AutoSketch: VLM-assisted Style-Aware Vector Sketch Completion

HSIAO-YUAN CHIN*, National Taiwan University, Taiwan
I-CHAO SHEN*†, The University of Tokyo, Japan
YI-TING CHIU, National Taiwan University, Taiwan
ARIEL SHAMIR, Reichman University, Israel
BING-YU CHEN†, National Taiwan University, Taiwan

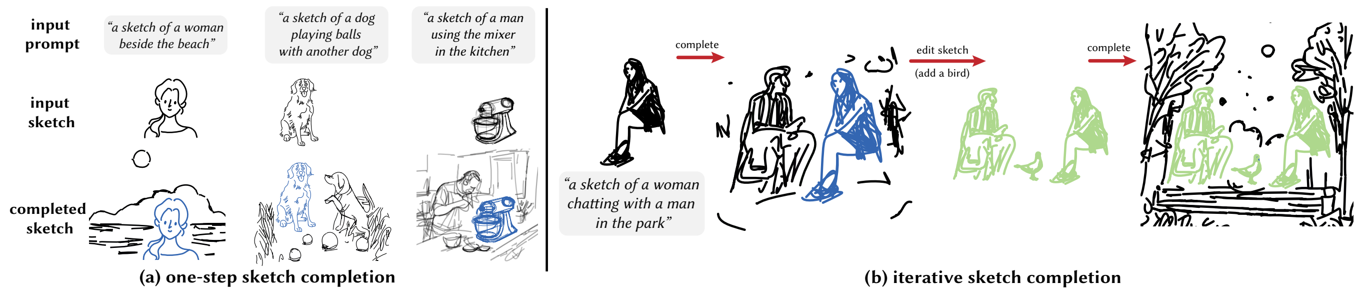

input prompt
"a sketch of a woman beside the beach"
"a sketch of a dog playing balls with another dog"
"a sketch of a man using the mixer in the kitchen"
input sketch
completed sketch
(a) one-step sketch completion
complete
edit sketch
(add a bird)
complete
"a sketch of a woman chatting with a man in the park"
(b) iterative sketch completion

Fig. 1. (a) Given an input prompt and a sketch, our method completes the input sketch by accurately representing the input prompt and maintain the style of the input partial sketch. (b) Users iteratively employ AutoSketch to create a complex sketch. For example, after the first completed sketch is generated, the user decide to retain the strokes representing the man and woman, draw 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 sketches.)

Sketches are an important medium of expression and recently many works concentrate on automatic sketch creations. One such ability very useful for amateurs is text-based completion of a partial sketch to create a complex scene, while preserving the style of the partial sketch. Existing methods focus solely on generating sketch that match the content in the input prompt in a predefined style, ignoring the styles of the input partial sketches, e.g., the global abstraction level and local stroke styles. To address this challenge, we introduce AutoSketch, a style-aware vector sketch completion method that accommodates diverse sketch styles and supports iterative sketch completion. AutoSketch completes the input sketch in a style-consistent manner using a two-stage method. In the first stage, we initially optimize the strokes to match an input prompt augmented by style descriptions extracted from a vision-language model (VLM). Such style descriptions lead to non-photorealistic guidance images which enable more content to be depicted through new strokes. In the second stage, we utilize the VLM to adjust the strokes from the previous stage to adhere to the style present in the input partial sketch through an iterative style adjustment process. In each iteration, the VLM identifies a list of style differences between the input sketch and the strokes generated in the previous stage, translating these differences into adjustment codes to modify the strokes. We compare our method with existing methods using various sketch styles and prompts, perform extensive ablation studies and qualitative and quantitative evaluations, and demonstrate that AutoSketch can support diverse sketching scenarios.

CCS Concepts: • **Computing methodologies → Computer graphics**.

Additional Key Words and Phrases: Vector Sketches, Sketch Completion, Style-Aware, Scene Completion, Bézier Curves

ACM Reference Format:
Hsiao-Yuan Chin, I-Chao Shen, Yi-Ting Chiu, Ariel Shamir, and Bing-Yu Chen. 2025. AutoSketch: VLM-assisted Style-Aware Vector Sketch Completion. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3757377.3763828

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 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.

Recent text-based sketch generation methods [Jain et al. 2023; Qu et al. 2023; Xing et al. 2023] leverage user-provided text prompts to generate intricate sketches either from scratch or progressively. However, these methods do not adequately consider input partial sketches, thus creating two major issues. First, they often generate redundant strokes that duplicate elements already present in the input partial sketch. Second, they ignores the styles of the input sketch, e.g., the global level of abstraction and the local stroke styles, such as stroke thickness, smoothness, curvature, opacity. Therefore,

*Both authors contributed equally to this research.
†Joint corresponding author.

Authors' Contact Information: Hsiao-Yuan Chin, National Taiwan University, Taiwan, r12725026@ntu.edu.tw; I-Chao Shen, The University of Tokyo, Japan, jdilyshen@gmail.com; Yi-Ting Chiu, National Taiwan University, Taiwan, r13922018@csie.ntu.edu.tw; Ariel Shamir, Reichman University, Israel, arik@idc.ac.il; Bing-Yu Chen, National Taiwan University, Taiwan, robin@ntu.edu.tw.

the styles of the generated strokes do not align with those of the input partial sketch.

To address these issues, we propose AutoSketch, a novel style-aware vector sketch completion method that takes a text prompt and a partial sketch as input. Our method completes the partial sketch by generating strokes that illustrate missing elements or concepts according to the prompt, while preventing the creation of redundant strokes and ensuring that the style aligns with that of the input sketch.

We observed that the stroke optimization method using fixed text augmentation adopted by [Jain et al. 2023; Qu et al. 2023; Xing et al. 2023] often results in unsatisfied completed sketches. In terms of content, the generated guidance image often contains blurred area or lacks clear boundaries, leading to some elements being overlooked during the sketch generation process. In terms of style, the use of a single type of stroke parameterization causes the style of the generated strokes to misalign with those of the input sketch.

Based on this observations, we utilize a pretrained vision-language model (VLM) along with a stroke optimization process. In the first stage, we augment the input prompt with VLM-generated style descriptions based on the input partial sketch. This augmentation allows ControlNet [Zhang et al. 2023] to generate guidance images that exhibit some non-photorealistic rendering styles for stroke generation. Next, we optimize the strokes based on these guidance images. To resolve the redundant strokes issue, we introduce an overlap penalty to ensure that the generated strokes do not overlap with those of the input partial sketch. In the second stage, we employ the VLM to iteratively adjust the styles of the optimized sketch. We task the VLM with identifying style differences between the input sketch and the most recent optimized sketch. Then, we ask the VLM to generate adjustment code based on the identified style differences and apply them to the latset optimized sketch to obtain the final sketch. While the VLM typically adjusts styles effectively in one iteration, this iterative process ensures effective style adjustment due to the inherent instability of the VLM.

