher resolution details using super-resolution diffusion models. Similarly, in the video domain, [Blattmann et al. 2023; Ho et al. 2022a] utilize a base video generation model combined with spatial and temporal video super-resolution models to generate high-quality videos. In the realm of 3D, [Hui et al. 2022; Lin et al. 2022] follow a cascaded approach where a coarse 3D model is generated initially, and then additional high-frequency details are added based on the output of the first stage. Drawing inspiration from these successes, we introduce a cascaded diffusion model for vector font generation. Our method harnesses the power of this cascading strategy to synthesize high-quality vector glyphs.

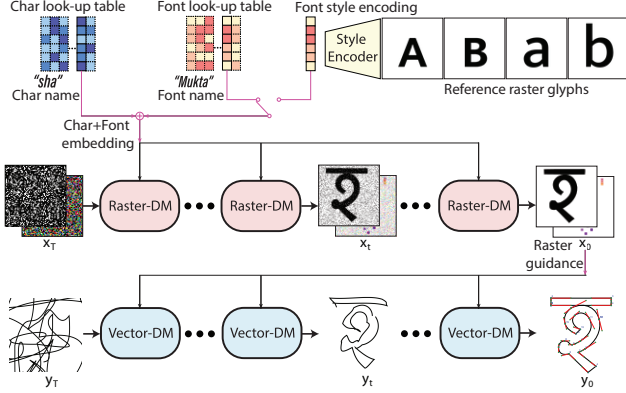


Fig. 2. Overview of the VecFusion’s cascade diffusion pipeline. Given a target character and font embedding, our raster diffusion stage (“Raster-DM”) produces a raster image representation of the target glyph in a series of denoising steps starting with a Gaussian noise image. The raster image serves as guidance to our next stage (vector diffusion stage - “Vector-DM”), which produces the final vector representation of the glyph also in a series of denoising steps starting with a Gaussian noise curve representation.

### 3 METHOD

**Overview.** The goal of VecFusion is to automatically generate vector graphics representations of glyphs. The input to our model is the Unicode identifier for a target character, also known as *code point*, and a target font style. The target font style can be specified in the form of a few representative raster images of other glyphs in that style or simply by the font style name. Figure 2 shows an example of the generated vector representation for the glyph corresponding to the input target letter “sha” of the Devanagari alphabet and the target font style “Mukta”. Our method is trained once on a large dataset of glyphs from various font styles. Once trained, it can generate glyphs not observed during training. Our model has several applications: generate missing glyphs in incomplete fonts, synthesize novel fonts by transferring the style of a few exemplar images of glyphs, or interpolate font styles.

**Output vector representation.** The generated vector representation for a glyph is in the form of ordered sequences of control points in cubic Bézier curve paths commonly used in vector graphics. Control points can be repeated in the generated sequences to manipulate the continuity of the vector path. Our method learns to generate an appropriate number of vector paths, control points, and point repetitions tailored to each character and font style. In addition, it learns the proper ordering of control points for each path, including where first and last control points are placed, since their placement patterns often reflect artist’s preferences.

**Pipeline.** Our method consists of a two-stage cascade. In the first stage (raster diffusion model, or in short “Raster-DM”, Figure 2), our method extracts embeddings for the target character code point and font style, then initiates a reverse diffusion process to generate a raster image conditioned on these embeddings. The generated raster image captures the shape and style of the target glyph at low resolution. Additionally, we generate an auxiliary set of *control point fields* encoding information for control point location, multiplicity,

and ordering. In the second stage (vector diffusion model or “Vector-DM”, Figure 2), our method proceeds by synthesizing the vector format capturing fine-grained placement of control points guided by the control point fields generated in the first stage. We observed that this two-stage approach results in generating higher-fidelity fonts compared to using diffusion in vector space directly or without any guidance from our control point fields. In the next sections, we discuss our raster and vector diffusion stages in more detail.

#### 3.1 Raster diffusion stage

Given a target character identifier and font style, the raster diffusion stage creates a raster image  $x_0$  encoding information about the target glyph in pixel space (Figure 2, “Raster-DM”). This is performed through a diffusion model that gradually transforms an image  $x_T$  sampled from a unit Gaussian noise distribution towards the target raster image  $x_0$  in a series of  $T$  denoising steps. At each step  $t = 1 \dots T$ , a trained neural network executes the transition  $x_t \rightarrow x_{t-1}$  by predicting the noise content to be subtracted from the image  $x_t$ . This denoiser network is conditioned on learned embeddings of the input character identifier and font style. In the following paragraphs, we explain the embeddings that encode the input character codepoint and font style, the target raster image, the denoiser network, and finally the training and inference process of this stage.

**Character identifier embeddings.** Inspired by similar approaches in NLP to represent words [Vaswani et al. 2017], we create a one-hot vector representation for all unique character codepoints available in our dataset. Given a target character’s codepoint, its one-hot vector representation is mapped to a continuous embedding  $g$  through a learned look-up table. The look-up table stores embeddings for all codepoints available in our dataset and retrieves them using the one-hot vector as indices.

