AttentionPainter: An Efficient and Adaptive Stroke Predictor for Scene Painting

Yizhe Tang*, Yue Wang*, Teng Hu, Ran Yi†, Xin Tan, Lizhuang Ma, Yu-Kun Lai, Paul L. Rosin

Abstract—Stroke-based Rendering (SBR) aims to decompose an input image into a sequence of parameterized strokes, which can be rendered into a painting that resembles the input image. Recently, Neural Painting methods that utilize deep learning and reinforcement learning models to predict the stroke sequences have been developed, but suffer from longer inference time or unstable training. To address these issues, we propose *AttentionPainter*, an efficient and adaptive model for single-step neural painting. First, we propose a novel scalable stroke predictor, which predicts a large number of stroke parameters within a single forward process, instead of the iterative prediction of previous Reinforcement Learning or auto-regressive methods, which makes AttentionPainter faster than previous neural painting methods. To further increase the training efficiency, we propose a Fast Stroke Stacking algorithm, which brings 13 times acceleration for training. Moreover, we propose Stroke-density Loss, which encourages the model to use small strokes for detailed information, to help improve the reconstruction quality. Finally, we design a Stroke Diffusion Model as an application of AttentionPainter, which conducts the denoising process in the stroke parameter space and facilitates stroke-based inpainting and editing applications helpful for human artists' design. Extensive experiments show that AttentionPainter outperforms the state-of-the-art neural painting methods.

Index Terms—Stroke-based Rendering, Neural Painting, Non-Photorealistic Rendering.

1 Introduction

Stroke-based Rendering (SBR) aims to recreate an image by predicting a sequence of parameterized brush strokes, which resembles the content of the input image and imitates the sequential process of human painting, as shown in Fig. 1(a). Recently, researchers have developed *Neural Painting* methods (Deep Learning (DL)-based SBR), which utilize deep learning and reinforcement learning models to predict stroke sequences and achieve Stroke-based Rendering, focusing on generating strokes sequentially and mimicking the painting process.

SBR methods predict sequential parameterized strokes (*e.g.*, Oil, Bézier) that compose a final painting. Key differences from pixel-based painting generation methods (*e.g.*, GANs/Diffusion Models) are twofold: 1) SBR methods model the sequential stroke-by-stroke painting process, while pixel-based generation outputs an entire image, without the modeling of the sequential process. This sequential modeling enables applications like robotic painting, painting education, painting games, *etc.*, which cannot be achieved by pixel-based painting generation. 2) By composing images from parameterized strokes, SBR ensures that local textures adopt the stroke's texture. In contrast, pixel-based generation methods typically lack direct local texture constraints, resulting in inconsistencies between local textures and real painting's detailed texture (Fig. 2).

Y. Tang, Y. Wang, T. Hu, R. Yi, and L. Ma are with the School of Computer Science, Shanghai Jiao Tong University, Shanghai 200240, China.
X. Tan is with the School of Computer Science and Technology, East China Normal University, Shanghai 200062, China.
Y. Lai, and P. Rosin are with the School of Computer Science and Informatics, Cardiff University, CF24 4AG Cardiff, U.K.
Yizhe Tang and Yue Wang contributed equally to this work.
(Corresponding author: Ran Yi.)

The key to stroke-based neural painting is to find the sequence of stroke parameters to reconstruct the image. Previous methods can be divided into two categories, as shown in Figs. 1(b-d): (1) The optimization-based methods [1], [2] directly optimize the randomly initialized stroke parameters to find suitable values for reconstruction. The reconstruction quality of this approach depends on the number of optimization iterations, and high-quality results require a long time of optimization. The inefficiency of optimization-based methods makes it difficult to extend to other applications. (2) Reinforcement Learning (RL)/auto-regressive methods [3]–[6] use an agent network to predict several strokes step-by-step. But these are inefficient in that they can only predict a few strokes (*e.g.*, 5) in a single forward step and require iterative predictions to obtain the final stroke sequence, which results in longer inference time. Also, the training process of the RL methods is unstable [5], and it is difficult to add extra conditions to the models. Therefore, the inefficiency and poor scalability become the development bottlenecks of neural painting.

In this paper, we propose **AttentionPainter**, an efficient and adaptive model for single-step neural painting. Different from RL/auto-regressive methods that only predict a small number (*e.g.*, 5) of strokes based on the current canvas, and require iterating many times for the final results, our AttentionPainter can predict **a large number of strokes within a single forward process**, as shown in Fig. 1(d). To handle a large number of strokes, we design a transformer-based module to extract image features and convert them to a sequence of stroke parameters. With the single-step prediction approach, AttentionPainter has a simpler and more stable training process compared to RL/auto-regressive methods, is faster than all the previous neural painting methods during the inference stage and can

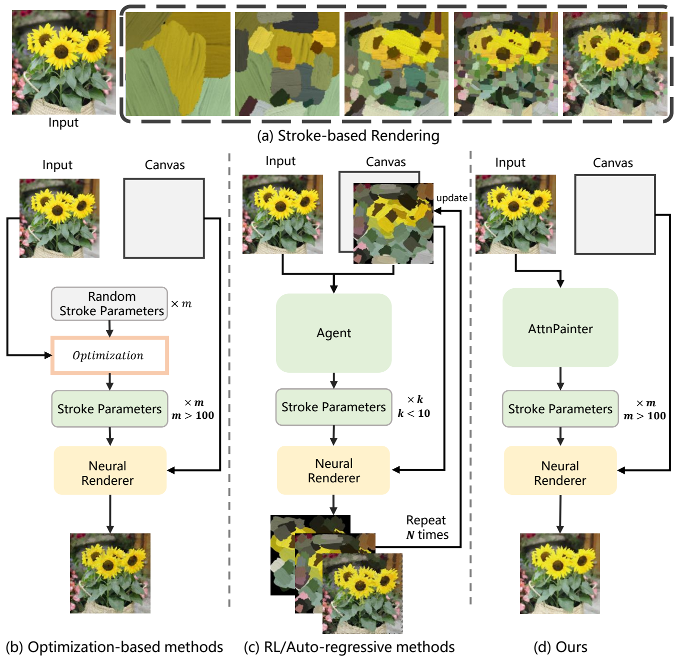


Fig. 1. Stroke-based Rendering (SBR) process and comparison between different methods. (a) SBR aims to recreate an image with a sequence of strokes. (b) Optimization-based methods optimize a sequence of stroke parameters to reconstruct the input image, which requires a separate optimization for each image. (c) Reinforcement Learning (RL)/Auto-regressive methods train an agent to predict a small number (k<10) of strokes at each step and iteratively obtain the final sequence. (d) Our AttentionPainter predicts a large number of strokes (m>100) within a single forward step, and is faster than the other methods during inference.

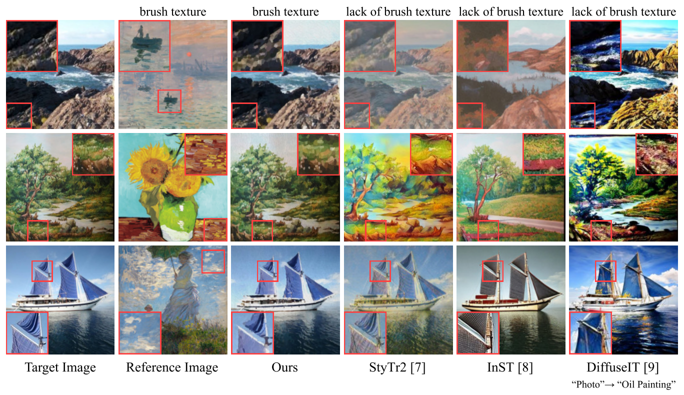


Fig. 2. Comparison with pixel-based oil painting generation methods including StyTr2 [7], InST [8] and DiffuseIT [9]. Pixel-based painting generation methods lack the brush stroke texture and details, while ours have more vivid brush stroke texture and details, similar to that of the real painting in column 2. Please zoom in for more details.

be easily extended to other applications.

Another efficiency bottleneck is the stroke rendering algorithm, where previous methods render the stroke one-by-one and stack each stroke frame iteratively. We propose the **Fast Stroke Stacking** method, which can significantly reduce the iteration number of stroke stacking by selecting the top k strokes for each pixel to stack, enabling simplified stroke rendering process and shortening training time by accelerating the backward process. To further improve the reconstruction quality, we propose **Stroke-density Loss** for detail reconstruction, which guides the model to adaptively use smaller strokes in content-dense areas for detailed information, while larger strokes in content-sparse areas, thus improving the reconstruction quality. Extensive experiments show that with the above design, our AttentionPainter is significantly faster and has better reconstruction results than the state-of-the-art methods.

Moreover, we design a **Stroke Diffusion Model** (SDM), as an application of AttentionPainter, which samples Gaussian noise in the stroke parameter space and denoises it into a meaningful stroke sequence that generates a plausible painting. In SDM, our AttentionPainter plays an important role in accelerating diffusion training and inference. We also demonstrate two applications, stroke-based inpainting and editing, to show how AttentionPainter can help human artists create works.

Our contributions are summarized as follows:

- *AttentionPainter*, a scalable neural painting network, is proposed. It can predict a large number of strokes in a single forward step, instead of iterative prediction used in previous methods, bringing simpler training and faster inference.
- AttentionPainter proposes *Fast Stroke Stacking* (FSS), which significantly reduces the iteration number of stroke stacking by selecting the top k strokes for each pixel to stack, shortening training time by accelerating the backpropagation process.
- AttentionPainter proposes *Stroke-density Loss* to improve detail quality, which guides the model to use smaller strokes in content-dense areas and larger strokes in content-sparse areas.
- *Stroke Diffusion Model* (SDM) is designed as an application of *AttentionPainter*, which conducts the denoising process in the stroke parameter space and enables generating stroke parameter sequences that compose new content.

2 Related Work

2.1 Stroke-Based Rendering

Stroke-Based Rendering (SBR) aims to decompose an input image into a sequence of parameterized strokes, which can be rendered into a painting that resembles the input image. It includes the process of turning the target image into a sequence of stroke parameters and the process of rendering the stroke parameters into a final image.

Different from SBR, there are some pixel-based painting generation methods. They develop deep learning based models to generate artistic images, including Convolutional Neural Networks (CNNs [10]–[12], GANs [13]–[15], Variational Autoencoders (VAEs) [16], Normalizing Flow models [17], Diffusion models [18], [19], [8], [9]. However, these models usually directly operate in the pixel space and lack the modeling of the painting process similar to that of human artists, who typically draw a sequence of shapes and strokes to create a painting [20]. In other words, these pixel-based painting generation methods directly generate an image instead of predicting a sequence of stroke parameters.

Their results often have an unrealistic stroke sense, as seen in Fig. 2 colums 4-6, lacking oil stroke texture.

Different from these pixel-based painting generation methods, instead of embedding an image into a natural distribution (*e.g.* Gaussian distribution), SBR methods convert an image into a sequence of parameterized strokes [21]. After the stroke parameters are generated, a rendering algorithm or neural renderer utilizes those parameters to recreate a new image. As shown in Fig. 2, SBR method's results have more vivid brush stroke texture and details, similar to that of the real painting; in contrast, the results of pixel-based painting generation methods often lack the brush stroke texture.