The advantage of using a VLM for style adjustment, compared to existing optimization-based methods, is its ability to support both continuous and discrete adjustments, including stroke deletion or splitting. Although prior work [Cai et al. 2023; Vinker et al. 2024] has shown that it is technically feasible to prompt a pretrained VLM to generate adjusted strokes directly, these methods are constrained by token limitations, which prevents them from generating complex sketches with a large number of strokes. By generating *style adjustment code* with the VLM, we can overcome these challenges, resulting in a more stylistically consistent sketch without losing essential content.

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

2 Related Work

2.1 Vector Sketch Generation

Previous studies [Eitz et al. 2012; Ha and Eck 2018; 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 2018; 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.

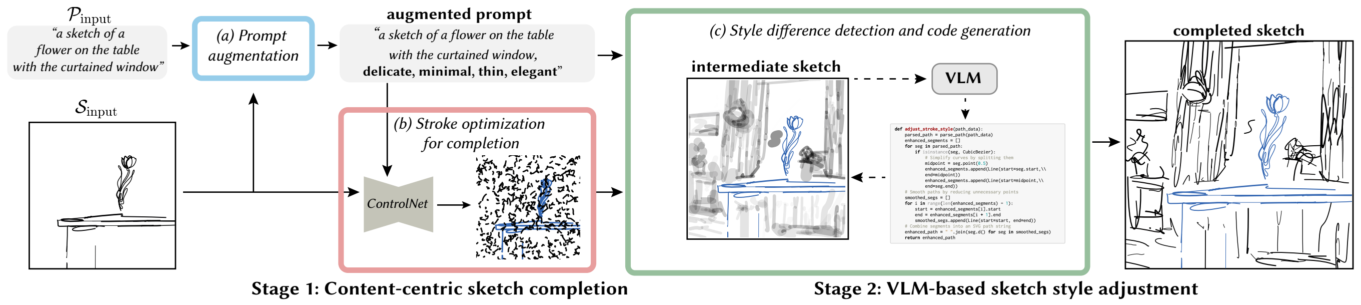
Recently, novel methods [Frans et al. 2022; Gal et al. 2024; Qu et al. 2023; Vinker et al. 2023, 2022; Xing et al. 2023] employs the "synthesis-through-optimization" paradigm have emerged. These methods typically optimize stroke geometry and appearance using priors derived from large pretrained models such as CLIP [Radford et al. 2021], and text-to-image [Rombach et al. 2022] and text-to-video [Wang et al. 2023] models. However, these methods either create sketches from scratch to fit the input prompt or modify sketches based on the updated prompt [Mo et al. 2024] in a predefined style, while neglecting the styles present in the input sketches.

2.2 Sketch Styles

In the past works, style is often discussed in terms of two components: local curve level and global abstraction level. At the local curve level, style is characterized by a combination of geometry (shape) and appearance (e.g., strokes and textures). For example, Li *et al.* [2013] identified geometric curve styles in a set of shapes, while Berger *et al.* [2013] analyzed both the geometric and appearance sketch styles, as well as global abstraction level for a specific artist in portrait sketching. Recently, Vinker *et al.* [2023] introduced a method for generating scene sketches that vary in levels of abstraction. Our method considers both the global abstraction level and the local geometric and appearance styles of sketches, ensuring that the style aligns between the input sketch and generated 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; Wu et al. 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, StarVector [Rodriguez et al. 2025] 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. However, many of these methods rely on additional large-scale training data to finetuning LLMs. In contrast, SketchAgent [Vinker et al. 2024] and Chat2SVG [Wu et al. 2024] use off-the-shelf LLMs without finetuning but is limited to simple concepts with a small number of strokes. Our method, without any finetuning, can handle partial sketches involving a larger number of strokes, enabling complex scenes with complex object interactions and compositions.


$\mathcal{P}_{\text{input}}$
"a sketch of a flower on the table with the curtained window"
(a) Prompt augmentation
augmented prompt
"a sketch of a flower on the table with the curtained window, delicate, minimal, thin, elegant"
$\mathcal{S}_{\text{input}}$
(b) Stroke optimization for completion
ControlNet
(c) Style difference detection and code generation
intermediate sketch
VLM
completed sketch
Stage 1: Content-centric sketch completion
Stage 2: VLM-based sketch style adjustment

```
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
```

Fig. 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 augmented prompt, the method then performs (b) stroke optimization to generate strokes that fill the missing regions, thus ensuring that the intermediate sketch can fully represents the content of the input prompt. To align the styles of the intermediate sketch and the input sketch, we (c) iteratively instruct the VLM to identify style differences and translate them into style adjustment codes that modifies the strokes of the intermediate sketch. Finally, we obtain a final completed sketch.

3 Overview

In Figure 2, we illustrate the overview of our method. Our method takes as input a text prompt $\mathcal{P}_{\text{input}}$ and a partial sketch $\mathcal{S}_{\text{input}}$. The text prompt describes the content to be illustrated in the completed sketch, and the input partial sketch represents only partial 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 $\mathcal{P}_{\text{input}}$ in a coherent style. Our method has two stages: *content-centric 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 input partial sketch, ensure that the complete sketch represents the content of $\mathcal{P}_{\text{input}}$ without considering the sketch styles. First, we augment $\mathcal{P}_{\text{input}}$ by leveraging a large vision-language model (VLM) to produce style descriptions of the input partial sketch $\mathcal{S}_{\text{input}}$ (Figure 2(a)). Then, we optimize the parameters of a set of randomly sampled strokes using a diffusion prior conditioned on the augmented text prompt (Figure 2(b)) and obtains an intermediate sketch.

In the second stage, the goal is to adjust the styles of the intermediate sketch to achieve a cohesive look throughout the completed sketch. We task the VLM to perform a style adjustment on the intermediate sketch. The VLM begins with identifying the style differences between strokes in the input sketch and the optimized strokes. The VLM then generates an adjustment code based on the detected style differences. We then apply the adjustment code to the intermediate sketch. We instruct the VLM to focus on differences in global abstraction levels and local stroke styles. In most of the situation, the VLM effectively identifies style differences and translates them into adjustment codes. However, it occasionally overlooks some differences when creating these codes. To resolve this issue, we repeat this process until the sketches are no longer updated.

