視覺語言模型輔助之風格感知向量草圖補全

秦孝媛*
國立臺灣大學
臺灣
r12725026@ntu.edu.tw

沈奕超*
東京大學
日本
ichaoshen@g.ecc.u-tokyo.ac.jp

邱奕庭
國立臺灣大學
臺灣
r13922018@csie.ntu.edu.tw

陳炳宇
國立臺灣大學
臺灣
robin@ntu.edu.tw

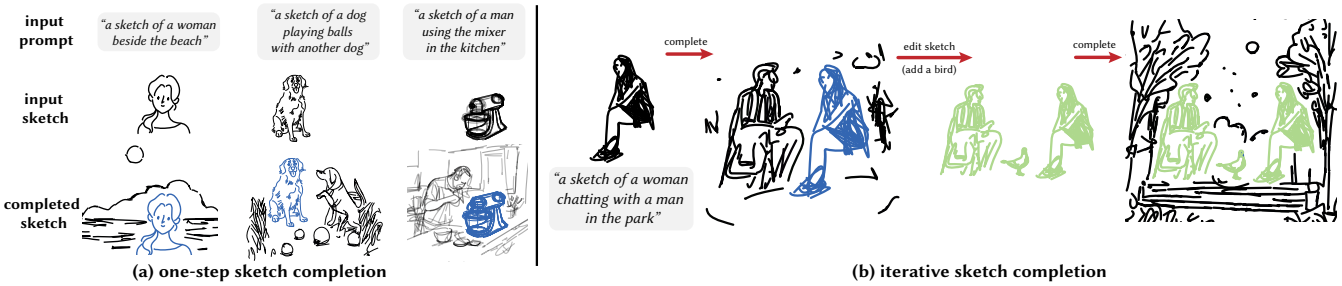


Figure 1: (a) Given an input prompt and a partial sketch, our method completes the partial sketch by accurately representing the input prompt and maintain various styles in the sketch. (b) Users iteratively employ the AutoSketch to create a complex sketch. For example, after the first complete sketch is generated, the user can decide to retain the strokes representing the man and woman, add some strokes representing a bird, and our method completes the sketch by adding strokes depicting the trees and grass. (The blue and green strokes denote the first and second iterations of the input partial sketches, respectively.)

ABSTRACT

自動補全描述複雜場景的部分草圖，例如「一位女性在公園與男性聊天」，在實際應用中具有高度實用性。然而，現有草圖生成方法多從空白開始作畫，難以延續原始草圖的風格。

為解決此問題，本研究提出一種風格感知的向量草圖補全方法，可適應多樣化的草圖風格。我們的核心觀察為：以自然語言描述草圖風格，有助於在補全過程中保留原始風格。因此，我們利用預訓練視覺語言模型（VLM）將部分草圖的風格轉換為自然語言，引導新筆劃生成，延續既有風格。

整體流程如下：首先，透過 VLM 擷取原始草圖的風格描述，並將其納入輸入提示中，作為生成引導。接著，根據融合風格資訊的提示生成新筆劃，維持整體風格一致性。最後，我們再次利用 VLM 自動產出風格調整程式碼，微調筆劃細節，使草圖更貼近目標風格。

在多種草圖風格與輸入提示下，我們與現有方法進行比較，並進行廣泛的消融實驗與定性定量評估。實驗結果顯示，本方法在風格延續與補全能力方面表現優異，能有效支援多樣化的草圖場景。

*Both authors contributed equally to this research.

CGW '25, July 10–11, 2025, Taipei, Taiwan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**.

KEYWORDS

Vector Sketches, Sketch Completion, Style-Aware, Scene Completion, Bézier Curves

ACM Reference Format:
秦孝媛, 沈奕超, 邱奕庭, and 陳炳宇. . 視覺語言模型輔助之風格感知向量草圖補全. In *Proceedings of (CGW '25).* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Sketching has long been a key form of visual expression that rapidly communicates ideas and expresses concepts. Even people with little experience can easily sketch simple objects and ideas. However, creating sketches that depict complex concepts and scenes remains a significant challenge for many. Typically, individuals begin sketching by creating a rough partial sketch but often struggle to turn this into a final complex sketch that maintains a unique style. One common challenge individuals face is the difficulty of vividly illustrating the interactions and compositions between the objects or subjects in the desired scene.

Although ShadowDraw [Lee et al. 2011] provides real-time guidance when sketching simple objects, it does not adequately address the above challenges that individuals encounter when trying to portray elaborate scenes in a consistent style. More recent

sketch generation methods [Vinker et al. 2023; Xing et al. 2023] make it easier to generate intricate sketches from scratch using user-provided text prompts or reference images. However, these methods lack the capacity to consider the user-provided partial sketches, thus creating two major issues: redundant strokes and style inconsistency. First, such methods tend to generate strokes that duplicate elements of the user-provided partial sketch. Second, all strokes are of the same style; the styles of the generated strokes are not adapted to match those of the user-provided partial sketch. These methods thus do not automatically complete partial sketches provided by users.

To address these issues, we propose AutoSketch, a novel style-aware vector sketch completion method that accepts both a text prompt and a partial sketch as input. The method completes the partial sketch by generating strokes that illustrate missing elements or concepts, while preventing the creation of redundant strokes and ensuring that the style aligns with that of the input partial sketch. Following [Xing et al. 2023], we begin by optimizing strokes based on a guidance image generated by a pretrained ControlNet model conditioned on the input partial sketch. We introduce a mask penalty to ensure that the generated strokes do not overlap with those of the input partial sketch, so there are no redundant strokes.