**Font style embedding.** To obtain the font style embedding, we experimented with two approaches. In the first approach, we create a one-hot vector representation for all font styles available in our dataset. Given a target font style, its one-hot vector is mapped to a continuous embedding  $f$  through a learned look-up table as above. As an alternative approach, we extract the font embedding from input images provided as examples for a particular font style. To this end, we use the image encoder proposed in DeepVecFont [Wang and Lian 2021]: given a set of reference images of glyphs, we concatenate them channel-wise and pass them through a convnet. Its architecture follows DeepVecFont [Wang and Lian 2021] – more details are provided in the supplement.

**Target raster image.** The target  $x_0$  produced in the raster diffusion stage is a  $N \times N$  image consisting of the following channels: (a) the first channel is composed of an image representing a grayscale rasterized image of the target glyph (Figure 3, top). (b) the rest of the channels store *control point fields* (Figure 3, bottom), whose goal is to encode information about the control point location, multiplicity, and ordering. During training, this control point field is created by rendering each control point as a Gaussian blob centered at the 2D coordinates  $(x, y)$  of the control point. The coordinates are normalized in  $[0, 1]^2$ . We also modulate the color of the blob based on (a) the index of the control point in the sequence of control points



of its vector path (e.g., first, second, third etc control point), and (b) its multiplicity. A look-up function is used to translate the ordering indices and multiplicities of control points to color intensities. In our implementation, we use 3 channels for this control point field, which can practically be visualized as an RGB image (Figure 3, bottom). These channels are simply concatenated with the raster image of the glyph, forming a 4-channel image.

**Raster denoiser.** The denoiser is formulated as a UNet architecture [Dhariwal and Nichol 2021]. The network takes the 4-channel image  $\mathbf{x}_t$  as input and is conditioned on the embedding of time step  $t$ , the character’s codepoint embedding  $\mathbf{g}$  and font style embedding  $\mathbf{f}$ . Following [Rombach et al. 2022], we add the three embeddings and input it to each residual block in the UNet. The denoiser network predicts the per-channel noise component of the input image, which is also a 4-channel image. For more details about the architecture of the raster denoiser, we refer to the supplement.

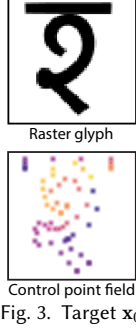
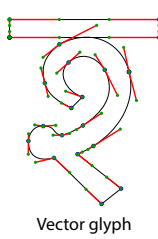


Fig. 3. Target  $\mathbf{x}_0$

**Training loss.** The network is trained to approximate an optimal denoiser under the condition that the images  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$  are created by progressively adding Gaussian noise to the image of the previous step [Ho et al. 2020]:  $q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{(1 - \beta_t)}\mathbf{x}_{t-1}, \beta_t \mathbf{I})$ , where  $\beta_t$  represents the variance of the Gaussian noise added at each step. The image  $\mathbf{x}_T$  converges to a unit Gaussian distribution as  $T \rightarrow \infty$ , or practically a large number of steps [Ho et al. 2020]. Following [Ho et al. 2020], we train the denoiser network with the training objective  $\|\epsilon(\mathbf{x}_t, t, \mathbf{f}, \mathbf{g}) - \epsilon\|^2$  i.e. the mean-squared error loss between the added training noise  $\epsilon$  at each step and the predicted noise  $\epsilon(\mathbf{x}_t, t, \mathbf{f}, \mathbf{g})$  from the network. The loss is used to train both the denoiser as well as the look-up tables.

**Inference.** At test time, given sampled unit Gaussian noise  $\mathbf{x}_T$ , a target character embedding  $\mathbf{g}$  and font embedding  $\mathbf{f}$ , the network is successively applied in  $T$  steps to generate the target raster image.

**Implementation details.** In all our experiments, we use the following hyperparameters. Following [Nichol and Dhariwal 2021], we set the number of diffusion steps  $T$  to 1000 and used cosine noise schedule in the forward diffusion process. We used the AdamW optimizer [Loshchilov and Hutter 2017] with learning rate  $3.24 \cdot 10^{-5}$ . The feature embeddings for character identifiers and font styles are set to be 896-dimensional. The control points are rendered as Gaussian blobs with radius of 2 pixels. The raster image resolution is set to  $64 \times 64$ . Lower resolutions cause increasing overlaps between the rendered blobs, making the control point field more ambiguous. Increasing the resolution increases the computational overhead for the raster denoiser. The above resolution represented a good trade-off, as we practically found in our experiments. As mentioned above, we use 3 channels to encode control point ordering and multiplicity as colors. We practically observed that 3 channels were enough to guide the vector diffusion stage. Depending on the dataset, fewer channels could be used instead e.g., in cases of glyphs with few control points, or no multiplicities. In general, these hyperparameters can be adjusted for different vector graphics tasks – our main point



Vector glyph

	path index	grid cell X index	grid cell Y index	$\Delta x$	$\Delta y$
CP1	0 0 0	1 1 0 0	0 1 0 0	0.34	0.47
CP2	0 0 0	0 1 1 0	0 1 0 0	0.15	0.20
⋮	⋮	⋮	⋮	⋮	⋮
CP21	0 0 1	1 0 1 0	1 1 1 0	0.03	0.50
⋮	⋮	⋮	⋮	⋮	⋮
NULL	1 1 1	0 0 0 0	0 0 0 0	0.00	0.00
NULL	1 1 1	0 0 0 0	0 0 0 0	0.00	0.00

Target representation

Fig. 4. Target tensor representation  $\mathbf{y}_0$ . Our vector diffusion model “denoises” this tensor representation which includes both path membership and spatial position for control points. The discrete values (path membership, grid cell coordinates) are denoised in the continuous domain and then discretized. The control point locations are computed from the predicted grid cell coordinates plus continuous displacements ( $\Delta x, \Delta y$ ) from them.

is that the raster images and fields are useful as guidance to produce high-fidelity vector fonts, as shown in our ablation.