4 Stage 1: Content-centric Sketch Completion

Inspired by previous works [Vinker et al. 2023; Xing et al. 2023], 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 Augmentation

Although the conditional T2I model generates images that match the input text prompt $\mathcal{P}_{\text{input}}$, we observed that these images are often unsuitable for directly generating strokes. The main reason is that the T2I model generates images in a photorealistic style, which tends to include many blurs and lacks clear boundaries. Previous SDS-based methods [Jain et al. 2023; Qu et al. 2023] have attempted to enhance sketch generation by augmenting the input prompt with a fixed term to promote a more sketch-like style. Such augmentation works effectively because they focus on a single type of sketch style. However, since our method needs to accommodate input sketches with various styles, using a fixed augmentation to promote non-photorealistic styles, is not effective. To address this issue, we augment the input prompt $\mathcal{P}_{\text{input}}$ with some style descriptions using the VLM based on the input partial sketch (Figure 2(a)). Specifically, we render the input partial sketch $\mathcal{S}_{\text{input}}$ into a raster image and then ask the VLM to generate textual descriptions capturing specifically the global level of abstraction and local stylistic cues of the rendered image. Then, we augment these style descriptions to the input prompt to obtain the augmented prompt.

4.2 Stroke Optimization for Completion

Using the augmented prompt, we generate strokes that fill the empty regions of the input partial sketch. We define the set of n strokes to be optimized as $\mathcal{S}_{\text{opt}} = \{s_1, \ldots, s_n\}$, and each stroke is defined 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. Initially, we generate a guidance image $\mathcal{I}_{\text{guide}}$ using a conditional T2I model based on the augmented prompt and the rendered sketch image. Then, we optimize all parameters of $\mathcal{S}_{\text{opt}}$ to obtain a sketch that is consistent with the guidance image $\mathcal{I}_{\text{guide}}$ (Figure 3). For

[1] https://huggingface.co/xinsir/controlnet-scribble-sdxl-1.0

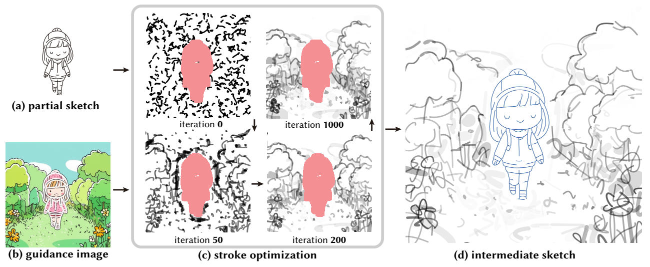
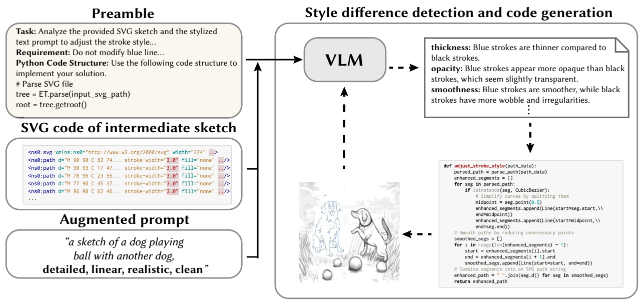


Fig. 3. **Overview of stroke optimization.** Given (a) the input partial sketch and the augmented prompt, our method generates (b) the guidance image. Then, our method (c) iteratively updates the position, opacity, and width of each stroke so the content in the guidance image is depicted faithfully. This ensures that (d) the resulting intermediate sketch aligns with the guidance image visually but does not overlap with the input partial sketch.


Preamble
Style difference detection and code generation
VLM
SVG code of intermediate sketch
Augmented prompt

Task: Analyze the provided SVG sketch and the stylized text prompt to adjust the stroke style...
Requirement: Do not modify blue line...
Python Code Structure: Use the following code structure to implement your solution.

```
# Parse SVG file
tree = ET.parse(input_svg_path)
root = tree.getroot()
...
```

```
<ns0:svg xmlns:ns0="http://www.w3.org/2000/svg" width="224" ..>
<ns0:path d="M 90 90 C 62 74... stroke-width="3.0" fill="none" ../>
<ns0:path d="M 90 93 C 17 47... stroke-width="3.0" fill="none" ../>
<ns0:path d="M 78 96 C 23 55... stroke-width="3.0" fill="none" ../>
<ns0:path d="M 77 90 C 45 37... stroke-width="3.0" fill="none" ../>
<ns0:path d="M 96 90 C 62 46... stroke-width="3.0" fill="none" ../>
...
```

"a sketch of a dog playing ball with another dog, **detailed, linear, realistic, clean***"*

thickness: Blue strokes are thinner compared to black strokes.
opacity: Blue strokes appear more opaque than black strokes, which seem slightly transparent.
smoothness: Blue strokes are smoother, while black strokes have more wobble and irregularities.
⋮

```
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
```

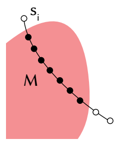
Fig. 4. **Overview of VLM-based sketch style adjustment.** The complete system prompt we provided to the VLM consists of a preamble, an augmented prompt, and the SVG code of the intermediate sketch. We input this information into the VLM, which then generates detected style differences and adjustment codes to iteratively modify the styles of the sketch.

initialization, we randomly place all strokes on the canvas. 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:

$$L_{\text{all}} = \alpha(1 - \text{sim}(\phi_{\text{vis}}(\mathcal{I}_{\text{sketch}}), \phi_{\text{vis}}(\mathcal{I}_{\text{guide}}))) + \beta(LPIPS(\mathcal{I}_{\text{sketch}}, \mathcal{I}_{\text{guide}})) + \gamma \sum_{x_k \in \mathbf{x}} \mathbb{1}\,[\mathbf{M}(x_k) = 1] \quad (2)$$

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 input partial sketch $\mathcal{S}_{\text{input}}$, we introduce an overlap penalty loss. Specifically, we first define a binary mask $\mathbf{M}$ that encodes the regions in $\mathcal{S}_{\text{input}}$ where strokes already exist:



$$\mathbf{M}(x) = \begin{cases} 1, & \text{if pixel } x \text{ belongs to strokes in } \mathcal{S}_{\text{input}}, \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

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 2 is the indicator function.