However, stroke optimization alone does not ensure that the completed sketch is satisfactory, because the style of the input partial sketch is not considered. This raises two main issues. First, the styles of the guidance images generated by ControlNet often do not match those of the partial sketches. Second, the styles of the generated strokes may not align with those of the input partial sketch. This underscores the importance of two tasks, i.e., "adding style descriptions to the input prompt before inputting it to the ControlNet" and "adjusting the styles of the generated strokes to ensure alignment with these descriptions".

Based on these observations, we utilized a pretrained vision-language model (VLM) in conjunction with the stroke optimization process. First, we leverage the VLM to extract style descriptions from the input partial sketch and then incorporate these descriptions into the input prompt. This enables the ControlNet to generate guidance images that are very similar to the input partial sketch, improving the completed sketch. Second, we employ the VLM to generate an executable code that adjusts the strokes in SVG format, thus enhancing the style coherence of the final completed sketch. The main reason for using the VLM for style adjustment is that the variety of sketch styles complicates the task of defining appropriate parameterizations to capture all potential styles effectively. Consequently, adjusting the styles using an optimization-based method becomes challenging. Although it is possible to use the VLM to directly adjust strokes, this often results in stroke loss. Moreover, given the token limitations, existing VLMs typically handle only a small number of strokes. The use of the VLM to generate an executable style adjustment code overcomes these challenges, resulting in a more stylistically consistent sketch without losing content.

We compare our results with those of existing methods across various sketch styles and prompts. Extensive quantitative and qualitative evaluations revealed that the completed sketches generated by our method better preserve the styles of the input partial sketches and more accurately represent the contents specified in the prompts.

2 RELATED WORK

2.1 Vector Sketch Generation

Previous studies [Eitz et al. 2012; Ha and Eck 2017; Sangkloy et al. 2016] have collected sketch datasets of amateur sketches that sought to realistically depict everyday objects, while OpenSketch [Gryaditskaya et al. 2019] contains professional sketches of product designs. Existing studies used these sketch datasets and various deep learning models [Ha and Eck 2017; Lin et al. 2020; Ribeiro et al. 2020; Zhou et al. 2018] to generate sketch sequences. However, given their reliance on these sketch datasets, such methods generally generate sketches of only simple objects.

Recently, with the development of differentiable rasterizers [Li et al. 2020], novel methods [Gal et al. 2024; Vinker et al. 2023, 2022; Xing et al. 2023] that employ the "synthesis through optimization" paradigm, have emerged. Such methods typically optimize stroke geometry and appearance using priors obtained from large pretrained models such as CLIP [Radford et al. 2021], and text-to-image [Rombach et al. 2022] and -video [Wang et al. 2023] models. However, such methods are usually generate sketches from scratch based solely on prompts; they do not complete partial sketches.

2.2 Visual Content Completion

Given the challenges associated with visual content creation, it would be useful to prepare only some partial content and then apply a method that automatically or semi-automatically completes the rest of the work. Previous works developed autocompletion systems for various visual content creation tasks using repetitive elements and the editing history, such as in 3D sculpting [Peng et al. 2018] and animation sculpting [Peng et al. 2020]. Other methods aim to complete sketches [Liu et al. 2019] or afford real-time guidance during freehand drawing [Lee et al. 2011]. Such approaches typically use category-specific priors learned from sketch datasets or edge maps of real-world photographs. However, these methods either require the editing history of the user or are limited to relatively simple objects. In contrast, our method uses diffusion priors to complete large missing regions and complex concepts in a partial sketch, and ensures that the style of the completed sketch aligns with that of the original partial sketch.

2.3 LLM-based Sketch and SVG Editing

Recent advancements in large language models (LLMs) have enabled extensive research on vector graphic generation and editing [Cai et al. 2023; Nishina and Matsui 2024; Zou et al. 2024]. This progress has led to the development of new benchmarks and frameworks aimed at evaluating enhancing the capabilities of LLMs. For example, SketchAgent [Vinker et al. 2024] leverages an LLM to iteratively generate sketch strokes based on text prompts, while StarVector [Rodriguez et al. 2023] presents a multimodal LLM designed to vectorize raster images. Other previous works [Tang et al. 2024; Wu et al. 2023; Xing et al. 2024] incorporate specialized tokenization methods or modular architectures to improve LLMs ' understanding of SVG structures, enabling advanced tasks such as text-guided icon synthesis and SVG manipulation.

Despite their successes, existing methods mainly focus on generating or editing vector graphics from scratch and fail to maintain

style consistency between existing strokes and newly generated ones. Also, they typically focus on depicting simple objects or concepts, such as individual man-made objects or animals. In contrast, our approach completes partial sketches for complex scenes and concepts that contains better object interactions and compositions in a coherent style.

3 OVERVIEW

In Figure 2, we illustrate the overview of our method. Our method takes a text prompt $\mathcal{P}_{\text{input}}$ and a partial sketch $\mathcal{S}_{\text{input}}$ as inputs. The prompt describes the content to be illustrated in the completed sketch, but the user-provided partial sketch represents only some of the content described in the prompt. The output is a completed sketch $\mathcal{S}_{\text{complete}} = \mathcal{S}_{\text{input}} \cup \mathcal{S}_{\text{opt}}$ that fully represents the content of $\mathcal{P}_{\text{input}}$. Our method has two stages: *style-agnostic sketch completion* and *sketch style adjustment*.

In the first stage, the goal is to optimize a set of parametric strokes that, when combined with the user-provided partial sketch, ensure that the complete sketch represents the content of $\mathcal{P}_{\text{input}}$ without consideration of sketch styles. First, we stylize $\mathcal{P}_{\text{input}}$ by leveraging a large vision-language model (VLM) to produce style descriptions $\mathcal{P}_{\text{aug}}$ of the given partial sketch $\mathcal{S}_{\text{input}}$, i.e., $\mathcal{P}_{\text{stylized}} = \{\mathcal{P}_{\text{input}} \cup \mathcal{P}_{\text{aug}}\}$ (Figure 2(a)). Then, we optimize the parameters of $\mathcal{S}_{\text{opt}}$ using a diffusion prior conditioned on the stylized text prompt $\mathcal{P}_{\text{stylized}}$ (Figure 2(b)) and obtains $\bar{\mathcal{S}}_{\text{complete}}$.