Stroke-based rendering methods can be divided into painterly rendering [22], [23], stippling [24], [25], sketching [26], [27], [28], [29], Scalable Vector Graphics [30]–[33], etc. according to different rendering targets. Early methods [22], [23], [34]–[36] mainly depend on greedy search algorithms to locate the position based on regions or edges, and decide other characteristics of strokes. On the other hand, the optimization based methods [37]–[40] iteratively search the stroke parameters to achieve a lower energy function [21]. However, most of these methods are very time-consuming and often require manual adjustment of hyperparameters to achieve better rendering results.

2.2 Neural Renderer

In order to visualize the stroke parameters obtained from the given model, a Renderer is necessary to convert the parameterized representation into pixel-based images. Traditional Renderer typically involves the operation of rasterization, which makes the rendering process non-differentiable. The emergence of differentiable Neural Renderers [2], [3], [41]–[43] breaks this limitation and allows for the calculation of gradients from the output images to the input parameters. Neural Renderers are neural networks trained to simulate the traditional non-differentiable renderers, where the training dataset of parameters and ground truth corresponding pixel-level images can be easily created by the non-differentiable renderer [20]. Due to the use of neural networks, the calculation on Neural Renderer is more efficient and enables end-to-end training. SoftRas [43] proposed a neural renderer for 3D mesh rendering, which supports rendering colorized meshes with differentiable functions and back-propagating supervision signals to mesh attributes. For stroke-based rendering, Learning to Paint [3] first designed a stroke representation of a quadratic Bezier curve with thickness and transparency and a neural renderer based on convolutional neural networks to simulate the effects of brushes. To solve the coupling of shape and color representations, Stylized Neural Painting [2] designed a dual-pathway neural renderer that disentangles color and shape through the rendering pipeline.

2.3 Neural Painting

In SBR, traditional methods usually adopt greedy search or require user inputs to obtain the stroke position and other parameters. With the development of deep learning (DL), many stroke-based rendering methods based on neural networks have been proposed, which utilize deep learning and reinforcement learning methods to predict stroke sequence and imitate the process of human painting. Such SBR methods based on neural networks (DL-based SBR) are usually called **Neural Painting** to distinguish from those traditional methods, which are like a subfield of SBR. For neural painting in specific domains, some previous methods, *e.g.*, StrokeNet [42] and Sketch-RNN [44], use recurrent neural networks (RNNs) [45] to generate sketch or stroke sequences for simple characters and hand drawings. Recent neural painting methods can be categorized into three types:

1) Reinforcement Learning (RL) based methods: Some methods [3], [46], [47] propose to use reinforcement learning in neural painting, using an agent network to predict several strokes step-by-step, learning the structure of images and reconstructing them. Recently, based on the Deep Reinforcement Learning (DRL) strategy, some researchers [4], [48] propose that dividing the input image into foreground and background [49] helps improve the drawing quality, while some researchers [50] propose Content Masked Loss to assign higher weights to the recognizable areas. However, the training process of Reinforcement Learning methods is often unstable [5], which limits their further applications.

2) Auto-regressive methods: All strokes of a single image form a vector sequence, and sequence data is the ideal data format for Transformers [51], [52]. Paint Transformer [5] is an auto-regressive Transformer-based neural painting method. However, it can only predict a few strokes (*e.g.*, 5) in one step, and requires iterative predictions to generate the final stroke sequence. Different from the Paint Transformer, in this paper, we propose a more efficient way to utilize an attention-based model to generate all strokes during a single forward process.

3) Optimization-based or search-based methods: Li et al. [53] propose a simple painterly rendering algorithm using a differentiable rasterizer to facilitate backpropagation between two domains. With the support of deep neural networks, optimization-based or search-based neural painting methods [1], [2], [54] also have significant improvement and can be naturally combined with style loss [10] to realize Stroke-based style transfer. Curved-SBR [55] uses thin plate spline interpolation [56] to achieve a more diverse brush-stroke effect. However, these methods all suffer from the problem of a long optimization time, which hinders their wide applications.

Recently, VectorPainter [57] and ProcessPainter [58] explore multi-modal stroke-based painting generation, taking a text prompt as input, with the former focusing on synthesizing stylized vector graphics by rearranging vectorized strokes, and the latter generating painting process videos that mimic human artists' techniques. Our task is different from theirs, mainly predicting a sequence of stroke parameters to recreate a given input image, without the need for textual guidance.

In contrast to previous neural painting methods, our method can generate many strokes in a single forward process without RL or an auto-regressive structure, and has a more stable training process than previous methods. Since our method is model-based and needs only a single forward process, it is faster than those methods based on optimization or search.

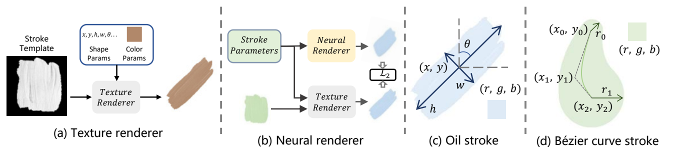


Fig. 3. Stroke renderer and stroke design. Texture Renderer performs geometric transformation based on a real stroke template, but it is not differentiable. Neural Stroke Renderer is used to simulate the Texture Renderer, but makes the process differentiable. For the stroke design, we use Oil strokes and Bézier curve strokes in this paper.

3 Problem Formulation and Preliminaries

Given an input image $\mathbf{I}$, stroke-based rendering methods predict a sequence of stroke parameters $\mathbf{S} = \{\mathbf{s}^1, \mathbf{s}^2, ..., \mathbf{s}^N\}$ (*e.g.*, shape, color, position), and use stroke renderer $\mathcal{R}$ to paint strokes onto the canvas sequentially. The rendered image $\hat{\mathbf{I}}$ is expected to be similar to the input image $\mathbf{I}$.

3.1 Stroke Renderer

Texture Renderer. *Texture Renderer* is the stroke renderer that performs geometric transformations directly on the stroke template. The Texture Renderer takes the stroke template and geometric parameters as input, such as the stroke position (x, y), stroke height and width h, w, and rotation angle θ. Then the texture renderer performs geometric transformations such as translation, scaling, and rotation, as shown in Fig. 3(a). The Texture Renderer can preserve the texture of the original stroke template well. However, the texture renderer performs geometric transformation based on discrete computer graphics operations, which makes the rendering process non-differentiable, and prevents backpropagation during end-to-end training.

Neural Renderer. To enable the backpropagation in the training process, Learning to Paint [3] proposes to train a neural network to simulate the rendering process of the Texture Renderer, which is called the *Neural Renderer*. Subsequently, various Neural Renderers have been developed, featuring different model architectures and stroke designs [2], [5], [6], [55]. As illustrated in Fig. 3(b), the Neural Renderer takes stroke parameters as input and produces a rendered stroke image, while the Texture Renderer is employed to generate diverse types of ground truth for training the Neural Renderer.

3.2 Stroke Design

In SBR, strokes are represented by stroke parameters. There are several types of strokes, *e.g.*, Bézier, Oil painting, Tape art. Among them, Oil stroke and Bézier curve stroke are two commonly adopted stroke types in stroke-based neural painting methods, *e.g.*, Learning To Paint [3] (both), CNP [6] (oil), Parameterized Brushstrokes [1] (Bézier), so in this paper, we experiment with these two kinds of strokes. (1) Oil stroke, which uses 8 parameters to represent a stroke, *i.e.*, 1 center point for position (x, y), the stroke height and width (h, w), the rotation angle θ, and (R, G, B). (2) Bézier curve stroke, which uses the coordinates of 3 control points $(x_0, y_0, x_1, y_1, x_2, y_2)$ to control the shape of the Bézier curve, 4 parameters (r_0, t_0, r_1, t_1) to control the thickness and transparency of the two endpoints of the curve, and (R, G, B) to control the color. The above two types of strokes are illustrated in Figs. 3(c)(d). The stroke renderer then takes the stroke parameters as input and outputs the stroke map $\mathcal{C}$ and alpha map $\mathcal{A}$. The stroke map $\mathcal{C}$ values are stroke color for each pixel inside the stroke area. The alpha map $\mathcal{A}$ refers to the stroke mask for Oil strokes, while for Bézier curve strokes, it refers to the transparency of the stroke.

TABLE 1
Setting comparison between different methods.

Method	Type	Single-Step Output Stroke Num
Learning to Paint [3]	RL	$\leq$10
CNP [6]	RL	$\leq$10
Paint Transformer [5]	Auto-regressive	$\leq$10
Stylized Neural Painting [2]	Optim-based	—
Ours	End-to-End	$\geq$200

4 AttentionPainter

4.1 Overview: Single-Step Neural Painting

Our scalable stroke predictor, AttentionPainter, is a single-step neural painting model, which can predict **a large number of strokes in a single forward process**. Different from the RL/auto-regressive neural painting methods (*e.g.*, Paint Transformer [5], Learning to Paint [3]) that predict a small number of strokes at each step and require iterative prediction to get the final stroke sequence, our AttentionPainter predicts all the strokes within a single forward process. Without the iterative predictions, AttentionPainter is faster than the previous methods during inference. We compare our setting with previous methods in Tab. 1.

AttentionPainter contains three important parts to achieve high-quality and fast neural painting. (1) Large-Number Stroke Prediction in a Single Step. We design an attention-based stroke predictor, which generates a large number of strokes in a single step and avoids iterative prediction as previous methods (in Sec. 4.2). (2) Fast Stroke Stacking (FSS) for fast stroke rendering (in Sec. 4.3). The previous neural rendering methods stack all the stroke frames one by one, which costs a lot of time when the number of strokes is large. FSS is significantly faster than the previous stacking method and helps accelerate the training process of both AttentionPainter and StrokeDiffusion (in Sec. 5). (3) Stroke-density loss for detail reconstruction, which encourages the model to put smaller strokes in high-density regions and improve reconstruction quality (in Sec. 4.4).

The single-step neural painting pipeline is shown in Fig. 4. The stroke predictor extracts features from the input image $\mathbf{I}$ and predicts a sequence of stroke parameters $\mathbf{S}$ using a Transformer-based structure. The stroke renderer $\mathcal{R}$ then renders the stroke parameters into stroke frames for each stroke. Finally, the fast stroke stacking module stacks the stroke frames into the final rendered image $\hat{\mathbf{I}}$.