Our objective function L_{all} is very similar to the one used in DiffSketcher [Xing et al. 2023], with the exception that we have discarded the augmentation SDS (ASDS) loss. The reason for this is that we found ASDS loss contributed very little to the optimization, as indicated by the results shown in DiffSketcher paper and their released code. Therefore, to reduce computation time and focus on achieving better sketch completion result, we opted to use the overlap penalty loss instead of the ASDS loss.

After optimizing L_{all}, we obtain the intermediate sketch by combining the optimized strokes with those of input partial sketch. The strokes in the intermediate sketch contain the overall content described in the input prompt, but the styles are not coherent yet.

5 VLM-based Sketch Style Adjustment

After optimizing the strokes in Section 4.2, we have filled in the empty areas. However, this does not guarantee that the strokes in the intermediate sketch will exhibit global stylistic coherence. The wide variety of sketch styles complicates the process of defining appropriate parameterizations that can capture all potential styles. Moreover, stroke optimization can only adjust stroke parameters continuously and cannot accommodate discrete style changes, such as stroke deletion or simplification. To address these challenges, we instruct the VLM to guide the style adjustment of the intermediate sketch. Specifically, we represent the intermediate sketch in SVG format and request the VLM to modify the SVG to achieve the desired style adjustments. However, several challenges arise due to the limitations of existing VLMs. First, existing VLMs can handle only a limited number of tokens, restricting the number of strokes that can be included in the intermediate sketch. Second, these VLMs often hallucinate, i.e., they may generate strokes absent in the intermediate sketch which does not match the input prompt.

To address these issues, we propose an iterative style differences detection and code generation process (Figure 4). In each iteration, we ask the VLM to generate a list of style differences between the strokes from the previously optimized sketch with those in the input sketch. Next, we instruct the VLM to generate style adjustment codes based on the identified style differences. Then, we apply the adjustment codes to the previously optimized sketch to obtain the final sketch. This process will be terminated until sketches are no longer updated.

Style difference detection. In this step, we provide the VLM with the following information:

- A preamble that contains the instructions for the task.
- A rendered image of the intermediate sketch. In this image, the strokes of the input sketch will be rendered in blue and others will be in black.
- The intermediate sketch in SVG format.

In the preamble, we clearly outline the types of style differences we encourage the VLM to identify, including the global levels of abstraction and local stroke styles. For local stroke styles, we ask the VLM to focus on stroke thickness, smoothness, curvature, opacity. Figure 5 shows an example list of detected style differences.

thickness: Black strokes vary in thickness, with an average of about 4-5 times thicker than blue strokes, which have a consistent stroke-width of 0.5.
opacity: Black strokes have significantly lower opacity, with most near 0.05 compared to blue strokes at 1.0.

⋮

global abstraction level: Blue strokes depict more detail and crisp representation, while black strokes provide a looser, sketchier abstraction with denser detail.
other: The overall composition demonstrates that blue lines have a wider spacing between them, creating a cleaner, less cluttered appearance compared to the dense and closely packed black lines.

Fig. 5. **An example list of detected style differences.**

Adjustment code generation. In this step, we provide the VLM with the following information:

- A preamble that contains the instructions for the task.
- A list of detected style differences.
- The intermediate sketch in SVG format..
- The augmented text prompt.
- A snippet of the skeleton style adjustment code.

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 to address the differences detected in the previous step. For example, based on the detected differences in Figure 5, the VLM generates the following adjustment code:

```
def adjust_stroke_style(path_data):
  for elem in parent.findall('.//svg:path', namespace):
    style = elem.attrib
    if 'stroke' in style and style['stroke'] != 'rgb(51, 102, 178)':
      # Get current attributes
      stroke_width = float(style.get('stroke-width', 2.0))
      stroke_opacity = float(style.get('stroke-opacity', 1.0))
      # 1. Remove extreme attributes
      if stroke_opacity < 0.12:
          parent.remove(elem)
          continue
      # 2. Adjust stroke width
      style['stroke-width'] = str(max(0.5, min(stroke_width,
      given_sketch_style['stroke-width'] * 0.9)))
      # 3. Adjust opacity
      style['stroke-opacity'] = str(min(max(stroke_opacity, 0.95), 1.0))
      # 4. Adjust path to increase distance (shift position slightly)
      d = style.get('d', '')
      new_d = re.sub(
          r"([MLC])\s*(-?\d+\.?\d*)\s*(-?\d+\.?\d*)",
          lambda match: f"{match.group(1)}
          {float(match.group(2)) + 2} {float(match.group(3)) + 2}", d)
      style['d'] = new_d
```

Please see supplement for the details of the preamble we provided to the VLM and other adjustment codes generated by the VLM. We show an example of iterative style adjustment in Figure 6.

6 Experiment

6.1 Implementation Details

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

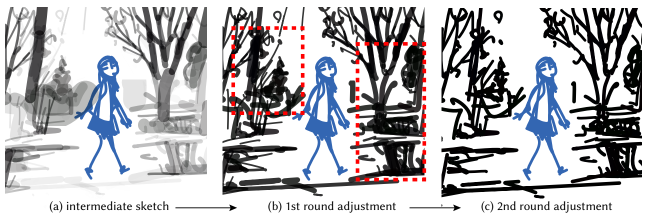


Fig. 6. **The iterative style adjustment process.** Some style differences in the (a) intermediate sketch, like stroke width and opacity in red areas, cannot be fully adjusted after (b) one iteration. These differences are addressed during (c) the second round of adjustment.

PyTorch [Paszke et al. 2019] and use the Adam [Kingma and Ba 2015] optimizer to optimize the strokes. We use 512 strokes for stroke optimization for all cases. During the style adjustment stage, some strokes are removed based on detected style differences, resulting in completed sketches that vary in density–some appearing more sparse while others are denser. In terms of computation time, it takes approximately 5 minutes to optimize 1000 iterations, while the second stage requires around 3 minutes. For all computations, we used a PC with an Intel i7CPU and an NVIDIA RTX 4080 GPU.

The input sketches used in our experiment come from three sources: sketches generated using CLIPasso [Vinker et al. 2022], trace sketches found public website, and selected design sketches from the OpenSketch dataset [Gryaditskaya et al. 2019].