### 3.2 Vector diffusion stage

Given the raster image generated in the previous stage, the vector diffusion stage creates a tensor  $\mathbf{y}_0$  representing the target glyph in vector graphics format (Figure 2 “Vector-DM”). The reverse diffusion process gradually transforms a noise tensor  $\mathbf{y}_T$  sampled from a unit Gaussian noise distribution towards a tensor  $\mathbf{y}_0$  in a series of denoising steps. In this domain, the noise represents noise on the *spatial position* and *path membership* of the control points, rather than the intensity of the pixel values as in the raster domain. In the following paragraphs, we explain the target tensor representation, the denoiser network, the training and inference of this stage.

**Target tensor.** The target tensor  $\mathbf{y}_0$  is a  $M \times D$  tensor (Figure 4), where  $M$  represents an upper bound to the total number of control points a glyph can have. Each entry in the tensor contains a  $D$ -dimensional representation of a control point. Specifically, each entry stores the following information:

(a) the index of the vector path the control point belongs to i.e. its path membership. During training, each vector path is assigned a unique index. Since the vector paths can be re-ordered arbitrarily without changing the resulting glyph, to reduce unnecessary variability during learning, we lexicographically sort vector paths using the coordinates of their control point closest to the top-left corner of the glyph raster image as sorting keys. Following [Chen et al. 2023], the resulting sorted path index is converted to binary bits. For each control point entry, we store the binary bits of its vector path. A null entry (i.e., all-one bits) is reserved for entries that do not yield control points – in this manner, we model vector fonts with a varying number of control points and paths.

(b) the index of the grid cell containing the control point. We define a coarse  $P \times P$  grid over the image, with  $P^2$  corresponding grid cell centroids. We assign each control point to the grid cell that has the closest centroid. Similar to path membership, the grid cell index is converted to binary bits. For each control point entry, we store the binary bits of its assigned grid cell.

(c) the continuous coordinates of the control point expressed relative to the center of the grid cell it belongs to. These are two continuous values capturing the location of each control point. We found that

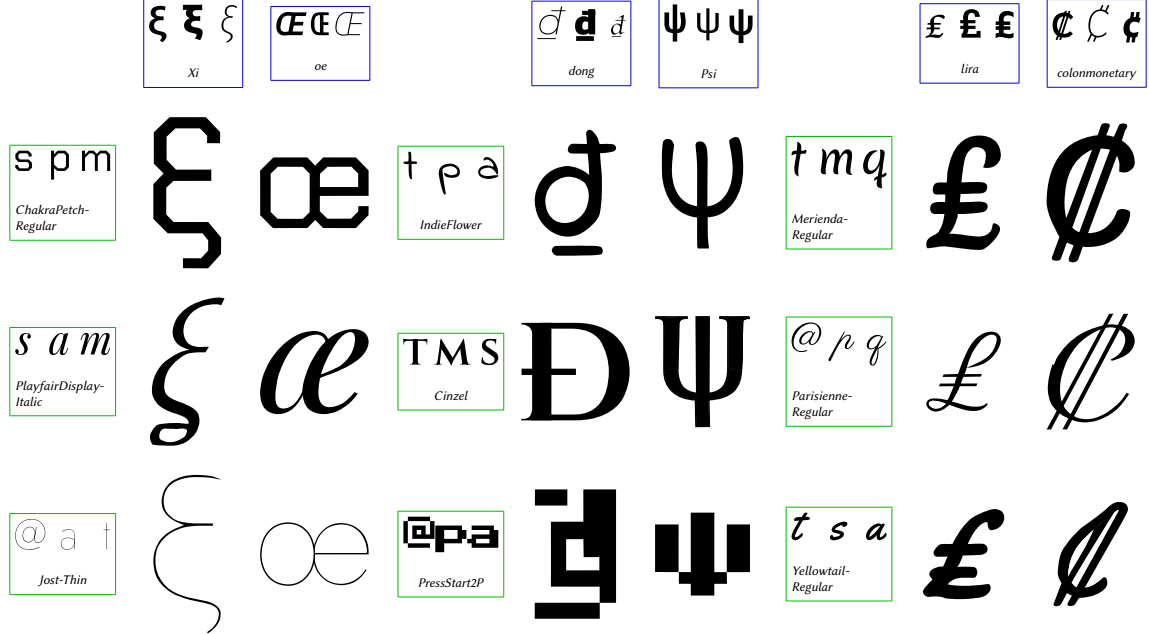


Fig. 5. Missing glyphs generated by our method on the Google Fonts dataset. On the left (green box), we show exemplar training glyphs for a target font. On the top (blue box), we show the target character in other font styles. In the center, we show generated glyphs for the target character and font.