4.2 Large-Number Stroke Prediction in a Single Step

We design a Single-Step Neural Painting pipeline. Different from the RL/auto-regressive methods that can only generate a few strokes in a single step, and repeat the forward process multiple times to get the final results, our method predicts

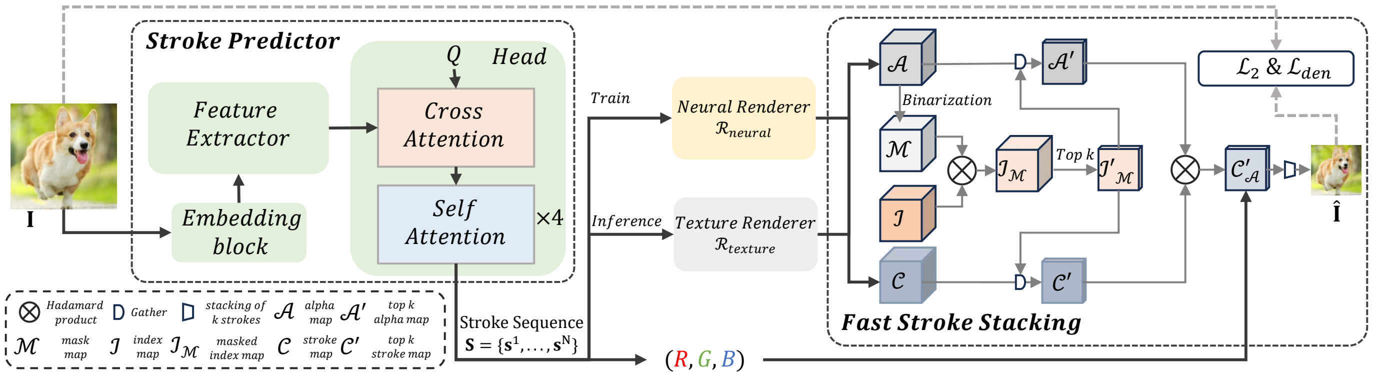


Fig. 4. AttentionPainter Architecture. Given an image $\mathbf{I}$, 1) the Stroke Predictor predicts a large number of strokes in a single forward, which first extracts features, and then predicts the stroke parameter sequence based on cross-attention and self-attention blocks. 2) With the predicted stroke parameters, the Stroke Renderer renders the stroke frame for each stroke. 3) Finally, the Fast Stroke Stacking (**FSS**) module simplifies the stroke stacking process by selecting the top k strokes for each pixel to stack, and creates the final rendering. AttentionPainter is trained with pixel-wise loss and a newly proposed stroke-density loss.

all the strokes in a single forward step. We use an attention-based network to solve this problem. Unlike Paint Transformer [5], our model eliminates the cumbersome iterative prediction process.

4.2.1 Stroke Predictor

Given an input image $\mathbf{I}$, we first use an embedding block to convert the input image to a token sequence, as most vision Transformers [59] do. The embedding block contains a convolution layer and a normalization layer[1]. After embedding, we use ViT-small as the feature extractor to extract the patch features of the input image. The stroke prediction head adopts a Transformer structure with 1 cross-attention block and 4 self-attention blocks. In the cross-attention block, we calculate the correlation between the patch features f and stroke queries Q. Since the attention block is a global operation, it helps to find the best strokes for the whole image. After the cross-attention block, we use 4 vanilla self-attention blocks to process the features, and the output of the last self-attention block is set as the stroke parameter sequence $\mathbf{S}$, which is a single-step result and contains all the strokes for rendering.

4.2.2 Neural Painting with Stroke Parameters

For each stroke $\mathbf{s}^i$ in the output stroke parameter sequence $\mathbf{S} = \{\mathbf{s}^1, \mathbf{s}^2, ..., \mathbf{s}^N\}$, the parameters can be divided into two parts: geometry parameters $\mathbf{s}_g$ and color parameters $\mathbf{s}_c$, *i.e.*, $\mathbf{s} = \{\mathbf{s}_g, \mathbf{s}_c\}$. The stroke sequence $\mathbf{S}$ can also be divided into geometry part $\mathbf{S}_g$ and color part $\mathbf{S}_c$. During the training stage, we use the Neural Renderer $\mathcal{R}_{neural}$ (Sec. 3), which is differentiable and enables back-propagation. During the inference stage, we use the Texture Renderer $\mathcal{R}_{texture}$, which performs geometric transformations directly on the stroke template, for better rendering results. Since most texture renderings are not differentiable, the Texture Renderer cannot perform backpropagation during training and can only be used for inference.

Each stroke with the stroke parameters $\mathbf{s}^i$ is rendered into a stroke frame, which contains an alpha map $\mathcal{A}^i$ and a colored stroke map $\mathcal{C}^i$. The rendered stroke frames are then stacked together into the final image $\hat{\mathbf{I}}$ in an iterative rendering process. Finally we can measure the difference between the input image $\mathbf{I}$ and the final image $\hat{\mathbf{I}}$, where we use the weighted combination of $\mathcal{L}_{MSE}$ loss and stroke-density loss $\mathcal{L}_{den}$ which will be introduced in Sec. 4.4. With the differentiability of the Neural Renderer, it is feasible to calculate the gradient of parameters in the Stroke Predictor, and optimize the model for a better stroke parameter sequence. However, when stacking a large number of strokes, the forward propagation will become complex, and the backpropagation process will take too long and be too difficult. To solve this problem, we propose Fast Stroke Stacking (in Sec. 4.3) to simplify stacking of the stroke frames to render the image $\hat{\mathbf{I}}$.

It should be noted that while our method outputs a temporally ordered stroke sequence, its generation order is not explicitly optimized to replicate the delicate workflow of a human artist, primarily due to the lack of sequential human painting process data as supervision. Instead, the key supervision signal in our training is applied to the final rendered output image, not to the intermediate stroke sequence or the specific order in which strokes are generated.

4.3 Fast Stroke Stacking

During the rendering stage, previous neural painting methods [3], [5], [6] use a **stroke-by-stroke process to stack** all the rendered stroke frames. This stacking strategy makes the backpropagation difficult and complex during training (see analysis below). To solve this problem, we propose the **Fast Stroke Stacking** (FSS) algorithm, which simplifies the rendering process by selecting the top k strokes of each pixel to stack, instead of stacking all strokes iteratively. Our FSS algorithm greatly shortens the training time by accelerating the backward process, and facilitates the convergence during training[2].

1. The conv layer's input channel is typically 3, but can be extended to 4 to include a 1-channel density map when calculating density loss.

2. Without FSS, our network would require more time for backpropagation, increasing training costs and making convergence difficult.

Analysis of Traditional Stroke Stacking and Backpropagation. The traditional stroke rendering process can be formulated as the following iterative form:

$$\hat{\mathbf{I}}^i = \hat{\mathbf{I}}^{i-1} \cdot (1 - \mathcal{A}^i) + \mathcal{A}^i \cdot \mathcal{C}^i, \tag{1}$$

where i is the index of a stroke ($i = 1, \cdots, N$), N is the total number of strokes, $\mathcal{A}^i$ is the alpha map of the i-th stroke, and $\mathcal{C}^i$ is the colored stroke frame of the i-th stroke. This formula describes the process of drawing a new stroke (the i-th stroke) onto the current canvas ($\hat{\mathbf{I}}^{i-1}$) to get the next canvas ($\hat{\mathbf{I}}^i$). This process is repeated N times until the final painting $\hat{\mathbf{I}}^N$ is obtained. We expand Eq. (1) to derive the final painting $\hat{\mathbf{I}}^N$ as follows:

$$\hat{\mathbf{I}}^N = \sum_{k=1}^{N-1} \left(\mathcal{A}^k \cdot \mathcal{C}^k \cdot \prod_{j=k+1}^{N} (1 - \mathcal{A}^j) \right) + \mathcal{A}^N \cdot \mathcal{C}^N, \tag{2}$$

where the final painting $\hat{\mathbf{I}}^N$ can be written as the linear combination of N colored stroke frames $\{\mathcal{C}^1, ..., \mathcal{C}^N\}$, and the weights are calculated from their Alpha maps $\{\mathcal{A}^1, ..., \mathcal{A}^N\}$.

Our training target is to train the Stroke Predictor, so the model gradients are related to the partial derivatives $\frac{\partial \hat{\mathbf{I}}^N}{\partial \mathcal{A}^m}$ and $\frac{\partial \hat{\mathbf{I}}^N}{\partial \mathcal{C}^m}$ ($1 \leq m \leq N$), which can be calculated as:

$$\frac{\partial \hat{\mathbf{I}}^N}{\partial \mathcal{A}^m} = (\mathcal{C}^m - \hat{\mathbf{I}}^{m-1}) \cdot \prod_{j=m+1}^{N} (1 - \mathcal{A}^j), \tag{3}$$

$$\frac{\partial \hat{\mathbf{I}}^N}{\partial \mathcal{C}^m} = \mathcal{A}^m \cdot \prod_{j=m+1}^{N} (1 - \mathcal{A}^j). \tag{4}$$

Therefore, when the stroke number N gets larger, the traditional stacking strategy suffers from a more complex backpropagation process ($O(N^2)$ complexity, detailed derivation in the supplementary material), leading to an unacceptable long training time. Thankfully, we observe that most computation is unnecessary in Eq. (1), and propose FSS to significantly reduce the iteration number during backpropagation.

Observation of Unnecessary Computation. As shown in Eq. (1) and Eq. (2), each canvas $\hat{\mathbf{I}}^{i-1}$ is multiplied with the alpha map $(1 - \mathcal{A}^i)$ ($\forall \alpha \in 1 - \mathcal{A}^i, 0 \leq \alpha \leq 1$). Since the iterative step (Eq. (1)) is repeated for multiple times, $\prod_{j=i}^{N}(1 - \mathcal{A}^j)$ will quickly approach zero. This means that most strokes have little influence on the final result after several overlapping operations. Moreover, from Eq. (3) and Eq. (4), we can see that both partial derivatives $\frac{\partial \hat{\mathbf{I}}^i}{\partial \mathcal{A}^m}$ and $\frac{\partial \hat{\mathbf{I}}^i}{\partial \mathcal{C}^m}$ contain the factor $\prod_{j=m+1}^{N}(1 - \mathcal{A}^j)$. This indicates that for the m-th stroke, the pixels drawn by the subsequent strokes hardly contribute to the backpropagation gradients, because for these pixels (covered by some of the subsequent strokes), the factor becomes zero, resulting in zero gradient contribution for the partial derivatives. The smaller the index m, the smaller the corresponding factor becomes, leading to a reduced impact of that stroke on the backpropagation gradients.

Fast Stroke Stacking. Therefore, for each pixel, we choose the alpha value and stroke color value of the **top** k **strokes that cover the pixel**, *i.e.*, the strokes with the k largest indices among all the strokes covering the pixel[3]. For a certain pixel, the neglected strokes are strokes with small indices and less impact on the final painting. Not stacking these strokes has a negligible influence on performance, but helps to reduce unnecessary computation and simplify the stacking process.

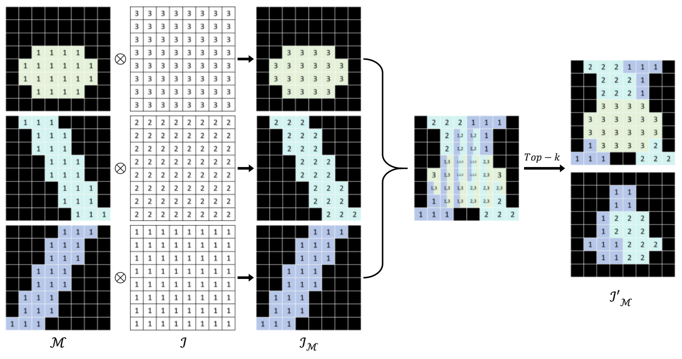


Fig. 5. An example of FSS calculation process. For better illustration, here we set the stroke number N as 3 (typically it is much larger, *e.g.*, 256), and the top-k as top-2, and we mark the strokes with different colors to distinguish between each other.