6.2 Comparison with Existing Methods

We qualitatively and quantitatively compare our method to SDS-based sketch generation methods, including SketchDreamer [Qu et al. 2023] and DiffSketcher [Xing et al. 2023]. For DiffSketcher, we replace the original vanilla Stable Diffusion using the ControlNet Scribble[2] used in SketchDreamer. In Figure 7, we show the results generated by our method and all compared methods using identical user-provided partial sketch and stylized prompts. The results generated by SketchDreamer and DiffSketcher often place numerous strokes onto the input partial sketches instead of using them to illustrate desired content. It reduce the amounts of strokes that can be used to depict desired content, and often results in missing content and low visual quality. We further reduce the overlapping strokes by using the input sketch mask. However, eliminating these overlaps does not enhance the content being represented. More importantly, the styles of generated sketches do not match the style of the input sketches, resulting in inconsistencies in the final sketches. In contrast, our method can generate strokes for depicting the desired content more effectively because we introduce the overlap penalty in Equation 2.

We further compared our method with Gemini Co-Drawing[3] which is built on Gemini 2 native image generation. As shown in Figure 7(e), although Gemini Co-Drawing can generate sketch that aligns with the input prompt, both the content and style of the input partial sketch are not well preserved. In contrast, our method consistently completed sketches that faithfully represent

[2]https://huggingface.co/lllyasviel/control_v11p_sd15_scribble
[3]https://huggingface.co/spaces/Trudy/gemini-codrawing

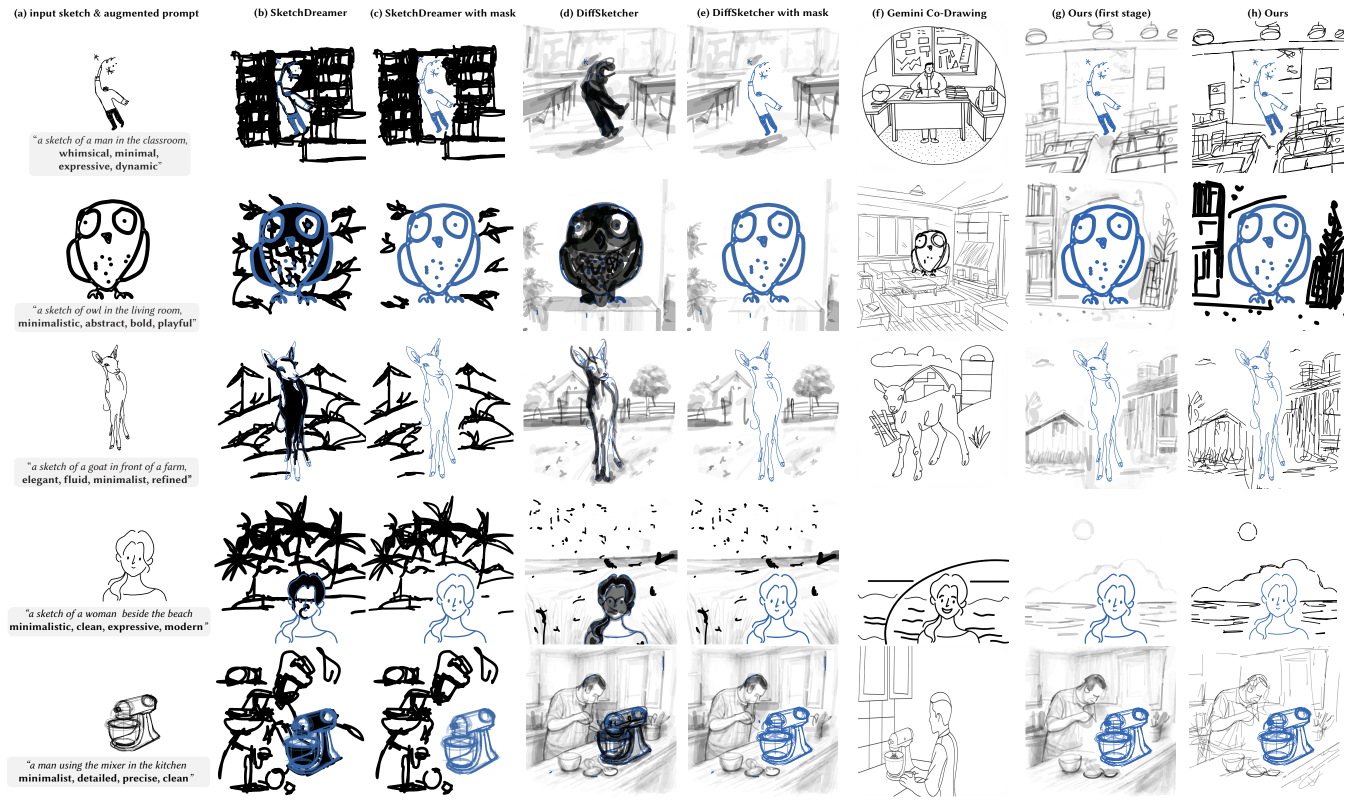

(a) input sketch & augmented prompt
(b) SketchDreamer
(c) SketchDreamer with mask
(d) DiffSketcher
(e) DiffSketcher with mask
(f) Gemini Co-Drawing
(g) Ours (first stage)
(h) Ours
"a sketch of a man in the classroom, whimsical, minimal, expressive, dynamic"
"a sketch of owl in the living room, minimalistic, abstract, bold, playful"
"a sketch of a goat in front of a farm, elegant, fluid, minimalist, refined"
"a sketch of a woman beside the beach minimalistic, clean, expressive, modern"
"a man using the mixer in the kitchen minimalist, detailed, precise, clean"

Fig. 7. **Comparison with existing methods.** Given (a) the input sketch and the augmented prompt, (b, d) the results generated by SketchDreamer and DiffSketcher wrongly place too many strokes at the input sketch region and fail to match the styles of the input sketch. (c, e) We further remove the overlapping strokes using the input sketch mask to enhance the visual quality of their results. However, these results still exhibit misaligned styles and often contains less desired content depicted in the input prompt. (e) Gemini Co-Drawing completes sketches that matches the prompt but fail to preserve the content and the style of the input partial sketch. (f) The completed sketches generated by the first stage of our method avoids to place strokes overlap with input sketch, but the styles of strokes are still inconsistent. (g) Our full method further adjusts the styles of all strokes to match the styles of the input sketches.