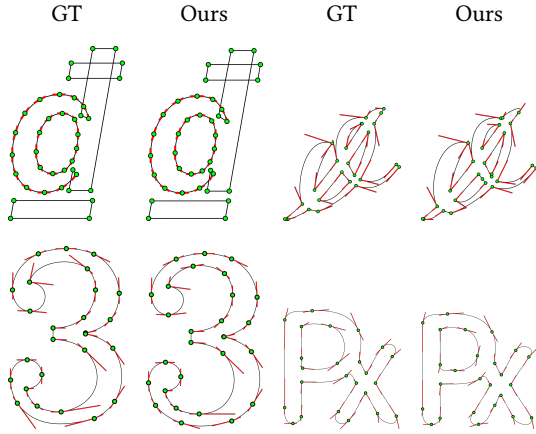


Fig. 6. Predicted control point distributions for our generated cubic Bézier curves. Green dots are control points and red lines are handles defined at these control points. Our model produces control point distributions similar to artist-specified ones (“ground-truth”/“GT” column).

capturing control point locations relative to cell centers achieves the best performance. Since the generated control point field (approximately) highlights regions storing control points, mapping the control point field to discrete cell indices plus small continuous residuals, or displacements, is an easier task and reduces the continuous coordinate variability needed to be captured by the model.

**Denoiser.** The denoiser for this stage is formulated as an encoder-only transformer [Devlin et al. 2018], which takes the tensor  $y_t$  as input and is conditioned on the embedding of time step  $t$ , and the generated raster image  $x_0$  from the raster diffusion model. We use a ConvNet to encode the raster image  $x_0$  into high-dimensional

features, which are input to the transformer via cross-attention similar to [Rombach et al. 2022]. The transformer predicts the noise content as a  $M \times D$  tensor at each step. For more details about the architecture, we refer to the supplemental.

**Training loss.** We train the denoiser network according to mean-squared error loss between training noise and predicted one at sampled time steps:  $\|\epsilon(y_t, x_0, t) - \epsilon\|^2$ .

**Inference.** At test time, given a sampled tensor  $y_T$  from unit Gaussian noise and a generated raster image  $x_0$  of the previous stage, the denoiser network is applied in a series of  $T$  steps to generate the target tensor  $y_0$ . Following the Analog Bits approach [Chen et al. 2023], the discrete binary bits in the target tensor representation are modeled as real numbers. These are simply thresholded to obtain the final binary bits at inference time. Given the predicted path membership, we create a set of vector paths according to the largest generated control path index number. Each non-null entry in the generated tensor yields a control point. The control points are implicitly ordered based on their entry index. The location of the control point is defined as the coordinate center of the assigned cell in the generated tensor plus the predicted relative displacement. Given this generated information, we directly reconstruct the vector paths without requiring any further refinement or post-processing.

**Implementation details.** In our implementation, we set the upper bound for the number of control points to  $M = 256$ , which was sufficient for the datasets we experimented with. We use 3 bits to represent the path membership, which can support up to 7 distinct vector paths. This was also sufficient for the datasets in our experiments. We set  $P$  to 16, resulting in 256 grid cells which can be represented by 8 binary bits. Together with the two-dimensional relative displacement, the final dimension of our target tensor  $D$  is

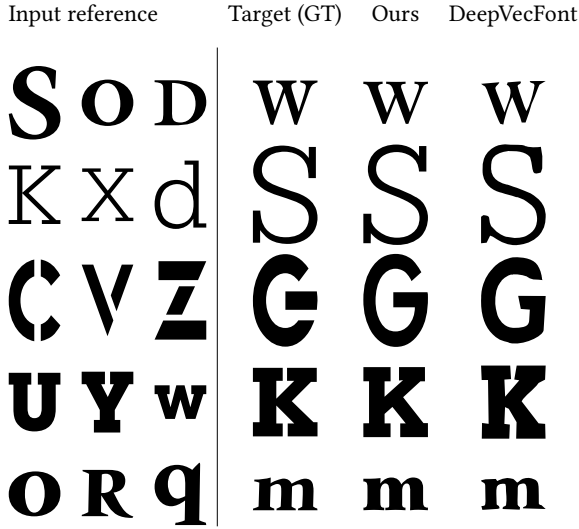


Fig. 7. Few-shot style transfer results. On the left, we show the raster images of glyphs used as a reference for a novel font style. On the right, we show the artist-made (“ground-truth”/“GT”) glyphs, our generated ones, and the ones produced by DeepVecFont [Wang and Lian 2021] for that style.

Method	$L_1 \downarrow$	CD $\downarrow$	#cp diff $\downarrow$	#vp diff $\downarrow$
Ours	<b>3.47</b>	<b>0.16</b>	<b>3.05</b>	<b>0.03</b>
Ours (cont. coord. only)	5.07	0.18	3.30	<b>0.03</b>
Ours (no cp fields)	4.00	0.60	12.46	0.13
Ours (vector only)	4.13	0.68	9.36	0.11
ChiroDiff [Das et al. 2023]	11.25	1.66	56.37	0.77

Table 1. Missing glyph generation evaluation. CD is scaled by  $10^2$ .

13 in our experiments. Similar to our raster diffusion model, we set the number of diffusion steps  $T$  to 1000, used cosine noise schedule, and the AdamW optimizer [Loshchilov and Hutter 2017] with learning rate  $3.24 \cdot 10^{-5}$ . During testing, we use the DDPM sampler [Ho et al. 2020] with 1000 steps. *Our source code will be publicly released.*

## 4 RESULTS

In this section, we present experiments in three different application scenarios for our method. In the first scenario, we address the problem of *missing glyph generation* for a font (Section 4.1). Many users often experience a frustrating situation when they select a font they prefer, only to discover that it lacks certain characters or symbols they wish to utilize. This issue is particularly prevalent when it comes to non-Latin characters and mathematical symbols. As a second application scenario, we apply our method for *few-shot font style transfer* (Section 4.2), where the desired font is specified in the form of a few exemplar images of glyphs, and the goal is to generate other glyphs in the same font. Finally, we discuss *interpolation of font styles* (Section 4.3) i.e., generate glyphs whose style lies in-between two given fonts.