Specifically, we first initialize an Index Tensor $\mathcal{I}$ ($B \times N \times H \times W$), where the i-th channel of $\mathcal{I}$ is assigned value i, and B, N, H, W denote the batch size, the channel number, the height and width of the image, respectively. Then we binarize the stroke alpha map $\mathcal{A}$ to 0 or 1 with a threshold[4] to obtain a binary stroke mask $\mathcal{M}$ ($B \times N \times H \times W$), which is then combined with the Index Tensor to obtain the indices of strokes that cover each pixel. We calculate the Masked Index Tensor $\mathcal{I}_{\mathcal{M}}$ as the Hadamard product of $\mathcal{M}$ and $\mathcal{I}$, where the in-stroke region in $\mathcal{I}^i_{\mathcal{M}}$ has value i, and the out-of-stroke region has value 0. Then **the top** k **strokes** of a **pixel** (x, y) are the strokes with the largest k indices in $\mathcal{I}^{(x,y)}_{\mathcal{M}}$. We select the top k indices for each pixel, to construct the Top-k Index Tensor $\mathcal{I}'_{\mathcal{M}}$ ($B \times k \times H \times W$). An example of this calculation process is shown in Fig. 5.

Then, given the original alpha map $\mathcal{A}$ with dimensions ($B \times N \times H \times W \times 1$) and stroke map with dimensions $\mathcal{C}$ ($B \times N \times H \times W \times 3$), along with the Top-$k$ Index $\mathcal{I}'_{\mathcal{M}}$, we obtain the alpha values and colors of the top k strokes at each pixel by extracting elements from $\mathcal{A}$ and $\mathcal{C}$ based on the given indices $\mathcal{I}'_{\mathcal{M}}$:

$$\mathcal{A}' = Gather(\mathcal{A}, indices = \mathcal{I}'_{\mathcal{M}}), \tag{5}$$

$$\mathcal{C}' = Gather(\mathcal{C}, indices = \mathcal{I}'_{\mathcal{M}}), \tag{6}$$

where the $Gather(\mathcal{A}/\mathcal{C}, indices = \mathcal{I}'_{\mathcal{M}})$ operation follows the implementation in PyTorch, which extracts k values out of a total of N elements from the given tensor $\mathcal{A}$ or $\mathcal{C}$ along the second dimension based on the specified indices $\mathcal{I}'_{\mathcal{M}}$.

Finally, we combine the top k stroke maps $\mathcal{C}'$ ($B \times k \times H \times W \times 3$) and top k alpha maps $\mathcal{A}'$ ($B \times k \times H \times W \times 1$)

3. Each pixel has its own top k strokes. Despite selecting a subset per pixel, experiments show that overall all strokes contribute to the stacking.

4. During training, the neural renderer creates an alpha map with values near 0 or 1. We apply binarization using a 0.5 threshold to make it binary. Due to the bimodal distribution, a threshold of 0.5 is sufficient.

with Hadamard product, and conduct k ($k \leq 10$) times of traditional stroke rendering step similar to Eq. (1), to obtain the final image $\hat{\mathbf{I}}$:

$$\hat{\mathbf{I}}^i = \hat{\mathbf{I}}^{i-1} \cdot (1 - \mathcal{A'}^i) + \mathcal{A'}^i \cdot \mathcal{C'}^i, \tag{7}$$

where $\cdot$ represents the element-wise (Hadamard) product. Due to the much fewer iterations ($k \leq 10$ compared to $N = 256$), our FSS algorithm can avoid a large amount of unnecessary backpropagation computation and greatly shorten the training process, improving the training efficiency by tens of times compared to the original stacking algorithm, with negligible degradation in quality.

4.4 Stroke-density Loss

In terms of visual perception, humans pay more attention to complex areas with high density, where more semantic information is stored. When painting on the canvas, human painters draw small and dense strokes in those complex areas, while big and sparse strokes are drawn in those content-sparse areas. Therefore, to draw like a human painter, we propose the Stroke-density Loss, which is computed by density map and stroke area map. Different from Im2Oil [51] which uses density information as a probability map to sample strokes, we optimize our AttentionPainter using our newly proposed Stroke-density Loss, in order to draw more and smaller strokes in content-dense areas, thereby achieving better reconstruction of details.

First, we calculate the stroke area of each stroke, to enable control of stroke sizes under the guidance of the density map. For the two types of strokes used in our paper, *i.e.*, Oil stroke and Bézier curve stroke, considering the complexity of computing area for Bézier curves, we only apply the stroke-density loss for Oil stroke. Specifically, we calculate the stroke area by the width and height parameters (defined in Oil stroke parameters). We define a stroke area map $\mathcal{M}_{area}$, where for each visible pixel in a stroke's region, it is assigned the area of that stroke:

$$\mathcal{M}_{area} = \mathcal{M} \cdot (S_h \cdot S_w), \tag{8}$$

where S_h, S_w ($B \times N \times 1$) are the tensors with the height and width values of all strokes from the stroke parameters sequence $\mathbf{S}$, and we consider the result of $S_h \cdot S_w$ as the area of each stroke. $\mathcal{M}$ is the binary stroke mask obtained in FSS process.

Then we gather the top k stroke area values in a manner similar to that described in Eq. (5) and Eq. (6):

$$\mathcal{M}'_{area} = Gather(\mathcal{M}_{area}, indices = \mathcal{I}'_{\mathcal{M}}), \tag{9}$$

in which way we obtain the top k stroke area maps $\mathcal{M}'_{area}$ ($B \times k \times H \times W \times 1$). Then, following the rendering formula in Eq. (7), we replace $\mathcal{C}'$ with $\mathcal{M}'_{area}$ to obtain the stroke area image $\hat{\mathbf{I}}_{area}$ by Eq. (10), where each pixel value in $\hat{\mathbf{I}}_{area}$ represents the area of the stroke at that pixel:

$$\hat{\mathbf{I}}^i_{area} = \hat{\mathbf{I}}^{i-1}_{area} \cdot (1 - \mathcal{A'}^i) + \mathcal{A'}^i \cdot \mathcal{M'}^i_{area}. \tag{10}$$

To compute the density map, we employ the Sobel operator and average pooling to estimate the density detail in the input image. Sobel operator and average pooling are commonly used methods to measure the density of information in an image and are also used in Im2Oil [60]. The Sobel operator can reflect the boundaries or content-dense regions of an image by high gradients in the image; while the average pooling operation smoothes out these responses to provide more stable density information. We denote the estimated density map as $\mathbf{I}_d$, which is calculated as:

$$G_x = \mathbf{I} * \mathbf{S}_x, G_y = \mathbf{I} * \mathbf{S}_y, \mathbf{I}_d = F_{mean}(\sqrt{G_x^2 + G_y^2}), \tag{11}$$

where $\mathbf{S}_x$ and $\mathbf{S}_y$ denote the Sobel Operator on Horizontal and Vertical directions, and F_{mean} denotes the Mean Filter used in average pooling. Subsequently, the stroke-density loss is formulated as:

$$\mathcal{L}_{den} = \bar{\mathbf{X}}, \quad \mathbf{X} = \hat{\mathbf{I}}_{area} \cdot \mathbf{I}_d, \tag{12}$$

where $\bar{\mathbf{X}}$ denotes the mean value of the elements of matrix $\mathbf{X}$. By minimizing $\mathcal{L}_{den}$, the sizes of strokes in high-density regions tend to decrease to lower values, resulting in smaller strokes in these regions to better capture detail. We find that our stroke-density loss can also be applied to previous neural painting methods and improve their performance (*results presented in Sec. 6.5*).

For the AttentionPainter training, we use two loss terms: pixel-wise loss and stroke-density loss. The pixel-wise loss achieves global reconstruction optimization, but some details of the image are still missing. The stroke-density loss can effectively optimize the spatial distribution of strokes to achieve better reconstruction results. For pixel-wise loss, we directly use MSE loss $\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{I}_i - \hat{\mathbf{I}}_i)^2$, and the total loss function is:

$$\mathcal{L}_{AP} = \mathcal{L}_{MSE} + \lambda \mathcal{L}_{den}, \tag{13}$$

where λ is a hyper-parameter to balance the loss terms.

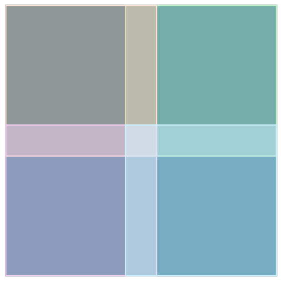


Fig. 6. Illustration of improved block dividing strategy: for the original divided blocks, we make their predicted regions extend a certain number of pixels (*e.g.*, 10 pixels) towards the neighboring blocks.

4.5 Inference Process

Our AttentionPainter can predict 256 strokes in one feed-forward pass. For high resolution results with more strokes, previous methods [2], [3], [5] usually apply divide-predict-combine operation where the target image is first divided into several non-overlapping blocks and then strokes for each block are predicted. This results in an unnatural discontinuity at the boundaries of adjacent image blocks. To overcome this problem, we make several improvements in the inference phase. First, we adjust the block dividing strategy

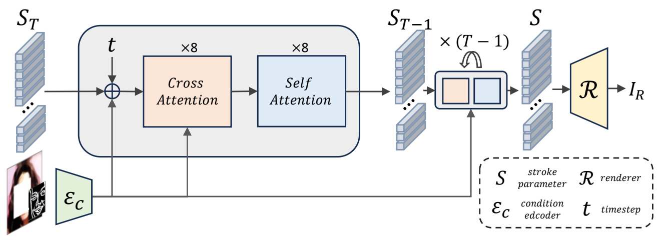


Fig. 7. Stroke Diffusion Model (SDM) conducts diffusion and the denoising process in the **stroke parameter space**, where the denoising stage uses an attention-based network (8 cross-attention blocks and 8 self-attention blocks). The proposed stroke predictor in AttentionPainter is used to obtain the stroke parameters from images, and the denoised stroke parameters are decoded to the output image by the Neural Renderer and our FSS module.

so that there are overlapping regions between neighboring target blocks. Specifically, for the original divided blocks, we make their predicted regions extend a certain number of pixels (*e.g.*, 10 pixels) towards the neighboring blocks (as shown in Fig. 6). To allow for more diversity, we randomly rotate (in a multiple of $90°$) and flip these input blocks, in which way there will be 8 types of input patterns (4 unique rotations, with or without flipping). Then, we still predict the stroke parameters for each target block image separately, and render the strokes for each block. Afterwards, we rotate and flip back these strokes in the reverse operation. Finally, when re-combining these blocks, we make the strokes of neighboring blocks overlap with each other in the overlapping regions. With such a pipeline, the final generated image has more natural transitions at the boundaries of the blocks, reducing artifacts at the boundaries.

5 Stroke Generation with Diffusion Model

With AttentionPainter, we can quickly predict the stroke parameters and render the final image, which further helps extend stroke-based rendering to other applications. Previous neural painting methods focus on reconstructing the input image with strokes, but they cannot generate a stroke-based unseen painting. In this section, we design a **Stroke Diffusion Model** (SDM), as an application of AttentionPainter, which directly conducts the diffusion process and denoising diffusion process in the **stroke parameter space**. With SDM, we can sample Gaussian noise in the stroke parameter space and denoise it into a meaningful stroke sequence, where the strokes constitute a plausible painting.