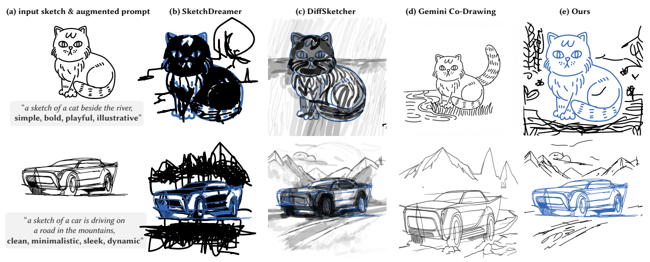

(a) input sketch & augmented prompt
(b) SketchDreamer
(c) DiffSketcher
(d) Gemini Co-Drawing
(e) Ours
"a sketch of a cat beside the river, simple, bold, playful, illustrative"
"a sketch of a car is driving on a road in the mountains, clean, minimalistic, sleek, dynamic"

Fig. 8. **More comparison results.** Given (a) the input sketch and the augmented prompt, (b,c) the results generated by SketchDreamer and DiffSketcher wrongly place too many strokes at the input sketch region and therefore alter the input sketch style. (d) Gemini Co-Drawing completes sketches that matches the prompt but fail to preserve the content and the style of the input partial sketch. (e) Our full method further adjusts the styles of all strokes to match the styles of the input sketches.

the contents of the input text prompts with consistent styles. We show additional comparison results in Figure 8.

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.

Quantitative Evaluation using existing metrics. We used DreamSim [Fu et al. 2023] and DINO [Caron et al. 2021] as the visual metrics to measure the style consistencies and image similarities between the input partial sketches and the generated completed sketches. Meanwhile, we assess the alignment between the content of each completed sketch and the input prompt using the VQA score [Lin et al. 2024]. 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. It is important to note that the visual metrics we used, including DreamSim and DINO, measure both style and content similarity rather than focusing solely on style. As a result, these metrics favor

	Visual		**Text**
	DreamSim↓	DINO↑	VQA score ↑
SketchDreamer	0.471	0.556	0.554
DiffSketcher	0.465	0.525	0.816
Gemini Co-Drawing	0.365	0.584	0.804
Our method	0.290	0.591	0.788
Our first stage	0.434	0.488	0.332
Our + Qwen3	0.305	0.578	0.781

Table 1. **Quantitative evaluation results.** We compare our method to two SDS-based methods [Qu et al. 2023; Xing et al. 2023] and Gemini Co-Drawing employing metrics that focus on visual and textual similarities. Our method consistently outperforms the other methods for visual metrics and achieves comparable performance with other methods on textual metric. We highlight the first and second best results.

images that closely replicate the input sketch and leave the rest blank, which goes against our goal of completing the sketch based on the input prompt. Therefore, it is essential to assess the results using both visual and text metrics.

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. We find that both the visual and text metrics alone used in this evaluation do not accurately reflect the preservation of the content and style from the input partial sketch, nor do they assess the quality of the completed sketch. Regarding the visual metric, Gemini Co-Drawing achieves a high DINO score, but the input partial sketches are significantly altered. On the text metric side, although both DiffSketcher and Gemini Co-Drawing receive higher scores, the completed sketch either lack quality or do not meet the preservation requirement necessary for our task.

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, SketchDreamer and DiffSketcher. We chose to conduct user evaluations on these two methods without Gemini Co-Drawing because it often alters the input partial sketch significantly and only generates raster output. Participants evaluated the quality of the generated completed sketches by conducting pairwise comparisons. For each input sketch and prompt, we created three comparative pairs, "Ours vs. SketchDreamer" and "Ours vs. DiffSketcher", resulting in 20 pairs for comparison. During each comparison, two completed sketches were shown side by side in random order, along 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.

To further evaluate AutoSketch, we recruited two amateur participants to manually complete two sketches and the results are shown in Figure 9. Participants were allowed to search for reference images online while completing the sketches. We then conducted a user evaluation using these examples and show the results in Table 2(c). The user evaluation results indicate that our method obtained higher ratings for style preservation and content depiction compared to human participants. Additionally, the participants noted that sketching multiple subjects interacting with one another while aligning the styles with the input sketch posed a particular challenge. Therefore, they believe our method would be very helpful to allow users to concentrate more on sketching the content they can create.

	Style			**Content**		
	Ours	Others	neither	Ours	Others	neither
(a) vs. SketchDreamer	94.29	5.36	0.36	86.43	3.21	10.36
(b) vs. DiffSketcher	92.14	7.50	0.36	55.71	40.36	3.93
(c) vs. manual completion	76.79	23.21	0.0	50.00	44.64	5.36

Table 2. **User evaluation results.** Compared to the two SDS-based methods and manual completion, 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 the comparing methods.) We highlight the best result.

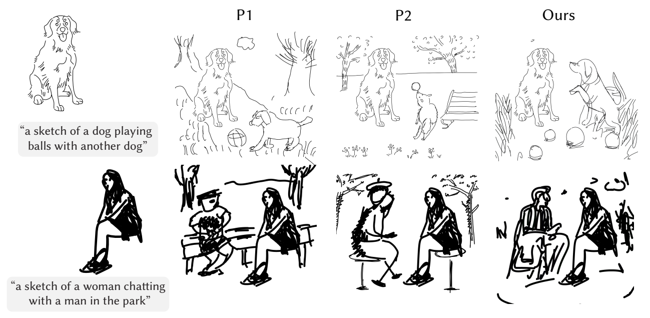

P1
P2
Ours
"a sketch of a dog playing balls with another dog"
"a sketch of a woman chatting with a man in the park"

Fig. 9. **Comparison with manual sketch completion.** We recruited two amateurs to manually complete the input partial sketch based on the prompt. Our method represents the subjects in the completed sketch more accurately and in a more cohesive style.

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 from the completed sketch and incorporating new ones (Figure 1(b) and Figure 10(b)), or by updating the input prompt (Figure 10(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 11(a), the completed sketches represent similar content but in different styles. Additionally, as shown in Figure 11(b), the completed sketches created using different input prompts can represent distinct contents but share a similar style.