### 4.1 Generation of missing Unicode glyphs

Existing public datasets or benchmarks to evaluate glyph generation are limited to a specific alphabet (e.g., Latin). Below we discuss a

new dataset for evaluating generation of glyphs across different languages, math symbols, and other signs common in the Unicode standard. Then we discuss comparisons, ablation study, metrics for evaluation, and results.

**Dataset.** We collected a new dataset of 1424 fonts from Google Fonts. The dataset contains 247K glyphs, covering 577 distinct Unicode glyphs in various languages (e.g., Greek, Cyrillic, Devanagari), math symbols and other signs (e.g. arrows, brackets, currency). We randomly partition the dataset into 237K-5K-5K glyphs for training, validation, and testing respectively <sup>1</sup>.

**Comparison.** We compare with the concurrent work of “ChiroDiff” [Das et al. 2023], which applies diffusion models to generate Kanji characters as polylines. Their method uses a set-transformer [Lee et al. 2019] to obtain a latent embedding from a 2D point set as the input condition. We replace their input condition to their diffusion model with the embeddings of characters and fonts using look-up tables, as done in our raster diffusion model. We trained and tuned their method, including the embeddings, to predict Bézier curve control points using our dataset, as in our method.

**Ablation.** In addition, we evaluate the following alternative variants of our method: (a) **Vector only:** in this ablation, we remove the raster diffusion model and use the vector diffusion model only – in this case, the font and character embeddings are used as input conditions to the vector diffusion model. (b) **No control point fields:** we remove the RGB control point field from the target raster image of our raster diffusion model – in this case, we condition the vector diffusion model only on the single-channel rasterized image of the glyph. (c) **Predict continuous coordinates only:** in the vector diffusion model, instead of predicting discrete grid cell indices plus displacements relative to grid cell centers, we directly predict the absolute coordinates  $x$  and  $y$  per control point.

**Evaluation metrics.** We compare the generated glyphs with ones designed by artists in the test split. We use the following metrics: (a)  $L_1$ : we compare the image-space absolute pixel differences of glyphs when rasterized. We use the same rasterizer for all competing methods and variants. This image reconstruction error was also proposed in [Wang and Lian 2021] for glyph evaluation. (b)  $CD$ : we measure the bidirectional Chamfer distance between artist-specified control points and generated ones. (c)  $\#cp\ diff$ : we measure the difference between the number of artist-made control points and predicted ones averaged over all paths. (d)  $\#vp\ diff$ : we measure the difference between the number of artist-specified vector paths and predicted ones.

For all the above measures, we report the averages over our test split. We propose the last three metrics for comparing glyphs in terms of the control point and path characteristics, which are more relevant in vector font design.

**Quantitative Results.** Table 1 shows the results for ChiroDiff and the alternative variants of our method based on the above evaluation metrics. The full version of our method outperforms ChiroDiff and our reduced variants on all evaluation metrics. Our method reduces the  $L_1$  reconstruction error by a factor of  $\sim 3.2$  relative to ChiroDiff,

<sup>1</sup>We will provide the dataset and splits upon acceptance.

Method	$L_1 \downarrow$	CD $\downarrow$	#cp diff $\downarrow$	#vp diff $\downarrow$
Ours	<b>6.28</b>	<b>0.61</b>	<b>11.24</b>	<b>0.15</b>
DeepVecFont [Wang and Lian 2021]	7.41	0.68	16.10	0.16

Table 2. Few-shot font style transfer evaluation. CD is scaled by  $10^2$ .

and almost an order of magnitude in terms of Chamfer distance. We also find that the number of control points and paths our method predicts matches the ground truth numbers much more closely relative to Chirodiff. Using vector diffusion alone is still better than Chirodiff, yet has  $\sim 4.2$  larger Chamfer distance, and higher  $L_1$  error than our full solution. Using only the raster image as input to our vector diffusion model without the control point fields (“no cp field” variant) degrades the performance in terms of Chamfer distance and difference in numbers of control points. Predicting continuous coordinates only (“cont. coord. only” variant) also increases  $L_1$  error. This indicates that our two-stage approach, and the mixed discrete-continuous representation of vector paths along with the control point fields are all important to achieve high performance.

**Qualitative Results.** Figure 8 shows qualitative comparisons for missing glyph generation. Compared to our method, we observe that ChiroDiff produces imprecise control points and curve structure, resulting in significant distortions and artifacts in the synthesized glyphs. We also observed degraded results in all alternative variants of our method in terms of misplaced or skipped control points. Figures 1, 5, 9, and 11 show additional results of missing glyph generation for our method on the Google Font dataset for various target fonts and characters. Figure 6 shows the control point distribution and resulting handles for the Bézier curves produced by our method. In the supplementary material, we include one more qualitative comparison: instead of using the vector diffusion model, we use a vectorizer approach [Bessmeltsev and Solomon 2019] on the rasterized font image produced by the raster diffusion stage. We also tried upsampling the  $64 \times 64$  output raster image to  $256 \times 256$  using ESRGAN [Wang et al. 2021] before passing it to this vectorizer. In both cases, this approach often failed to produce coherent curve topology and structure.