5.1 Stroke Diffusion Model Design

Inspired by LDM [61], we first embed the image into the stroke parameter space with the stroke predictor, and decode the stroke parameters to the output image with the neural stroke renderer and FSS module. Different from previous diffusion models that add noise and denoise in the image space or latent space, our Stroke Diffusion Model is designed to generate stroke parameters for a conditional image and denoise in the **stroke parameter space**. Since the stroke parameters are a sequence of vectors, instead of a 2D feature map, the UNet-based diffusion module is less suitable here. To effectively process the stroke parameters, we design a ViT-based diffusion module to deal with

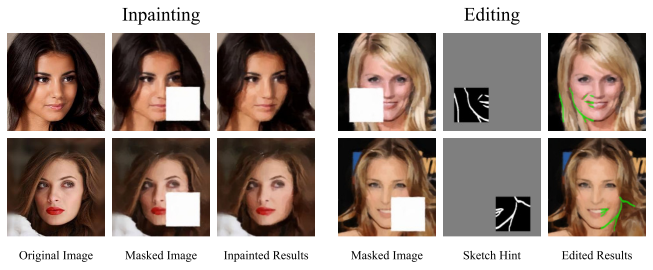


Fig. 8. Applications of Stroke Diffusion Model (SDM). The left are Stroke-based Inpainting results, and the right are Stroke-based Image Editing results (the sketch hints are overlaid onto the results for better visualization). Stroke Diffusion Model generates stroke parameters for a conditional input and denoises in the stroke parameter space. The conditional input is a masked image (left), or a masked image and a sketch hint (right), and the output of Stroke Diffusion Model is a sequence of stroke parameters, which are then rendered into a complete painting.

the stroke sequence. As illustrated in Fig. 7, the diffusion module contains 8 cross-attention blocks and another 8 self-attention blocks. The cross-attention blocks are used to allow for conditional input (a condition encoder encodes the conditional input and the processed features then participate in the cross-attention calculation). The loss function of our SDM is:

$$\mathcal{L}_{SDM} = \mathbb{E}_{\mathcal{E}(x),y,\epsilon\sim\mathcal{N}(0,1),t}[||\epsilon - \epsilon_\theta(z_t, t, \mathcal{E}_c(y))||_2^2], \quad (14)$$

where $\mathcal{E}(x)$ is the stroke predictor, y is the conditional input and $\mathcal{E}_c(y)$ is the condition encoder, t is the timestep, and z_t is the noised sample (stroke parameters) at timestep t.

5.2 Stroke-based Inpainting

Given a partially painted image, our SDM generates a sequence of stroke parameters, which are then rendered into a complete painting. We refer to this process as Stroke-based Inpainting (Fig. 8(a)). The conditional input for our inpainting model is a masked image. Following the denoising diffusion process in the stroke parameter space, the model outputs the entire sequence of stroke parameters, which can be rendered into the full stroke-based image.

5.3 Stroke-based Image Editing

Apart from inpainting from a masked image, we can further add more conditions to the inpainting model to generate more controllable strokes, which we refer to as Stroke-based Image Editing. In the editing network, we add a sketch hint of the masked region as one of the conditional inputs (using a condition encoder trained on the sketch domain to encode this sketch hint), and the users can edit the mask region with the freehand sketches (Fig. 8(b)). Our SDM can edit the painting according to the hint in the conditional sketch by predicting a sequence of stroke parameters that represent the editing strokes.

6 Experiments

6.1 Implementation Details

Model Details: 1) For AttentionPainter, the embedding layer is a convolution block that converts the image to

TABLE 2
Quantitative comparison with state-of-the-art methods under different stroke numbers and two stroke types. Our AttentionPainter outperforms the state-of-the-arts in almost all settings.

Stroke Type	Stroke number	Method	ImageNet			CelebAMask-HQ		
			L2 ↓	SSIM ↑	LPIPS ↓	L2 ↓	SSIM ↑	LPIPS ↓
Oil Stroke	250	Paint Transformer [5]	0.0247	0.4328	0.1760	0.0176	0.5095	0.1692
		Learning to Paint [3]	0.0124	0.4972	0.1700	0.0081	0.5922	0.1506
		Semantic Guidance+RL [4]	0.0163	0.4721	0.1966	0.0097	0.5914	0.1629
		Stylized Neural Painting [2]	**0.0087**	0.5030	0.1587	0.0048	0.5819	0.1516
		Im2oil [60]	0.0295	0.3803	0.1735	0.0168	0.4649	0.1549
		CNP [6]	0.0109	0.4899	**0.1576**	0.0061	0.5711	0.1481
		Ours	**0.0087**	**0.5412**	**0.1576**	**0.0043**	**0.6616**	**0.1279**
	1000	Paint Transformer [5]	0.0157	0.4756	0.1580	0.0102	0.5580	0.1550
		Learning to Paint [3]	0.0079	0.5430	0.1413	0.0042	0.6421	0.1258
		Semantic Guidance+RL [4]	0.0160	0.4723	0.1965	0.0094	0.5915	0.1623
		Stylized Neural Painting [2]	0.0061	0.5558	0.1403	0.0032	0.6362	0.1324
		Im2oil [60]	0.0176	0.4139	0.1430	0.0078	0.5290	0.1233
		CNP [6]	0.0076	0.5491	0.1391	0.0039	0.6172	0.1353
		Ours	**0.0059**	**0.5765**	**0.1288**	**0.0030**	**0.6594**	**0.1206**
	4000	Paint Transformer [5]	0.0103	0.5539	0.1386	0.0069	0.6139	0.1430
		Learning to Paint [3]	0.0052	0.6025	0.1276	0.0025	0.6810	0.1181
		Semantic Guidance+RL [4]	0.0159	0.4698	0.1970	0.0095	0.5891	0.1628
		Stylized Neural Painting [2]	0.0072	0.5692	0.1339	0.0048	0.6331	0.1303
		Im2oil [60]	0.0090	0.5208	0.1138	0.0035	0.6369	0.0948
		CNP [6]	0.0058	0.6178	0.1186	0.0025	0.6849	0.1135
		Ours	**0.0035**	**0.6771**	**0.0922**	**0.0016**	**0.7395**	**0.0821**
Bézier Stroke	250	Learning to Paint [3]	0.0092	0.5427	0.1845	0.0042	0.6791	0.1434
		Semantic Guidance+RL [4]	0.0133	0.4969	0.2057	0.0095	0.6164	0.1765
		Parameterized Brushstrokes [1]	0.0596	0.3907	0.1647	0.0809	0.3609	0.1691
		Ours	**0.0076**	**0.5744**	**0.1662**	**0.0031**	**0.7155**	**0.1301**
	1000	Learning to Paint [3]	0.0055	0.6191	0.1439	0.0020	0.7512	0.1083
		Semantic Guidance+RL [4]	0.0124	0.5075	0.2054	0.0081	0.6395	0.1734
		Parameterized Brushstrokes [1]	0.0518	0.3734	0.1456	0.0719	0.3319	0.1616
		Ours	**0.0044**	**0.6606**	**0.1189**	**0.0016**	**0.7784**	**0.0880**
	4000	Learning to Paint [3]	0.0030	0.7339	0.0828	0.0010	0.8229	0.0575
		Semantic Guidance+RL [4]	0.0123	0.5095	0.2064	0.0080	0.6402	0.1749
		Parameterized Brushstrokes [1]	0.0433	0.3767	0.1346	0.0589	0.3273	0.1543
		Ours	**0.0023**	**0.7818**	**0.0666**	**0.0008**	**0.8576**	**0.0477**

patch tokens, the Feature Extractor is based on ViT-s/16, and the Stroke head contains 1 cross-attention block and 4 self-attention blocks where the embedding dimension is 256. The shape of query Q is determined by the stroke type, *e.g.*, for oil stroke, we set Q as a $B \times 256 \times 8$ tensor, and for Bézier curve we set Q as $B \times 256 \times 13$, where B is the batch size, and 8 and 13 match the number of parameters for each stroke. For the neural renderer used during training, the output resolution is set to 128×128, following previous works [5], [6]. For the inference stage, we use the same texture renderer for our method and comparison methods, which performs geometric transformation on the original stroke template. The experimental results show that AttentionPainter can generalize to arbitrary scenes. We can use an improved block dividing strategy (detailed below) to divide the image into smaller blocks and then predict strokes for each block for more detailed reconstruction results or to handle images with higher resolutions.

2) For SDM, the embedding dimension of the attention block is 512 and the total number of diffusion steps is 1,000. For inpainting task, we use another AttentionPainter to encode the masked image into strokes and concatenate the extra strokes with the noised strokes as the diffusion module input. For the editing task, we use DexiNed model [62] to obtain the sketches and use cross-attention blocks to process the sketch hint.

TABLE 3
Efficiency study of different neural painting methods. Our method has the fastest inference speed.

Method	Type	Inference Time per Image ↓	
		Oil Stroke	Bézier Stroke
Stylized Neural Painting [2]	Optim-based	≈500s	-
Parameterized Brushstrokes [1]	Optim-based	-	≈210s
Im2oil [60]	Optim-based	≈100s	-
Paint Transformer [5]	Auto-regressive	0.27s	-
Learning to Paint [3]	RL	0.28s	0.26s
Semantic Guidance+RL [4]	RL	1.82s	1.80s
CNP [6]	RL	5.21s	-
Ours	End-to-end	**0.08s**	**0.08s**

Training Details: 1) We train AttentionPainter on CelebA [63] with 200 training epochs and the batch-size as 48. The learning rate warm ups to 6.25×10^{-4} in 6 epochs, and cosine decays towards 0 during the rest of the epochs. We use 1 NVIDIA RTX 4090 GPU to train AttentionPainter, and it takes about 20 hours for training. 2) We train SDM on the CelebAMask-HQ [64] dataset to verify the generation ability. For SDM, we set the batch size to 24, and the learning