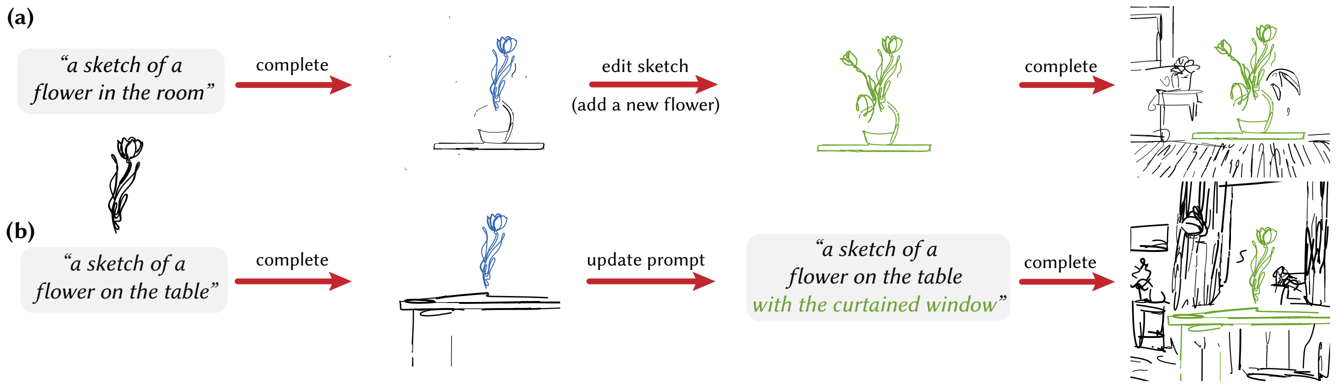

(a)
"a sketch of a flower in the room"
complete
edit sketch
(add a new flower)
complete
(b)
"a sketch of a flower on the table"
complete
update prompt
"a sketch of a flower on the table with the curtained window"
complete

Fig. 10. **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.)

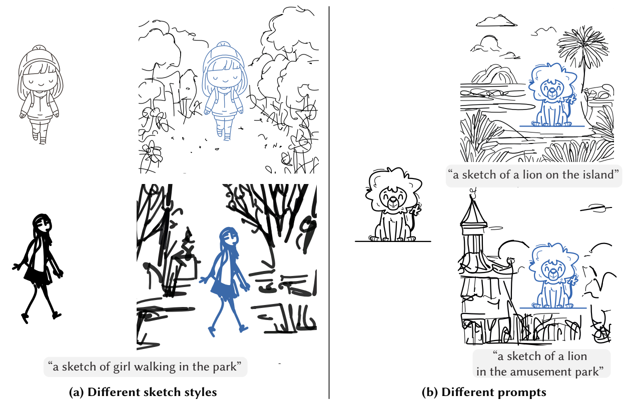

"a sketch of girl walking in the park"
(a) Different sketch styles
"a sketch of a lion on the island"
"a sketch of a lion in the amusement park"
(b) Different prompts

Fig. 11. **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.

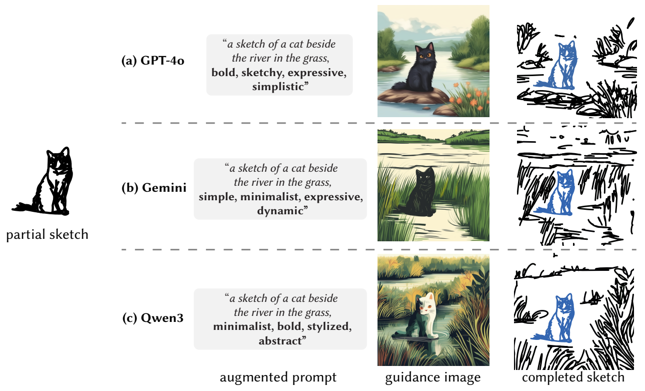

(a) GPT-4o
"a sketch of a cat beside the river in the grass, bold, sketchy, expressive, simplistic"
(b) Gemini
"a sketch of a cat beside the river in the grass, simple, minimalist, expressive, dynamic"
(c) Qwen3
"a sketch of a cat beside the river in the grass, minimalist, bold, stylized, abstract"
partial sketch
augmented prompt
guidance image
completed sketch

Fig. 12. **VLM generalization example.** Our method utilizing (a) GPT-4o, (b) Gemini, and (c) Qwen3 as the VLM can generate completed sketches that exhibit similar content and style based on the input partial sketch.

6.4 Ablation Study

6.4.1 The effectiveness of the style adjustment stage We compared the results generated by only the first stage to those of our full method. As shown in Figure 7(f), while the results of the first stage are both visually appealing and adequately represent the content of the input prompt, the sketch styles do not align well with those of the user-provided partial sketches. In contrast, as shown in Figure 7(g) and Table 1, our full method completes sketches that are better aligned with those of the user-provided sketch.

6.4.2 Generalization of VLMs Our method can utilize different VLMs to augment the input prompt and adjust style. In Figure 12, we show completed sketches using Gemini VLM [Team et al. 2023] and a open-sourced VLM Qwen3 [Yang et al. 2025]. Also, we present the quantitative results of using Qwen3 in Table 1. These results demonstrate that our method, utilizing different VLMs, can achieve results comparable to our original method using GPT-4o.

6.4.3 The effectiveness of adaptive prompt augmentation Our method enhances the input prompt by incorporating VLM-generated style descriptions of the input partial sketch, guiding the optimization of the intermediate sketch. To assess its effectiveness, we compared results generated using the input prompt, our proposed adaptive augmented prompt, the input prompt, and a fixed augmented prompt that appended "in non-photorealistic styles" to each input prompt. Figure 13 shows that our adaptive augmented prompt generates outputs that better reflect the desired content and align with input sketch style. Additionally, the quantitative results in Table 3(a,b) support these findings. We only report the text metric (VQA score), as the visual metrics tend to favor results that include only the input sketch with blank or minimal strokes, due to the content-style entanglement issue discussed in Section 6.2.

6.4.4 The effectiveness of generating style adjustment code We found that requesting the VLM to generate style adjustment code leads