In the second stage, the goal is to adjust the styles of $\bar{\mathcal{S}}_{\text{complete}}$ to ensure a coherent style across the final sketch. We task the VLM using a carefully crafted prompt that contains the completed sketch of the first stage in SVG format and the text prompt $\mathcal{P}_{\text{stylized}}$. The VLM then generates executable code that adjusts the styles of the new strokes in $\bar{\mathcal{S}}_{\text{complete}}$ to the style of the original partial sketch.

4 STAGE 1: STYLE-AGNOSTIC SKETCH COMPLETION

Inspired by previous works [Vinker et al. 2023; Xing et al. 2023], we take a "synthesis through optimization" approach. We optimize the parameters of a group of strokes by leveraging the prior of a pretrained text-to-image (T2I) model. Unlike previous works, our method employs a user-provided partial sketch $\mathcal{S}_{\text{input}}$ as an additional input. Therefore, we employ a conditional T2I model (e.g., ControlNet Scribble[1]) to optimize the stroke parameters.

4.1 Prompt Stylization

Although the conditional T2I model generates images that match the input text prompt $\mathcal{P}_{\text{input}}$, the styles of the generated images are often not those of the given partial sketch $\mathcal{S}_{\text{input}}$. It is likely that the style of the optimized sketch will deviate from that of $\mathcal{S}_{\text{input}}$. To address this issue, we first stylize the input prompt $\mathcal{P}_{\text{input}}$, i.e., augment style descriptions to $\mathcal{P}_{\text{input}}$ using the VLM. Specifically, we render the partial sketch $\mathcal{S}_{\text{input}}$ into a raster image and then request the VLM to generate textual descriptions capturing both the semantic and stylistic cues of the rendered image. Then, we augment the style descriptions $\mathcal{P}_{\text{aug}}$ to the input prompt. The final prompt becomes: i.e., $\mathcal{P}_{\text{stylized}} = \{\mathcal{P}_{\text{input}} \cup \mathcal{P}_{\text{aug}}\}$.

[1]https://huggingface.co/lllyasviel/sd-controlnet-scribble

4.2 Stroke Optimization for Completion

Using the stylized prompt $\mathcal{P}_{\text{stylized}}$, we generate strokes that fill the empty regions of the user-provided partial sketch. We define the strokes to be optimized as $\mathcal{S}_{\text{opt}} = \{s_1, \ldots, s_n\}$, and the stroke parameterization as:

$$s_i = \left\{ \{p_i^j\}_{j=1}^4, o_i, w_i \right\}, \tag{1}$$

where $\{p_i^j\}_{j=1}^4$ are the control points of a cubic Bézier curve, o_i denotes an opacity attribute, and w_i denotes the stroke width. Specifically, we first generate an guidance image $\mathcal{I}_{\text{guide}}$ using a conditional T2I model that is based on the stylized prompt $\mathcal{P}_{\text{stylized}}$. Then, we optimize the control points to obtain a sketch that is consistent with both the stylized prompt $\mathcal{P}_{\text{stylized}}$ and the guidance image $\mathcal{I}_{\text{guide}}$ (Figure 3). Specifically, at iteration t, we rasterize the strokes using a differentiable rasterizer R to generate the raster sketch: $\mathcal{I}_{\text{sketch}} = R(\mathcal{S}_{\text{complete}})$, and we optimize the following objective function when updating the strokes:

$$L_{\text{all}} = \alpha \underbrace{\left(1 - \text{sim}\big(\phi_{\text{vis}}(\mathcal{I}_{\text{sketch}}), \phi_{\text{vis}}(\mathcal{I}_{\text{guide}})\big)\right)}_{\text{CLIP visual alignment}} \tag{2}$$

$$+ \beta \underbrace{\left(LPIPS(\mathcal{I}_{\text{sketch}}, \mathcal{I}_{\text{guide}})\right)}_{\text{perceptual loss}} \tag{3}$$

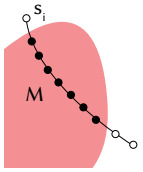
$$+ \gamma \underbrace{\sum_{x_k \in \mathbf{x}} \mathbb{1}\left[\mathbf{M}(x_k) = 1\right]}_{\text{Overlap penalty}}, \tag{4}$$

where α, β, γ control the relative importance of the three terms. The first term measures the visual alignment between the guidance image $\mathcal{I}_{\text{guide}}$ and the raster sketch $\mathcal{I}_{\text{sketch}}$ using the CLIP visual encoder $\phi_{\text{img}}(\cdot)$, where $\text{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$ is the cosine similarity. Additionally, we further minimize the LPIPS loss to enhance the visual similarity of $\mathcal{I}_{\text{sketch}}$ and $\mathcal{I}_{\text{guide}}$.

To ensure that the strokes do not overlap with those of the user-provided sketch $\mathcal{S}_{\text{input}}$, we introduce an overlap penalty term. Specifically, we first define a binary mask $\mathbf{M}$ that encodes the regions in $\mathcal{S}_{\text{input}}$ where strokes already exist and should thus not be altered:

$$\mathbf{M}(x) = \begin{cases} 1, & \text{if pixel } x \text{ belongs to strokes in } \mathcal{S}_{\text{input}}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Then, we sample 10 points on each stroke $s_i \in \mathcal{S}_{\text{opt}}$. For each sample point x_k, if that point falls in M (the filled black circles in the inset), we introduce a penalty, where $\mathbb{1}[\cdot]$ in Equation 4 is the indicator function.