## 4.2 Few-shot font style transfer

For this application, we compare with DeepVecFont [Wang and Lian 2021], which previously demonstrated few-shot font style transfer. To perform the comparison, we use the dataset proposed in the DeepVecFont paper. The dataset includes 52 lowercase and uppercase Latin characters represented in various font styles – there are total 8,035 training fonts and 1,425 test ones. Each test case contains 4 reference characters from a novel font (unobserved during training). The reference characters are available in both vector and raster format. Methods are supposed to transfer this novel font style to testing characters. Here, we use the same convnet encoder architecture proposed in DeepVecFont to extract font style embeddings for our method from the 4 reference characters in each test case, but train it with the diffusion loss. We note that DeepVecFont requires as additional input condition the vector representation of the reference characters, while our method only uses the raster reference images. From this aspect, our method can be considered as disadvantaged in this comparison.

**Quantitative results.** Table 2 shows numerical comparisons based on the same evaluation metrics as in the missing glyph generation application. Despite the disadvantage, our method still outperforms DeepVecFont on all metrics: we achieve relative reduction of  $\sim 15\%$  in terms of  $L_1$  error,  $\sim 10\%$  in terms of Chamfer distance, and  $\sim 30\%$  in terms of difference in the number of control points, while the performance is similar in terms of number of vector paths.

**Qualitative results.** Figure 7 demonstrates font style transfer results. We observed that DeepVecFont tends to succeed in capturing the font style of the reference glyphs, yet still often produces subtle distortions in the vector paths (e.g., see top of character “W” and “S”, bottom of “m”, right side of “K”). Our method produces results that match the style of reference glyphs with less artifacts, even in challenging areas to vectorize precisely such as sarifs. There are also cases where both methods fail to capture well the reference style (e.g., see character “G”), possibly due to the limitations in the convnet encoder to capture fine-grained details and subtle patterns in the input reference glyphs.

## 4.3 Font style interpolation

Finally, we experimented with interpolating two given font styles. To perform interpolation, we first obtain the font embeddings from our trained look-up table, then perform linear interpolation of these embeddings. Our diffusion model is then conditioned on the interpolated embedding vector for font style. We demonstrate qualitative results in Figure 10. Our results smoothly interpolates artistic properties of the source and target font, such as the variable stroke width and local curvature, while preserving structural and topological properties of the glyphs e.g., their genus.

## 5 CONCLUSION

We presented a generative model of vector fonts. We show that a cascade of a raster and vector diffusion model can overcome the challenges of neural parametric curve prediction, and outperforms the naive application of vector diffusion models.

**Limitations.** Our method has a number of limitations. First, we require vector paths as supervision and cannot take advantage of raster images as additional supervision that may be available. A larger cascade could be used to generate low-to-high resolution raster images, or low-to-high resolution vector paths. The resolution of raster images, control point field parameters (e.g., blob size), the upper bound in the number of control points and paths, or the grid used for the vector font representation may need to be tuned differently for other vector graphics tasks.

**Future work.** Generative models of images in the raster domain have been successfully used to learn complex priors about the visual world that can be used in various tasks such as inpainting and 3D tasks. We believe that a generative model for vector fonts could also have additional downstream applications, such as automatic completion of vector logos or icons, or be extended to produce 3D parametric curves or surfaces.