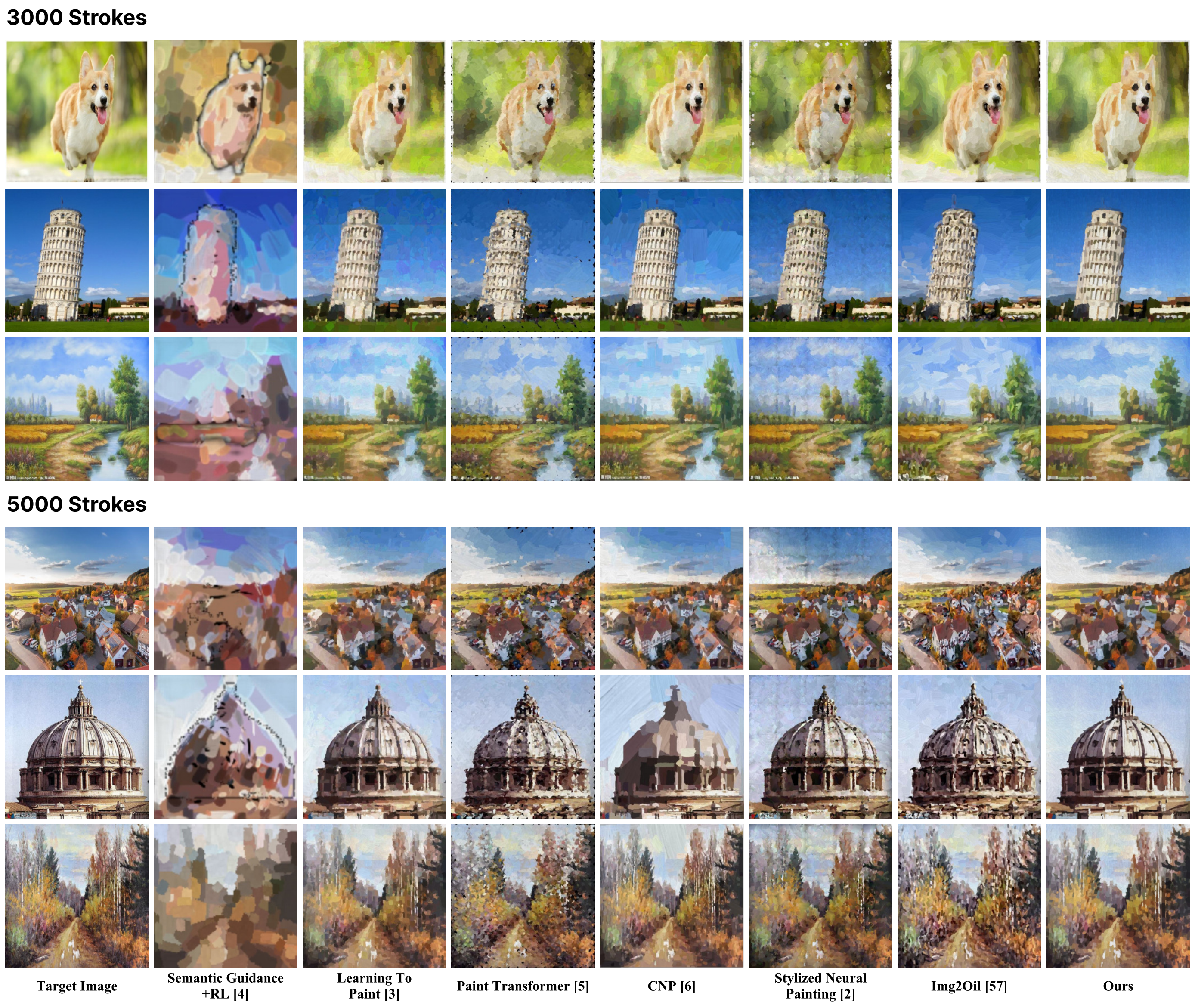


Fig. 9. Qualitative Comparison on Oil Stroke with state-of-the-art neural painting methods under 3,000 and 5,000 strokes, including Semantic Guidance + RL [4], Learning to Paint [3], Paint Transformer [5], CNP [6], Stylized Neural Painting [2], and Im2Oil [60]. Compared with previous methods, our AttentionPainter achieves the best reconstruction results.

rate to 4.8×10^{-5}. We use 1 NVIDIA RTX 4090 GPU to train the SDM, and it takes about 2 days for training.

6.2 Comparison with Other Neural Painting Methods

6.2.1 Quantitative Comparison

We quantitatively compare our method with 7 state-of-the-art methods, including Paint Transformer [5], Learning to Paint [3], Semantic Guidance+RL [4], Stylized Neural Painting [2], Parameterized Brushstrokes [1], Im2Oil [60] and Compositional Neural Painting (CNP) [6], with their official codes and models. We use $\mathcal{L}_2$, SSIM [65], and LPIPS [66] as metrics for our quantitative comparison.

We compare on 2 kinds of strokes (Oil stroke & Bézier curve stroke) with several stroke number settings (250, 1,000, 4,000) on CelebAMask-HQ [64] and ImageNet-mini [67]. We rearrange the comparison methods for the two stroke types into two groups: the comparison methods for **Oil stroke** include Paint Transformer [5], Learning to Paint [3], Semantic Guidance + RL [4], Stylized Neural Painting [2], Im2Oil [60], CNP [6]; while the comparison methods for **Bézier curve stroke** include Learning to Paint [3], Semantic Guidance + RL [4], and Parameterized Brushstrokes [1].

The quantitative comparisons with state-of-the-art methods are reported in Tab. 2, where our method outperforms previous methods in almost all settings. Especially, when the stroke number is large, our method has an obvious advantage over previous methods, where we get 0.0035 $\mathcal{L}_2$ on ImageNet-mini with 4,000 strokes.

We also calculate the Fréchet Inception Distance (FID) [68] score on the WikiArt dataset [69] to evaluate the similarity of painterly effects between generated images and real oil paintings. The results are shown in the supplemen-

TABLE 4
Ablation study of Stroke Predictor architecture. The attention-based painter performs better than convolution-based one.

Stroke Num	Architecture	ImageNet L2 ↓	ImageNet SSIM ↑	ImageNet LPIPS ↓	CelebAMask-HQ L2 ↓	CelebAMask-HQ SSIM ↑	CelebAMask-HQ LPIPS ↓
250	Convolution	0.0104	0.5329	0.1770	0.0063	0.6543	0.1458
	Attention	**0.0076**	**0.5744**	**0.1662**	**0.0031**	**0.7155**	**0.1301**
4000	Convolution	0.0049	0.6792	0.0909	0.0027	0.7680	0.0698
	Attention	**0.0023**	**0.7818**	**0.0666**	**0.0008**	**0.8576**	**0.0477**

TABLE 5
Training efficiency study of FSS. With FFS, we achieve a $13\times$ faster speed than the traditional stacking strategy.

Method	Training Time / Step forward	Training Time / Step backward	Training Time (200 epochs)	Validation on ImageNet L2 ↓	Validation on ImageNet SSIM ↑	Validation on ImageNet LPIPS ↓
w/o FSS	0.08s	6.34s	10.8 days	0.0032	0.6790	0.0855
w/ FSS	0.34s	0.14s	20.4 hours	0.0033	0.6729	0.0878

tary material.

6.2.2 Qualitative Comparison

Since our method can predict a large number of strokes in a short time, we mainly show the results with large stroke numbers (*e.g.*, 3,000, 5,000). The qualitative comparison results are shown in Fig. 9, where we compare with existing methods on oil strokes (more qualitative comparisons, including the comparison on Bézier curve strokes, are shown in the supplementary material Sec. 4 and Sec. 5). It can be seen that our results are closer to the target images, and have more detailed information than other methods. The Semantic Guidance + RL has a low reconstruction quality that loses the most detailed information. Other methods paint more details but still have problems with small strokes. CNP is capable of drawing tiny details, but sometimes it will be stuck in a loop of repainting the same area due to the unstable prediction of Reinforcement Learning, leading to very poor results. Our method is good at painting small strokes and can reconstruct more details compared with previous methods.

6.2.3 Efficiency Comparison

In this section, we mainly discuss the efficiency of the neural painting methods. Previous methods can be divided into optimization-based, RL, and auto-regressive methods. For each method, we measure the average inference time of a single image with 4,000 strokes on a single NVIDIA RTX 4090 GPU. The inference time is reported in Tab. 3. We conclude that the optimization-based method is too slow for applications. The RL/auto-regressive methods are faster than optimization-based methods, while our method achieves over three times faster than the previous methods (0.08s vs. 0.27s). Specifically, compared with the fastest previous method Paint Transformer, our AttentionPainter is not only three times faster than it, but also achieves much better similarity (given from 0.0103 to 0.0035 for L2). Compared with the best similarity method CNP, our AttentionPainter not only achieves even better similarity (from 0.0058 to 0.0035 L2), but also achieves 65 times faster inference speed.

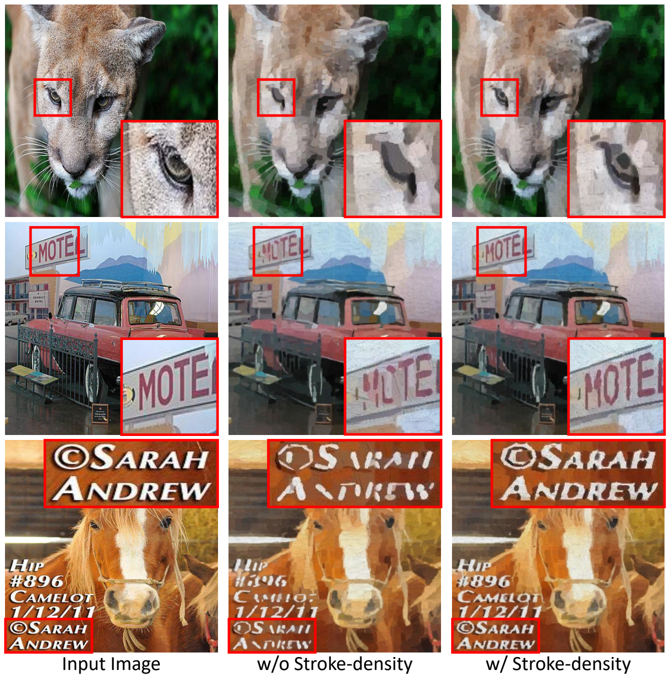


Fig. 10. Ablation study of Stroke-density Loss. With stroke-density loss, more details can be reconstructed.

6.3 Ablation Study

6.3.1 Ablation Study of Stroke Predictor Architecture

To validate the architecture of stroke predictor, we design another stroke prediction head based on convolution layers and FPN (denoted as ConvPainter). The ablation study results are shown in Tab. 4. Compared with ConvPainter, AttentionPainter has better performance on all settings, which indicates our attention-based network is more suitable for stroke prediction.

6.3.2 Training Efficiency Study of Fast Stroke Stacking

We analyze the influence of Fast Stroke Stacking (FSS), and the results are reported in Tab. 5. We compare the training speed of AttentionPainter under both with FSS and without FSS and set the batch size to 48. On a single NVIDIA RTX 4090, we achieve a 13× faster training speed by using FSS. Without FSS, it costs 6.42s per step; while with FSS, it only costs 0.48s per step. The saved training time mainly comes from the backpropagation process. FSS significantly accelerates the training process of AttentionPainter and makes it more extensible.

Moreover, to validate that our FSS improves training efficiency without affecting the final results, we further compare the performance of the model with and without FSS in Fig. 11. The results show that both models perform almost identically, with only minor visual differences that do not impact visual quality. Therefore, our FSS achieves a significant boost in training speed while maintaining the same training outcomes.

6.3.3 Ablation Study of Stroke-density Loss

In this section, we discuss the influence of stroke-density loss. The stroke-density loss helps the model to focus on more detailed information, and we illustrate the analysis

TABLE 6
Ablation study of Stroke-density Loss. The metric results show that our Stroke-density Loss works both for our AttentionPainter and SNP [2], effectively improving the performance.

Method	L2↓	SSIM↑	LPIPS↓
Ours w/o Stroke-density loss (1,000 strokes)	0.0058	0.5395	0.1318
Ours w/ Stroke-density loss (1,000 strokes)	**0.0054**	**0.6048**	**0.1152**
Ours w/o Stroke-density loss (4,000 strokes)	0.0036	0.6207	0.1016
Ours w/ Stroke-density loss (4,000 strokes)	**0.0033**	**0.6729**	**0.0878**
SNP w/o Stroke-density loss (500 strokes)	0.0101	0.4988	0.1623
SNP w/ Stroke-density loss (500 strokes)	**0.0090**	**0.5099**	**0.1524**
SNP w/o Stroke-density loss (1,000 strokes)	0.0093	0.5279	0.1488
SNP w/ Stroke-density loss (1,000 strokes)	**0.0084**	**0.5368**	**0.1406**

TABLE 7
Ablation study on hyper-parameters. The results show a trainable feature extractor which predicts 256 strokes during a single forward achieves the best performance, which is the setting adopted in our experiments.