After optimizing L_{all}, we obtain the style-agnostic completed sketch $\bar{\mathcal{S}}_{\text{complete}}$ by combining the optimized strokes $\bar{\mathcal{S}}_{\text{opt}}$ with those of $\mathcal{S}_{\text{input}}$. The strokes in $\bar{\mathcal{S}}_{\text{complete}}$ contain the overall content in $\mathcal{P}_{\text{input}}$, but the styles are not coherent.

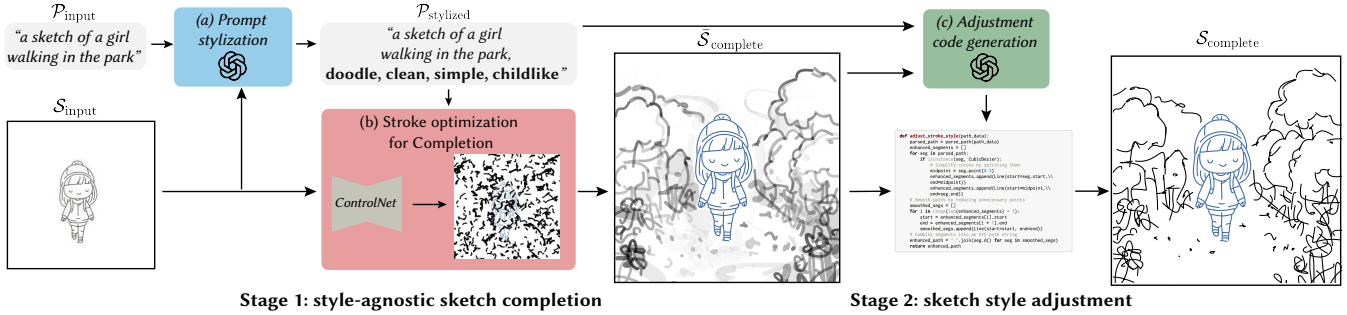


Figure 2: Overview of our method. Given a user-provided prompt $\mathcal{P}_{\text{input}}$ and a partial sketch $\mathcal{S}_{\text{input}}$, our method first (a) stylizes the input prompt by augmenting it using style descriptions generated by the VLM (bold text). Using the stylized prompt $\mathcal{P}_{\text{stylized}}$, the method then performs (b) stroke optimization to generate strokes that fill the missing regions, thus ensuring that the style-agnostic completed sketch $\bar{\mathcal{S}}_{\text{complete}}$ can fully represents the content of the user-provided prompt. To align the styles of $\bar{\mathcal{S}}_{\text{complete}}$ and $\mathcal{S}_{\text{input}}$, we (c) instruct the VLM to generate an executable style adjustment code that modifies the strokes of $\bar{\mathcal{S}}_{\text{complete}}$. Finally, we obtain a final completed sketch $\mathcal{S}_{\text{complete}}$ wherein the styles of the strokes are aligned to those of the $\mathcal{S}_{\text{input}}$.

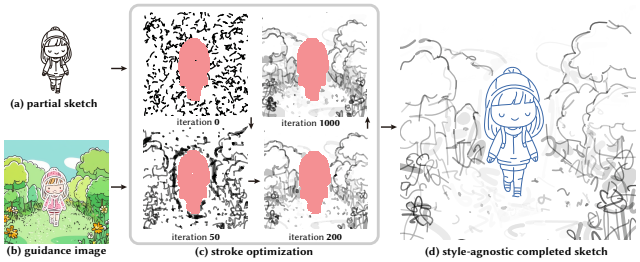


Figure 3: Overview of stroke optimization. Given (a) the user-provided partial sketch and (b) the guidance image generated by the conditional T2I model, our method (c) iteratively updates the position, opacity, and width of each stroke. This ensures that the resulting style-agnostic completed sketch is in visual alignment with the guidance image but does not overlap with the user-provided partial sketch.

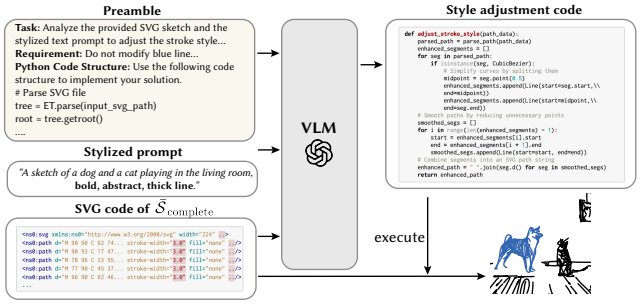


Figure 4: Overview of VLM style adjustment code generation. The complete system prompt we provided to the VLM contains a preamble, the stylized prompt $\mathcal{P}_{\text{stylized}}$, and the SVG code of the style-agnostic completed sketch $\bar{\mathcal{S}}_{\text{complete}}$. We fed it into the VLM, which generates the style adjustment code. Finally, we execute the style adjustment code on the SVG code.

5 SKETCH STYLE ADJUSTMENT

In Stage 1, we effectively complete the empty areas, but this does not guarantee that the strokes of $\bar{\mathcal{S}}_{\text{complete}}$ will exhibit global stylistic coherence. The variety of sketch styles complicates the process of defining appropriate parameterizations that can capture all potential styles. To address this, we utilize style descriptions extracted from the VLM to guide the style adjustment of $\bar{\mathcal{S}}_{\text{complete}}$. Intuitively, we can represent $\bar{\mathcal{S}}_{\text{complete}}$ in SVG codes and request the VLM to edit the codes to achieve the desired style adjustment. However, several challenges then arise, given the limitations of existing VLMs. First, many such VLMs handle only limited numbers of tokens, restricting the number of curves that can be included in $\bar{\mathcal{S}}_{\text{complete}}$. Second, such VLMs often hallucinate. In other words, they may generate strokes absent in $\bar{\mathcal{S}}_{\text{complete}}$ or delete many genuine strokes in $\bar{\mathcal{S}}_{\text{complete}}$.