## REFERENCES

- Samaneh Azadi, Matthew Fisher, Vladimir G Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. 2018. Multi-content gan for few-shot font style transfer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7564–7573.
- Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. 2022. ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324* (2022).
- Mikhail Bessmeltsev and Justin Solomon. 2019. Vectorization of Line Drawings via Polyvector Fields. *ACM Trans. Graph.* 38, 1, Article 9 (Jan. 2019), 12 pages.
- Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. 2023. Align your Latents: High-Resolution Video Synthesis with Latent Diffusion Models. *arXiv preprint arXiv:2304.08818* (2023).
- Neill DF Campbell and Jan Kautz. 2014. Learning a manifold of fonts. *ACM Transactions on Graphics (ToG)* 33, 4 (2014), 1–11.
- Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems* 33 (2020), 16351–16361.
- Junbum Cha, Sanghyuk Chun, Gayoung Lee, Bado Lee, Seonghyeon Kim, and Hwalsuk Lee. 2020. Few-shot compositional font generation with dual memory. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX* 16. Springer, 735–751.
- Ting Chen, Ruixiang ZHANG, and Geoffrey Hinton. 2023. Analog Bits: Generating Discrete Data using Diffusion Models with Self-Conditioning. In *The Eleventh International Conference on Learning Representations*.
- Ayan Das, Yongxin Yang, Timothy Hospedales, Tao Xiang, and Yi-Zhe Song. 2023. ChiroDiff: Modelling chirographic data with Diffusion Models. In *International Conference on Learning Representations*.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems* 34 (2021), 8780–8794.
- Yue Gao, Yuan Guo, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. 2019. Artistic glyph image synthesis via one-stage few-shot learning. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–12.
- Yiming Gao and Jiangqin Wu. 2020. Gan-based unpaired chinese character image translation via skeleton transformation and stroke rendering. In *proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 646–653.
- David Ha and Douglas Eck. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017).
- Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. 2022a. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303* (2022).
- Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. *arXiv preprint arxiv:2006.11239* (2020).
- Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. 2022b. Cascaded Diffusion Models for High Fidelity Image Generation. *J. Mach. Learn. Res.* 23, 47 (2022), 1–33.
- Ka-Hei Hui, Ruihui Li, Jingyu Hu, and Chi-Wing Fu. 2022. Neural wavelet-domain diffusion for 3d shape generation. In *SIGGRAPH Asia 2022 Conference Papers*. 1–9.
- Yue Jiang, Zhouhui Lian, Yingmin Tang, and Jianguo Xiao. 2019. Sfont: Structure-guided chinese font generation via deep stacked networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 4015–4022.
- Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).
- Yuxin Kong, Canjie Luo, Weihong Ma, Qiyuan Zhu, Shenggao Zhu, Nicholas Yuan, and Lianwen Jin. 2022. Look Closer to Supervise Better: One-Shot Font Generation via Component-Based Discriminator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13482–13491.
- Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. PMLR, 3744–3753.
- Zhouhui Lian, Bo Zhao, Xudong Chen, and Jianguo Xiao. 2018. EasyFont: a style learning-based system to easily build your large-scale handwriting fonts. *ACM Transactions on Graphics (TOG)* 38, 1 (2018), 1–18.
- Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. 2022. Magic3D: High-Resolution Text-to-3D Content Creation. *arXiv preprint arXiv:2211.10440* (2022).
- Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A learned representation for scalable vector graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7930–7939.
- Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).
- Alexander Quinn Nichol and Prafulla Dhariwal. 2021. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*. PMLR, 8162–8171.
- Song Park, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. 2021. Multiple heads are better than one: Few-shot font generation with multiple localized experts. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 13900–13909.
- Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. 2021. Im2vec: Synthesizing vector graphics without vector supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7342–7351.
- Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. 2020. Sketch-former: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 14153–14162.
- Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10684–10695.
- Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems* 35 (2022), 36479–36494.
- Nikita Srivatsan, Si Wu, Jonathan T Barron, and Taylor Berg-Kirkpatrick. 2021. Scalable font reconstruction with dual latent manifolds. *arXiv preprint arXiv:2109.06627* (2021).
- Rapee Suveeranont and Takeo Igarashi. 2010. Example-based automatic font generation. In *Smart Graphics: 10th International Symposium on Smart Graphics, Banff, Canada, June 24–26, 2010 Proceedings 10*. Springer, 127–138.
- Licheng Tang, Yiyang Cai, Jiaming Liu, Zhibin Hong, Mingming Gong, Minhu Fan, Junyu Han, Jingtuo Liu, Errui Ding, and Jingdong Wang. 2022. Few-Shot Font Generation by Learning Fine-Grained Local Styles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7895–7904.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- Qiang Wang, Haoqiang Deng, Yonggang Qi, Da Li, and Yi-Zhe Song. 2023a. SketchKnitter: Vectorized Sketch Generation with Diffusion Models. In *The Eleventh International Conference on Learning Representations*.
- Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. 2021. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1905–1914.
- Yizhi Wang and Zhouhui Lian. 2021. DeepVecFont: Synthesizing High-quality Vector Fonts via Dual-modality Learning. *ACM Transactions on Graphics* 40, 6 (2021), 15 pages. <https://doi.org/10.1145/3478513.3480488>
- Yuqing Wang, Yizhi Wang, Longhui Yu, Yuesheng Zhu, and Zhouhui Lian. 2023b. DeepVecFont-v2: Exploiting Transformers to Synthesize Vector Fonts with Higher Quality. *arXiv preprint arXiv:2303.14585* (2023).





Fig. 8. Glyph generation results for test cases from our proposed Google font dataset. We compare our method to ChiroDiff [Das et al. 2023] and ablations of our method. Our full method is able to generate glyphs that are much closer to artist-made (“ground-truth”/“GT”) ones compared to alternatives.

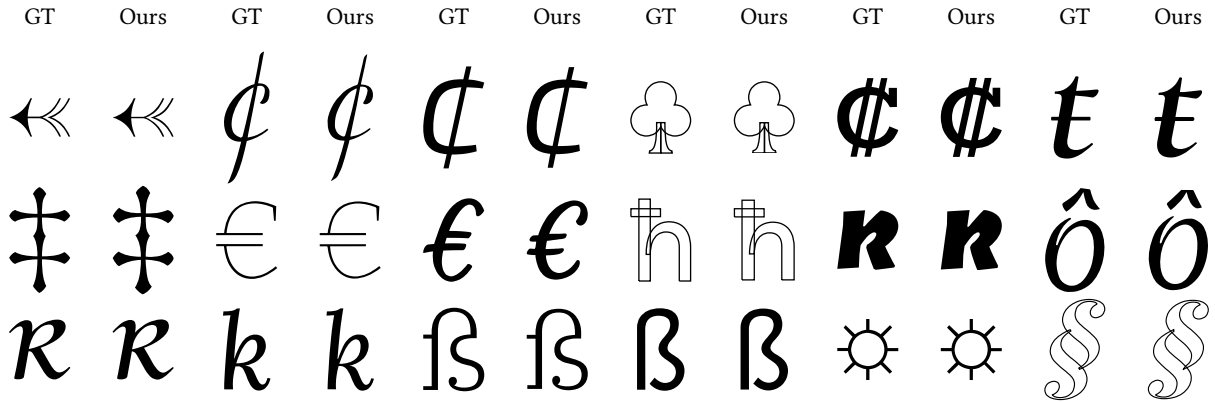


Fig. 9. More glyph generation results and artist-made glyphs (“GT” column) for test cases from the Google font dataset.