Method	Validation on CelebAMask-HQ L2↓	SSIM↑	LPIPS↓
128 Strokes per forward	0.0026	0.7076	0.0985
256 Strokes per forward (Ours)	**0.0015**	**0.7343**	**0.0758**
512 Strokes per forward	0.0037	0.6554	0.1208
256 Strokes per forward, w/ E_{frozen}	0.0130	0.5199	0.1549

results and zoom in on the detail parts in Fig. 10. With the stroke-density loss, in the right column, more details of the animal eyes and complex characters can be reconstructed, while without the stroke-density loss, in the middle column, such details are hardly reconstructed. The quantitative results in Tab. 6 also demonstrate that the stroke-density loss helps better reconstruct the image, with better metric results.

6.3.4 Ablation Study of Hyperparameters

We explore two hyper-parameter settings, the number of predicted strokes during a single forward, and whether the feature extractor is frozen during training. The results are reported in Table 7. We adopt the setting with the best scores, *i.e.,* a trainable feature extractor which predicts 256 strokes during a single forward. Moreover, we also present the qualitative comparison results in Fig. 12. We find that 1) when predicting fewer strokes in a single forward (*e.g.,* 128), the number of strokes is too few to fully reconstruct the details. Although the number of divided patches is larger, since the limited strokes cannot adequately capture the fine details or complex structures within each patch, the results contain some artifacts. 2) When predicting more strokes during a single forward (*e.g.,* 512), the number of output stroke parameters doubles, which makes it harder for the stroke predictor to learn the correct stroke parameters due to increased model complexity, leading to poor generated results. As shown in Fig. 12, the model with 256 paths in a single forward pass generates the most detailed results, while the other two models exhibit reduced painting accuracy or display black slits in the output (second column, second row).

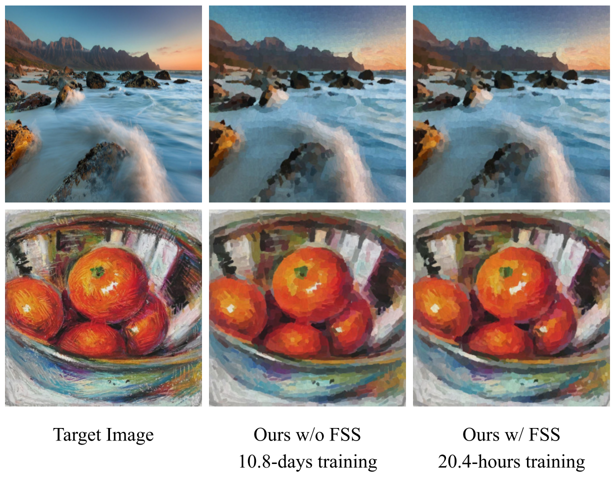


Fig. 11. Ablation study of FSS, where the difference in generation effects between the two models is very tiny, showing that the FSS largely accelerates training while having little impact on generation quality.

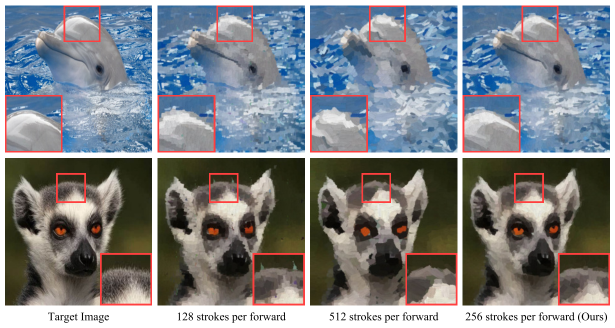


Fig. 12. Ablation study of predicting different numbers of strokes in a single forward step, where the model predicting 256 strokes in a single forward step achieves the best performance. The results of all ablated models in this figure are with 4,096 strokes in total.

6.4 User Study

To further validate the effectiveness of our method, we conduct a user study to measure the user preference between AttentionPainter and previous methods. We invite 40 volunteers who are mostly computer science related researchers to participate in our study, and each volunteer is asked to rank 20 groups of the oil paintings generated by our method and 4 previous methods from 1st to 5th (1st means the best). We ask the volunteers to take into account three aspects for their ranking: 1) The overall visual effect is good and similar to the target image; 2) The details of the target image are painted more carefully; 3) The painting looks more like human work, and the traces of machine painting are not obvious. Then we calculate the average ranking and ranking 1st rate, as is shown in Tab. 8. It can be seen that our method gets 1.39 average ranking and 77.91% ranking 1st rate, which greatly outperforms other methods, demonstrating that the paintings generated by our method are more preferred by users.

6.5 Analysis of Stroke-density Loss

In our AttentionPainter, we use Stroke-density Loss to reconstruct the detailed information. Surprisingly, we also find that our Stroke-density Loss can be applied to some previous methods to generate more attractive images. We apply the Stroke-density Loss to Stylized Neural Painting (SNP) [2], and it turns out to have obviously better results.

TABLE 8
User study results. Our method gets the highest average ranking, outperforming other methods.

Methods	Learning To Paint	Paint Transformer	Stylized Neural Painting	Im2Oil	Ours
Avg. Rank	2.61	3.76	3.34	3.90	**1.39**
Rank 1st (%)	6.27%	3.58%	9.10%	3.13%	**77.91%**

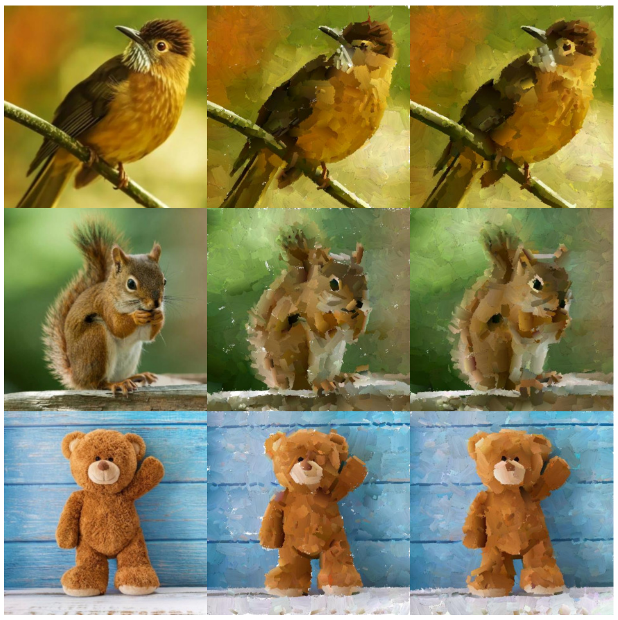


Fig. 13. Results of applying Stroke-density Loss to Stylized Neural Painting (SNP) under 500 Strokes.

For SNP, we combine the Stroke-density Loss with $\mathcal{L}_1$ Loss instead of only using $\mathcal{L}_1$ Loss. The qualitative comparison results are shown in Fig. 13. It can be seen that SNP with Stroke-density method can reconstruct more detailed information like animals' eyes and stripes. Moreover, SNP with Stroke-density Loss method can cover the whole canvas with strokes, while the original method may leave some regions blank. The quantitative results are shown in Table 6. The $\mathcal{L}_2$, SSIM [65], and LPIPS [66] results also demonstrate that Stroke-density Loss helps reconstruct the image better.

7 Limitations

Currently, although AttentionPainter is able to predict a large number of strokes in a single forward prediction, it still lacks the ability to reconstruct very densely detailed regions with sufficiently fine strokes under a limited number of strokes. In addition, although our method is able to reconstruct high-resolution images with the block dividing strategy, it inevitably ignores the global information of the whole image. Meanwhile, how to adaptively adjust the number of predicted strokes to achieve reconstruction at different levels of refinement or abstraction for input images of arbitrary resolution is also one of our future research directions.

Furthermore, the shapes of the generated strokes are constrained by the stroke's parameterized representation and the rendering process (Sec. 3.1). The texture renderer applies only rigid transformations (translation, scaling, rotation) to the stroke template, lacking the ability to bend or warp it. This limitation leads to generated strokes resembling the template shape and exhibiting less free-form variation than human-created strokes. Addressing this limitation, potentially through exploring non-rigid deformations or more complex stroke representations, remains an interesting direction for future work.

Additionally, the temporal order of the generated stroke sequence is not explicitly optimized to replicate the delicate workflow of a human artist, primarily due to the lack of sequential human painting process data as supervision. While this does not affect the final rendering quality, enabling human-aligned stroke ordering remains an important direction for future work.

Since our stroke predictor is a Transformer-based model, for the same input image, the output stroke parameters are the same (the stroke predictor has no randomness). Although we introduced 8 different input patterns by random rotation and flipping, the non-randomness defect of the model has not been fully addressed. How to introduce reasonable randomness into the model to promote more diverse stroke-painting results remains a future research direction.

8 Conclusion

We propose AttentionPainter, an efficient and adaptive model for single-step neural painting. We propose a stroke predictor that can predict a large number of strokes in a single forward pass, a fast stroke stacking algorithm to render the final image, and a stroke-density loss to better reconstruct details. We also design a stroke diffusion model as an application of our AttentionPainter, which can generate new content with strokes and demonstrate its applications in stroke-based inpainting and editing. AttentionPainter outperforms the state-of-the-art methods in terms of reconstruction quality, efficiency, and scalability, showing the promising potential of neural painting methods for the artists' community.

Acknowledgement

This work was supported by National Key R&D Program of China (2024YFE0115500) and National Natural Science Foundation of China (No. 62302297).

References

[1] D. Kotovenko, M. Wright, A. Heimbrecht, and B. Ommer, "Rethinking style transfer: From pixels to parameterized brushstrokes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 196–12 205.
[2] Z. Zou, T. Shi, S. Qiu, Y. Yuan, and Z. Shi, "Stylized neural painting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 689–15 698.
[3] Z. Huang, W. Heng, and S. Zhou, "Learning to paint with model-based deep reinforcement learning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 8709–8718.
[4] J. Singh and L. Zheng, "Combining semantic guidance and deep reinforcement learning for generating human level paintings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 387–16 396.

[5] S. Liu, T. Lin, D. He, F. Li, R. Deng, X. Li, E. Ding, and H. Wang, "Paint transformer: Feed forward neural painting with stroke prediction," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 6598–6607.

[6] T. Hu, R. Yi, H. Zhu, L. Liu, J. Peng, Y. Wang, C. Wang, and L. Ma, "Stroke-based neural painting and stylization with dynamically predicted painting region," in *Proceedings of the 31st ACM International Conference on Multimedia*, 2023, pp. 7470–7480.

[7] Y. Deng, F. Tang, W. Dong, C. Ma, X. Pan, L. Wang, and C. Xu, "Stytr2: Image style transfer with transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 326–11 336.

[8] Y. Zhang, N. Huang, F. Tang, H. Huang, C. Ma, W. Dong, and C. Xu, "Inversion-based style transfer with diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 10 146–10 156.

[9] G. Kwon and J. C. Ye, "Diffusion-based image translation using disentangled style and content representation," *arXiv preprint arXiv:2209.15264*, 2022.

[10] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2414–2423.

[11] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1501–1510.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[14] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[15] R. Yi, Y.-J. Liu, Y.-K. Lai, and P. L. Rosin, "APDrawingGAN: Generating artistic portrait drawings from face photos with hierarchical GANs," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 10 743–10 752.

[16] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013.

[17] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 2617–2680, 2021.