To address the abovementioned issues, we request the VLM to generate an executable style adjustment code C that can operate on $\bar{\mathcal{S}}_{\text{complete}}$, as illustrated in Figure 4. Specifically, we provide the VLM with the following information:

- A preamble that contains the instructions for the task.
- A symbolic representation of the style-agnostic completed sketch $\bar{\mathcal{S}}_{\text{complete}}$ (e.g., SVG code).
- The stylized text prompt $\mathcal{P}_{\text{stylized}}$.
- A snippet of the skeleton style adjustment code that specifies how to read and write an SVG file, and defines a section for which the VLM should fill in the code for adjustment of $\bar{\mathcal{S}}_{\text{complete}}$.

The VLM then completes the missing part of the skeleton code snippet, yielding a style adjustment code that specifies how to adjust the newly generated strokes (e.g., stroke width, curvature, or smoothness) to match the style of $\mathcal{S}_{\text{input}}$. For example, the VLM can generate simple code to make the strokes thicker:

```
def adjust_stroke_style(path_data):
    parsed_path = parse_path(path_data)
    if "bold" in stylized_prompt_lower:
        width_scale_factor *= 1.2
    for seg in parsed_path:
        seg = seg.width * width_scale_factor
```

Meanwhile, the VLM generate the following complex code to simplify the path structures of $\bar{S}_{\text{complete}}$:

```
def adjust_stroke_style(path_data):
    parsed_path = parse_path(path_data)
    enhanced_segments = []
    for seg in parsed_path:
        if isinstance(seg, CubicBezier):
            # Simplify curves by splitting them
            midpoint = seg.point(0.5)
            enhanced_segments.append(\\
            Line(start=seg.start, end=midpoint))
            enhanced_segments.append(\\
            Line(start=midpoint, end=seg.end))
    # Smooth paths by reducing unnecessary points
    smoothed_segs = []
    for i in range(len(enhanced_segments) - 1):
        start = enhanced_segments[i].start
        end = enhanced_segments[i + 1].end
        smoothed_segs.append(Line(start=start, end=end))
    # Combine segments into an SVG path string
    enhanced_path = " ".join(seg.d() for seg in \\
                    smoothed_segs)
    return enhanced_path
```

Finally, we execute C on $\bar{S}_{\text{complete}}$ to obtain S_{complete}. Please see supplement for the details of the preamble we provided to the VLM and other adjustment codes generated by the VLM.

6 EXPERIMENT

6.1 Implementation Details and Performance

In this work, we use the GPT-4o model [Hurst et al. 2024] as the VLM, which extracts style descriptions and generates style adjustment codes. We implement the first stage of our method using PyTorch [Paszke et al. 2019]. The Adam [Kingma and Ba 2015] optimizer is used to optimize the strokes. The first stage, consisting of 1,000 iterations, takes approximately 5 minutes to complete, while the second stage requires around 3 minutes when a sketch contains 512 strokes. For all computations, we used a PC with an Intel i7-12700 CPU and an NVIDIA RTX 4080 GPU.

6.2 Comparison with Existing Methods

We qualitatively and quantitatively compare our method to conditional T2I models used to generate sketches and line drawings, namely ControlNet LineArt[2] and ControlNet Scribble[3] both qualitatively and quantitatively. In Figure 5, we show the results generated by our method and the two methods using identical user-provided partial sketch and stylized prompts. The partial sketches were prepared by re-tracing publicly shared sketches and clipart, or were generated by other sketch generation methods, such as CLIPasso [Vinker et al. 2022]. The results of the Control-based methods (Figure 5(b,c)) often exhibit incomplete or inconsistent content. Additionally, these methods tend to apply style transformations that deviate significantly from those of the provided sketches, sometimes entirely altering the styles. In contrast, our method consistently generates completed sketches that faithfully represent the contents of the text prompts. Also, the styles of the generated strokes and the provided sketches are consistent.

[2]https://huggingface.co/ControlNet-1-1-preview/control_v11p_sd15_lineart
[3]https://huggingface.co/lllyasviel/sd-controlnet-scribble

	Visual			Text
	LPIPS↓	DreamSim↓	DINO↑	VQA score ↑
ControlNet LineArt	0.285	0.486	0.217	0.854
ControlNet Scribble	0.416	0.532	0.202	0.965
Our method	0.258	0.270	0.525	0.954

Table 1: Quantitative evaluation results. We compare our method to two ControlNet-based methods employing metrics that focus on visual and textual similarities. Our method consistently outperforms the other methods across the various visual metrics and achieves comparable performance with other methods on textual metric. We highlight the best result for each metric.

To further validate the effectiveness of our method in terms of preserving the sketch styles and completing the content, we gather an evaluation set containing 10 sketches and perform two types of quantitative evaluation. First, we use commonly use visual and text metrics to evaluate the performance of our method. However, since these metrics are typically not used for evaluating the sketch completion task and have their own limitation, we additionally conduct an user evaluation which further validate our method.

Evaluation using existing metrics. In terms of visual metrics, we used LPIPS [Zhang et al. 2018], DINO [Caron et al. 2021], and DreamSim [Fu et al. 2023]. These metrics were used to measure the style consistencies and image similarities between the input partial sketches and the generated completed sketches. However, visual metrics alone cannot be used to sufficiently evaluate performance because input partial sketches that do not receive additional strokes tend to achieve the best scores. Therefore, we also assess the alignment between the content of each completed sketch and the input prompt using the VQA score [Lin et al. 2024] to eliminate the bias associated with visual metrics. The VQA score measures prompt-image alignment on compositional prompts more effectively than the CLIP score [Radford et al. 2021] and is more closely aligned with human judgement. As shown in Table 1, our method significantly outperforms the other methods across all visual metrics and achieves a comparable score on the text metric.