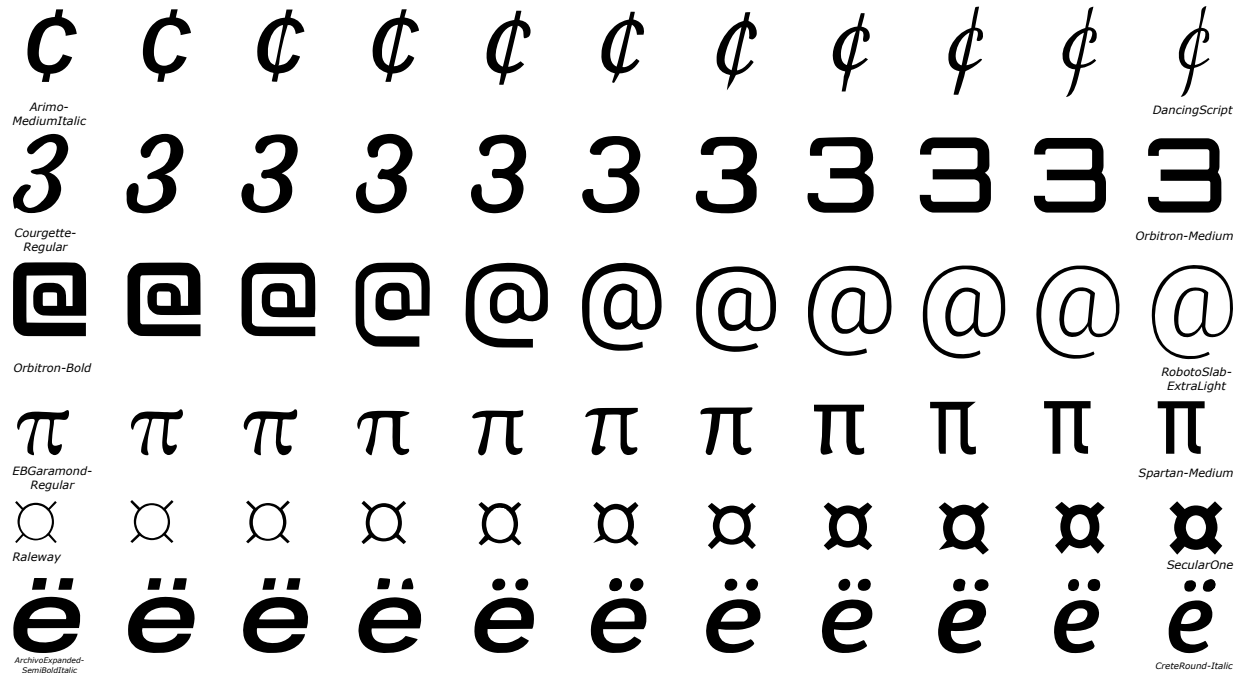


Fig. 10. Font interpolation. We perform linear interpolation in embedding space from source font (left) → target font (right).

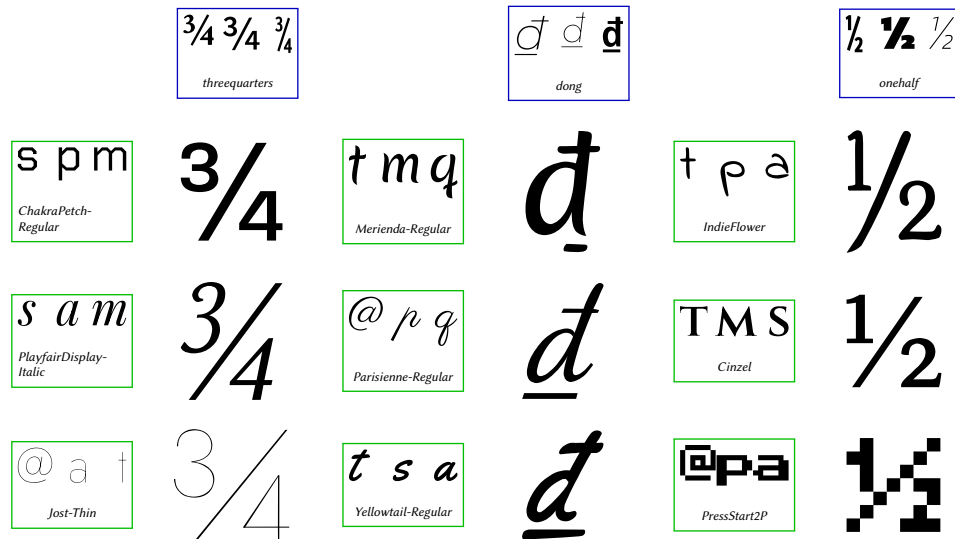


Fig. 11. Additional missing glyphs generation results. Green box: exemplar training glyphs for a target font. Blue box: target character in other font styles. We show generated glyphs for the target character and font.