[18] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[19] T. Hu, J. Zhang, L. Liu, R. Yi, S. Kou, H. Zhu, X. Chen, Y. Wang, C. Wang, and L. Ma, "Phasic content fusing diffusion model with directional distribution consistency for few-shot model adaption," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2406–2415.

[20] F. Nolte, A. Melnik, and H. Ritter, "Stroke-based rendering: From heuristics to deep learning," *arXiv preprint arXiv:2302.00595*, 2022.

[21] A. Hertzmann, "A survey of stroke-based rendering." Institute of Electrical and Electronics Engineers, 2003.

[22] P. Haeberli, "Paint by numbers: Abstract image representations," in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 207–214.

[23] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pp. 453–460.

[24] O. Deussen, S. Hiller, C. Van Overveld, and T. Strothotte, "Floating points: A method for computing stipple drawings," in *Computer Graphics Forum*, vol. 19, no. 3. Wiley Online Library, 2000, pp. 41–50.

[25] A. Secord, "Weighted voronoi stippling," in *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, 2002, pp. 37–43.

[26] S. Simhon and G. Dudek, "Sketch interpretation and refinement using statistical models." in *Rendering Techniques*, 2004, pp. 23–32.

[27] P. Tresset and F. F. Leymarie, "Portrait drawing by paul the robot," *Computers & Graphics*, vol. 37, no. 5, pp. 348–363, 2013.

[28] X. Xing, C. Wang, H. Zhou, J. Zhang, Q. Yu, and D. Xu, "Diffsketcher: Text guided vector sketch synthesis through latent diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 15 869–15 889, 2023.

[29] Q. Wang, H. Deng, Y. Qi, D. Li, and Y.-Z. Song, "Sketchknitter: Vectorized sketch generation with diffusion models," in *The Eleventh International Conference on Learning Representations*, 2023.

[30] P. Selinger, "Potrace: a polygon-based tracing algorithm," 2003.

[31] Y.-K. Lai, S.-M. Hu, and R. R. Martin, "Automatic and topology-preserving gradient mesh generation for image vectorization," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, pp. 1–8, 2009.

[32] H. Zhu, J. Ian Chong, T. Hu, R. Yi, Y.-K. Lai, and P. L. Rosin, "SAMVG: A multi-stage image vectorization model with the segment-anything model," in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 4350–4354.

[33] T. Hu, R. Yi, B. Qian, J. Zhang, P. L. Rosin, and Y.-K. Lai, "SuperSVG: Superpixel-based scalable vector graphics synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 24 892–24 901.

[34] P. Litwinowicz, "Processing images and video for an impressionist effect," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 407–414.

[35] M. Shiraishi and Y. Yamaguchi, "An algorithm for automatic painterly rendering based on local source image approximation," in *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 2000, pp. 53–58.

[36] Y.-Z. Song, D. Pickup, C. Li, P. Rosin, and P. Hall, "Abstract art by shape classification," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 8, pp. 1252–1263, 2013.

[37] A. Hertzmann, "Paint by relaxation," in *Proceedings. Computer Graphics International 2001*. IEEE, 2001, pp. 47–54.

[38] P. O'Donovan and A. Hertzmann, "AniPaint: Interactive painterly animation from video," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 3, pp. 475–487, 2011.

[39] J. P. Collomosse and P. M. Hall, "Genetic paint: A search for salient paintings," in *Workshops on Applications of Evolutionary Computation*. Springer, 2005, pp. 437–447.

[40] H. W. Kang, C. K. Chui, and U. K. Chakraborty, "A unified scheme for adaptive stroke-based rendering," *The Visual Computer*, vol. 22, pp. 814–824, 2006.

[41] R. Nakano, "Neural painters: A learned differentiable constraint for generating brushstroke paintings," *arXiv preprint arXiv:1904.08410*, 2019.

[42] N. Zheng, Y. Jiang, and D. Huang, "StrokeNet: A neural painting environment," in *International Conference on Learning Representations*, 2018.

[43] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3d reasoning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 7708–7717.

[44] D. Ha and D. Eck, "A neural representation of sketch drawings," *CoRR*, vol. abs/1704.03477, 2017. [Online]. Available: http://arxiv.org/abs/1704.03477

[45] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[46] Y. Ganin, T. Kulkarni, I. Babuschkin, S. A. Eslami, and O. Vinyals, "Synthesizing programs for images using reinforced adversarial learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1666–1675.

[47] J. F. Mellor, E. Park, Y. Ganin, I. Babuschkin, T. Kulkarni, D. Rosenbaum, A. Ballard, T. Weber, O. Vinyals, and S. Eslami, "Unsupervised doodling and painting with improved spiral," *arXiv preprint arXiv:1910.01007*, 2019.

[48] J. Singh, C. Smith, J. Echevarria, and L. Zheng, "Intelli-Paint: Towards developing more human-intelligible painting agents," in *European Conference on Computer Vision*. Springer, 2022, pp. 685–701.

[49] H. Mei, Y. Liu, Z. Wei, D. Zhou, X. Wei, Q. Zhang, and X. Yang, "Exploring dense context for salient object detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 3, pp. 1378–1389, 2022.

[50] P. Schaldenbrand and J. Oh, "Content masked loss: Human-like brush stroke planning in a reinforcement learning painting agent," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 1, 2021, pp. 505–512.

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

[52] X. Li, X. Tan, Z. Zhang, Y. Xie, and L. Ma, "Point mask transformer for outdoor point cloud semantic segmentation," *Computational Visual Media*, 2025.

[53] T.-M. Li, M. Lukáč, M. Gharbi, and J. Ragan-Kelley, "Differentiable vector graphics rasterization for editing and learning," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–15, 2020.

[54] X.-C. Liu, Y.-C. Wu, and P. Hall, "Painterly style transfer with learned brush strokes," *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[55] B. Tang, T. Hu, Y. Du, R. Yu, and L. Ma, "Curved-stroke-based neural painting and stylization through thin plate spline interpolation," *Scientia Sinica Informationis*, vol. 54, no. 2, pp. 301–315, 2024.

[56] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 11, no. 6, pp. 567–585, 1989.

[57] J. Hu, X. Xing, Z. Zhang, J. Zhang, and Q. Yu, "Vectorpainter: A novel approach to stylized vector graphics synthesis with vectorized strokes," *arXiv preprint arXiv:2405.02962*, 2024.

[58] Y. Song, S. Huang, C. Yao, X. Ye, H. Ci, J. Liu, Y. Zhang, and M. Z. Shou, "ProcessPainter: Learn painting process from sequence data," *arXiv preprint arXiv:2406.06062*, 2024.

[59] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[60] Z. Tong, X. Wang, S. Yuan, X. Chen, J. Wang, and X. Fang, "Im2Oil: Stroke-based oil painting rendering with linearly controllable fineness via adaptive sampling," in *ACM International Conference on Multimedia*. ACM, 2022, pp. 1035–1046.

[61] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[62] X. Soria, A. Sappa, P. Humanante, and A. Akbarinia, "Dense extreme inception network for edge detection," *Pattern Recognition*, vol. 139, p. 109461, 2023.

[63] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 3730–3738.

[64] C.-H. Lee, Z. Liu, L. Wu, and P. Luo, "MaskGAN: Towards diverse and interactive facial image manipulation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5549–5558.

[65] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[66] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.

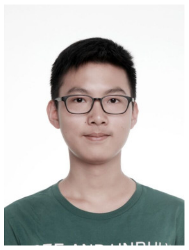
[67] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[68] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local nash equilibrium," *Neural Information Processing Systems,Neural Information Processing Systems*, Jan 2017.

[69] W. Tan, C. Chan, H. Aguirre, and K. Tanaka, "Improved ArtGAN for conditional synthesis of natural image and artwork," *IEEE Transactions on Image Processing,IEEE Transactions on Image Processing*, Aug 2017.

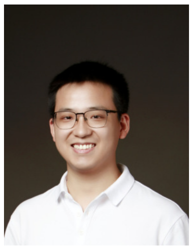
Yizhe Tang is a master student with the School of Computer Science, Shanghai Jiao Tong University, China. He received his BEng degree from the School of Computer Science and Engineering, Central South University, China, in 2023. His research interests include computer vision and computer graphics.

Yue Wang received the bachelor's degree in the Department of Remote Sensing and Geography Information from Sun Yat-sen University, Guangzhou, China, in 2021, and the master's degree in the Department of Computer Science and Technology from Shanghai Jiao Tong University, Shanghai, China, in 2024.



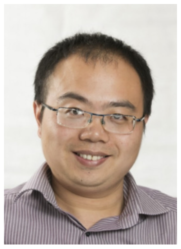
Teng Hu is a PhD student with the School of Computer Science, Shanghai Jiao Tong University, China. He received his BEng degree from the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China, in 2022. His research interests include computer vision and computer graphics.

Ran Yi is an Associate Professor with the School of Computer Science, Shanghai Jiao Tong University. She received the BEng degree and the PhD degree from Tsinghua University, China, in 2016 and 2021. Her research interests mainly fall into computer vision, computer graphics, and machine intelligence. She serves as a Program Committee Member/Reviewer for SIGGRAPH, CVPR, ICCV, IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Visualization and Computer Graphics, IEEE Transactions on Image Processing, and IJCV.



Xin Tan received the B.Eng. degree in automation from Chongqing University, China, in 2017, and the dual Ph.D. degrees in computer science from Shanghai Jiao Tong University and the City University of Hong Kong. He is currently a Research Professor (Zijiang Young Scholar) with the School of Computer Science and Technology, East China Normal University, China. His research interests include computer vision and deep learning. He serves as a Program Committee Member/Reviewer for CVPR, ICCV, ECCV, AAAI, IJCAI, IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Transactions on Image Processing, and IJCV. He is also serving as the associate editor for Pattern Recognition and The Visual Computer.

Lizhuang Ma received the B.S. and Ph.D. degrees from the Zhejiang University, China in 1985 and 1991, respectively. He is now a Distinguished Professor, Ph.D. Tutor, and the Head of the Digital Media and Computer Vision Laboratory at the School of Computer Science, Shanghai Jiao Tong University, China. He was a Visiting Professor at the Frounhofer IGD, Darmstadt, Germany in 1998, and was a Visiting Professor at the Center for Advanced Media Technology, Nanyang Technological University, Singapore from 1999 to 2000. He has published more than 200 academic research papers in both domestic and international journals. His research interests include computer aided geometric design, computer graphics, scientific data visualization, computer animation, digital media technology, and theory and applications for computer graphics, CAD/CAM.



Yu-Kun Lai is a Professor at School of Computer Science and Informatics, Cardiff University, UK. He received his B.S. and PhD degrees in Computer Science from Tsinghua University, in 2003 and 2008 respectively. His research interests include computer graphics, computer vision, geometric modeling and image processing. For more information, visit https://users.cs.cf.ac.uk/Yukun.Lai/

Paul L. Rosin is a Professor at School of Computer Science and Informatics, Cardiff University, UK. Previous posts include lecturer at the Department of Information Systems and Computing, Brunel University London, UK, research scientist at the Institute for Remote Sensing Applications, Joint Research Centre, Ispra, Italy, and lecturer at Curtin University of Technology, Perth, Australia. His research interests include low level image processing, performance evaluation, shape analysis, facial analysis, cellular automata, non-photorealistic rendering and cultural heritage. For more information, visit http://users.cs.cf.ac.uk/Paul.Rosin/