User evaluation. We conducted a user evaluation to further validate that our method generates sketches whose styles match those in user-provided partial sketches and depict complete content in the input prompt. We use the same evaluation set used in Section 6.2 generated by our method, ControlNet LineArt, and ControlNet Scribble. Participants evaluated the quality of the generated completed sketches by conducting pairwise comparisons. For each input sketch and prompt, we created two comparative pairs, "Ours vs. ControlNet LineArt" and "Ours vs. ControlNet Scribble", resulting in 20 pairs for comparison. During each comparison, two completed sketches were shown side by side in random order, along

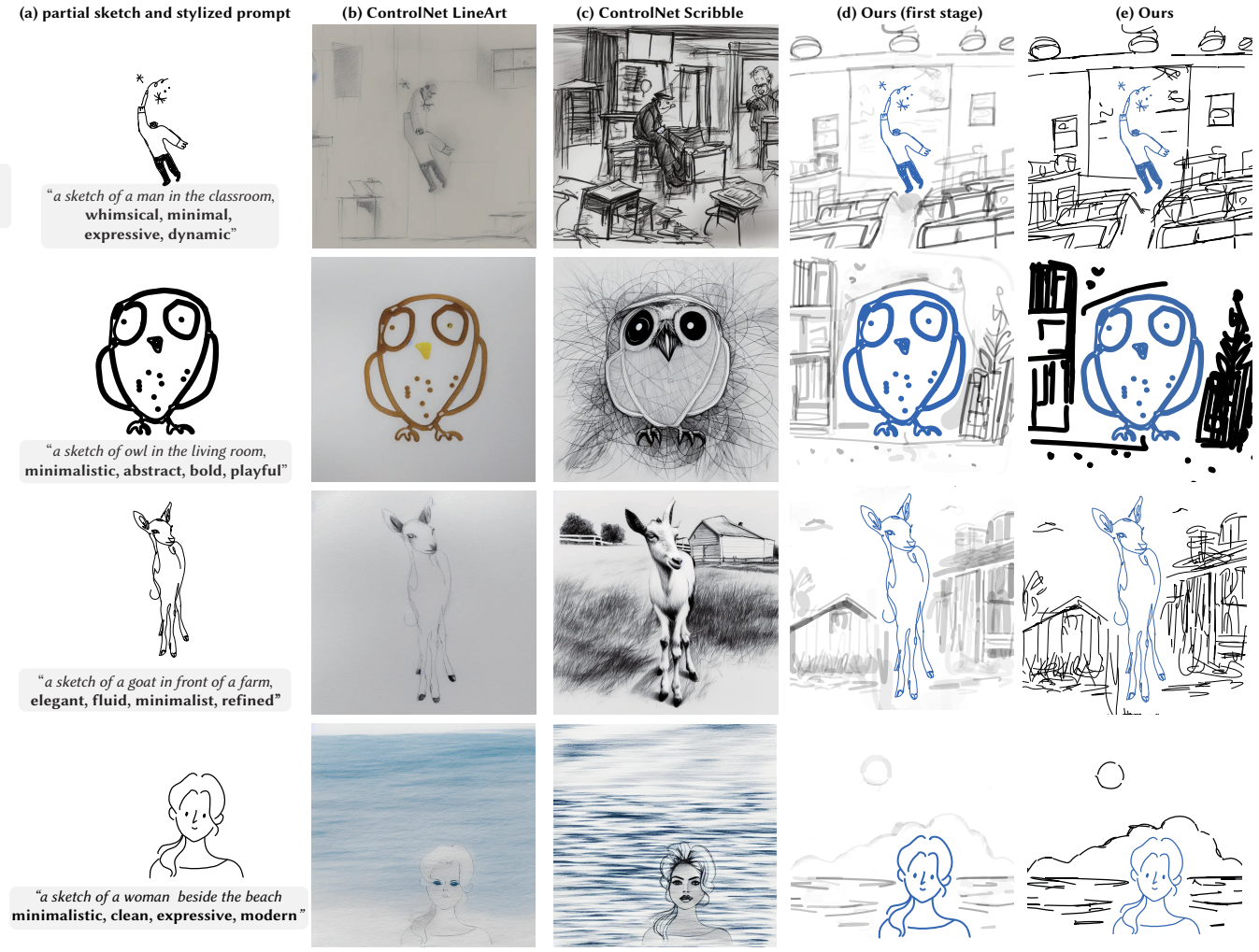


Figure 5: Comparision with existing methods. Given (a) the input partial sketch and the stylized prompt, (b) the results generated by ControlNet LineArt often do not accurately depict the content of the input prompt. (c) ControlNet Scribble generates completed sketches with more details of the input prompt compared to ControlNet LineArt, but the partial sketches are sometimes missing, and the styles deviate significantly from those of the input partial sketches. (d) Completed sketches generated by the first stage of our method accurately represent the contents of the input prompts, but the styles are inconsistent. (e) Our full method further adjusts the styles of all strokes to match the styles of the partial sketches. (bold text:style descriptions.)

with their inputs. Participants were asked to judge the sketches based on two criteria: "How well they preserved the *styles* of the input partial sketch" and "How effectively they depicted the *content* of the input prompt". Each comparison was evaluated by 25 different participants. As shown in Table 2, the participants preferred our method for both criteria.

	Style			Content		
	Ours	Others	neither	Ours	Others	neither
vs. LineArt	98.4%	0.8%	0.8%	84.0%	9.6%	6.4%
vs. Scribble	96.8%	2.4%	0.8%	64.8%	30.4%	4.4%

Table 2: User evaluation results. Compared to the two ControlNet-based methods, the participants consistently preferred the completed sketches generated by our method in terms of both the style preservation and content depiction criteria. ("Others" denote to either ControlNet LineArt or Scribble.) We highlight the best result.

6.3 Diverse Sketch Scenario

Iterative sketch completion. Sketching is often an iterative process, where users may want to introduce new details by adding new strokes or modifying the original prompt. Our method enables users to achieve iterative sketch completion by retaining some strokes

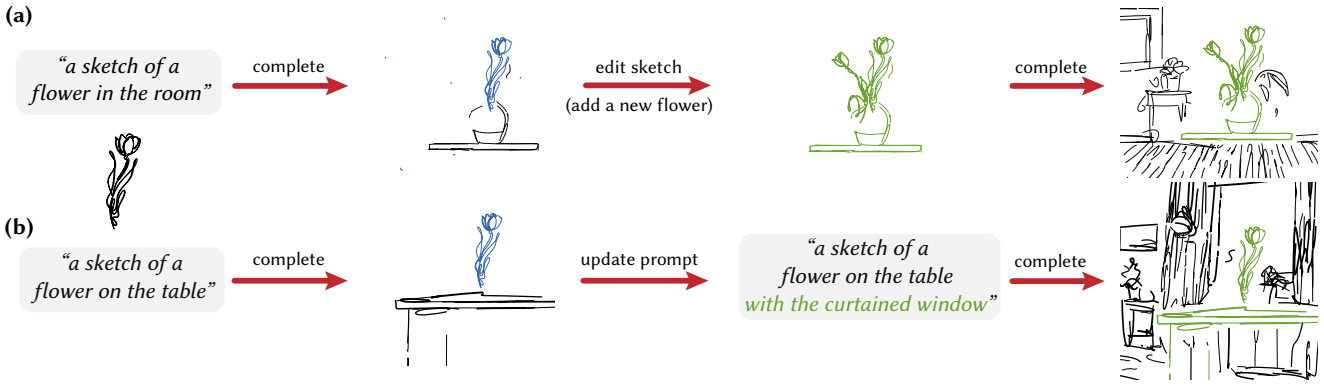


Figure 6: Examples of iterative sketch completion. After the initial sketch completion, the user can keep the strokes generated in the first completion and (a) edit the sketch or (b) update the input prompt . Then, our method will complete the sketch once again to add more details. (The blue and green line denotes the input partial sketch of the first and second iteration, respectively.)

from the completed sketch and incorporating new ones (Figure 1(b) and Figure 6(b)), or by updating the input prompt (Figure 6(a)).

Sketches with different prompts, or distinct sketches. Users may seek to employ a variety of partial sketches when generating sketches that depict the same content in the input prompt. As shown in Figure 7(a), the completed sketches represent similar content but in different styles. Additionally, as shown in Figure 7(b), the completed sketches created using different input prompts can represent distinct contents but share a similar style.

Figure 7: Various sketch scenarios. (a) Given the same prompt, our method can generate completed sketches that depict the same content in different styles that align with those of the user-provided partial sketches. (b) Given the same partial sketch, our method can generate different completed sketches representing the contents of various prompts.

7 LIMITATIONS AND FUTURE WORK

Reliance on large pretrained models. Our method uses two pretrained models: ControlNet to generate the guidance images and a VLM to stylize input prompt and create the style adjustment code.

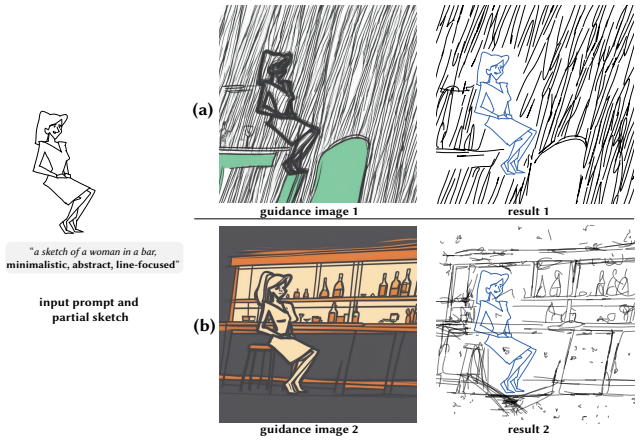


Figure 8: Limitation. Our method cannot generate completed sketch that accurately depict content in the input prompt and maintain the styles in the partial sketch with (a) a broken guidance image generated by the ControlNet or (b) a broken style adjustment code generated by the VLM.

It is thus inevitable that these models may occasionally generate unsatisfied results. For example, as shown in Figure 8(a), when the guidance image lacks the content specified in the input prompt, our method cannot generate strokes that accurately depict the desired content. Also, the style adjustment code generated by the VLM may not accurately adjust the strokes to represent the content and preserve the style, even when the guidance image is clear (Figure 8(b)).

Non-interactive generation. Our method allows users to simply co-create sketches using machine learning methods. However, currently, stroke optimization and adjustment code generation require a few minute. This limitation hinders our ability to provide users with interactive feedback and completed sketch. In the future, we will explore multi-scale stroke optimization, which will allow us

to provide users with previews and enable interactive sketch completion.

8 CONCLUSION

In this paper, we introduce AutoSketch, a style-aware vector sketch completion method that accommodates diverse sketch styles by leveraging both the recognition and generation capabilities of a pretrained vision-language model (VLM). Our method allows users to provide only a partial sketch, and our method will complete missing content specified in the input prompt by optimizing strokes and stroke style adjustment. We demonstrate that the style descriptions extracted by the VLM from the partial sketch enable our method to accurately complete the sketch, reflecting both the intended content and the style in the input partial sketch. Extensive experiment results indicate our method is effective across various sketch scenarios.

REFERENCES

Mu Cai, Zeyi Huang, Yuheng Li, Utkarsh Ojha, Haohan Wang, and Yong Jae Lee. 2023. Leveraging large language models for scalable vector graphics-driven image understanding. *arXiv preprint arXiv:2306.06094* (2023).

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. 2021. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9650–9660.

Mathias Eitz, James Hays, and Marc Alexa. 2012. How Do Humans Sketch Objects? *ACM Trans. Graph. (Proc. SIGGRAPH)* 31, 4 (2012), 44:1–44:10.

Stephanie Fu, Netanel Tamir, Shobhita Sundaram, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. 2023. DreamSim: Learning New Dimensions of Human Visual Similarity using Synthetic Data. In *Advances in Neural Information Processing Systems*, Vol. 36. 50742–50768.

Rinon Gal, Yael Vinker, Yuval Alaluf, Amit Bermano, Daniel Cohen-Or, Ariel Shamir, and Gal Chechik. 2024. Breathing Life Into Sketches Using Text-to-Video Priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Accepted.

Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Fredo Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 38, 6 (2019), 232.

David Ha and Douglas Eck. 2017. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477* (2017).

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Yong Jae Lee, C Lawrence Zitnick, and Michael F Cohen. 2011. Shadowdraw: real-time user guidance for freehand drawing. *ACM Transactions on Graphics (ToG)* 30, 4 (2011), 1–10.

Tzu-Mao Li, Michal Lukáč, Gharbi Michaël, and Jonathan Ragan-Kelley. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6 (2020), 193:1–193:15.

Hangyu Lin, Yanwei Fu, Xiangyang Xue, and Yu-Gang Jiang. 2020. Sketch-bert: Learning sketch bidirectional encoder representation from transformers by self-supervised learning of sketch gestalt. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6758–6767.

Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. 2024. Evaluating Text-to-Visual Generation with Image-to-Text Generation. *arXiv preprint arXiv:2404.01291* (2024).

Fang Liu, Xiaoming Deng, Yu-Kun Lai, Yong-Jin Liu, Cuixia Ma, and Hongan Wang. 2019. Sketchgan: Joint sketch completion and recognition with generative adversarial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5830–5839.

Kunato Nishina and Yusuke Matsui. 2024. SVGEditBench: A Benchmark Dataset for Quantitative Assessment of LLM's SVG Editing Capabilities. *arXiv preprint arXiv:2404.13710* (2024).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Mengqi Peng, Li-yi Wei, Rubaiat Habib Kazi, and Vladimir G Kim. 2020. Autocomplete animated sculpting. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 760–777.

Mengqi Peng, Jun Xing, and Li-Yi Wei. 2018. Autocomplete 3D sculpting. *ACM Transactions on Graphics (ToG)* 37, 4 (2018), 1–15.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*. PMLR, 8748–8763.

Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. 2020. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 14153–14162.

Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. 2023. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556* (2023).

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.

Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. 2016. The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies. *ACM Transactions on Graphics (proceedings of SIGGRAPH)* (2016).

Zecheng Tang, Chenfei Wu, Zekai Zhang, Mingheng Ni, Shengming Yin, Yu Liu, Zhengyuan Yang, Lijuan Wang, Zicheng Liu, Juntao Li, and Duan Nan. 2024. StrokeNUWA: Tokenizing Strokes for Vector Graphic Synthesis. *arXiv preprint arXiv:2401.17093* (2024).

Yael Vinker, Yuval Alaluf, Daniel Cohen-Or, and Ariel Shamir. 2023. Clipascene: Scene sketching with different types and levels of abstraction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4146–4156.

Yael Vinker, Ehsan Pajouheshgar, Jessica Y. Bo, Roman Christian Bachmann, Amit Haim Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. 2022. CLIPasso: Semantically-Aware Object Sketching. *ACM Trans. Graph.* 41, 4, Article 86 (jul 2022), 11 pages. https://doi.org/10.1145/3528223.3530068

Yael Vinker, Tamar Rott Shaham, Kristine Zheng, Alex Zhao, Judith E Fan, and Antonio Torralba. 2024. SketchAgent: Language-Driven Sequential Sketch Generation. *arXiv preprint arXiv:2411.17673* (2024).

Jiuniu Wang, Hangjie Yuan, Dayou Chen, Yingya Zhang, Xiang Wang, and Shiwei Zhang. 2023. Modelscope text-to-video technical report. *arXiv preprint arXiv:2308.06571* (2023).

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. 2023. IconShop: Text-Guided Vector Icon Synthesis with Autoregressive Transformers. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–14.

Ximing Xing, Juncheng Hu, Guotao Liang, Jing Zhang, Dong Xu, and Qian Yu. 2024. Empowering LLMs to Understand and Generate Complex Vector Graphics. *arXiv preprint arXiv:2412.11102* (2024).

XiMing Xing, Chuang Wang, Haitao Zhou, Jing Zhang, Qian Yu, and Dong Xu. 2023. DiffSketcher: Text Guided Vector Sketch Synthesis through Latent Diffusion Models. In *Thirty-seventh Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=CY1xatvEQj

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.

Tao Zhou, Chen Fang, Zhaowen Wang, Jimei Yang, Byungmoon Kim, Zhili Chen, Jonathan Brandt, and Demetri Terzopoulos. 2018. Learning to sketch with deep q networks and demonstrated strokes. *arXiv preprint arXiv:1810.05977* (2018).

Bocheng Zou, Mu Cai, Jianrui Zhang, and Yong Jae Lee. 2024. Vgbench: Evaluating large language models on vector graphics understanding and generation. *arXiv preprint arXiv:2407.10972* (2024).