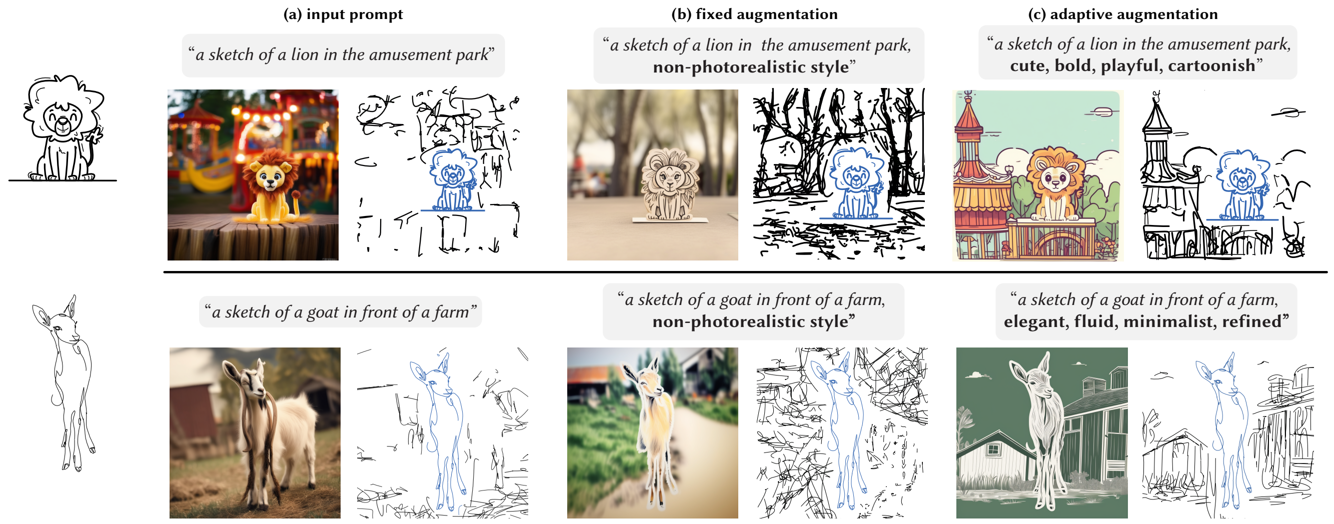

(a) input prompt
"a sketch of a lion in the amusement park"
(b) fixed augmentation
"a sketch of a lion in the amusement park, non-photorealistic style"
(c) adaptive augmentation
"a sketch of a lion in the amusement park, cute, bold, playful, cartoonish"
"a sketch of a goat in front of a farm"
"a sketch of a goat in front of a farm, non-photorealistic style"
"a sketch of a goat in front of a farm, elegant, fluid, minimalist, refined"

Fig. 13. **Prompt augmentation ablation study examples.** (a,b) The guidance image generated using the partial sketch and the original input prompt or prompt with fixed augmentation contain unsuitable blurs and lack of clear boundaries. Thus, our method could not then generate a final completed sketch that accurately depicted the complete content and the desired style. (b) In contrast, the prompt with adaptive augmentation created a guidance image that allows our method to create a more completed sketch. (The **bold text** in the prompt are style descriptions generated by the VLM.)

	(a) w/o aug	(b) fixed aug	(c) VLM edit	(d) Ours
VQA score ↑	0.443	0.532	0.331	0.788

Table 3. **Quantitative results of ablation study.** Our method completes sketches with higher alignment with the input prompt compared to other alternatives. We highlight the best result for each metric.

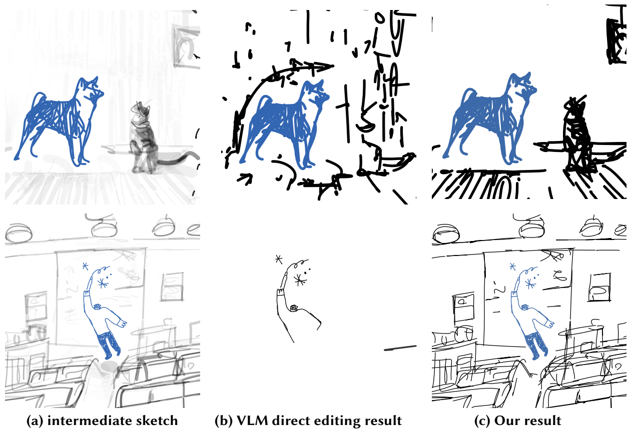


Fig. 14. **Style adjustment code ablation study examples.** Many important strokes depicted in (a) the intermediate sketch are missing from (b) the VLM direct editing results. In contrast, (c) our method effectively preserves such strokes while adjusting the styles.

to sketches that are more consistent in styles and contain more complete content than directly editing the SVG code. As shown in Figure 14, some strokes may disappear if we request the VLM to adjust styles directly. In contrast, the style adjustment code preserves the content when adjusting the styles to match those of the user-provided sketch. Similar quantitative result is observed in Table 3(c). We also only report the results of VQA score because of the same reason as Section 6.4.3.

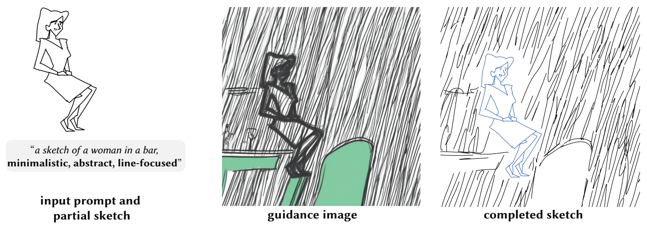

"a sketch of a woman in a bar, minimalistic, abstract, line-focused"
input prompt and partial sketch
guidance image
completed sketch

Fig. 15. **Limitation.** Our method cannot complete a sketch that accurately depicts the content of the input prompt and maintains the styles of the partial sketch with a broken guidance image generated by the ControlNet.

7 Limitations and Future Work

Reliance on large pretrained models. Our method relies on two pretrained models: ControlNet for generating the guidance images and a VLM for augmenting the input prompt and create style adjustment codes. Consequently, these models may occasionally generate unsatisfactory results. As shown in Figure 15, if the guidance image lacks the specified content, our method cannot generate strokes that depict the desired content. We will explore finetuning both ControlNet and the VLM to increase the robustness of our method.

Non-interactive generation. Our current method cannot generate completed sketch in real-time, which limits its application for interactive sketch completion. We will explore possible speedup options, such as reducing optimization steps, lowering the number of stroke samples to minimize inside/outside checks (Equation 3), and running the VLM locally.

Does not support texture styles. Texture is a popular artistic styles, but our current adjustment method struggles to distinguish between strokes for geometry and those for texture. We plan to classify these strokes and apply different style adjustment methods.

8 Conclusion

In this paper, we introduce AutoSketch, a style-aware vector sketch completion method that accommodates diverse sketch styles by leveraging a pretrained VLM. Our method allows users to input a sketch and automatically completes any missing content based on the specified prompt in a coherent style. Additionally, users can adjust strokes in the input sketch, and our method will adjust the styles of the remaining strokes to match the adjustments. We demonstrate that the style descriptions extracted by the VLM from the input sketch enable our method to accurately complete the sketch regarding the desired content. Furthermore, the style differences identified by the VLM between strokes enable our method to adjust the styles of strokes and achieve cohesive style. Extensive experiment results indicate our method is effective across various sketch scenarios.

Acknowledgments

We thank Chi-Lan Yang for brainstorming the idea of this project, Zhongyi Zhou for sharing the experiences with the use of LLM, and the anonymous reviewers for their valuable feedback. This work was supported in part by the JSPS Grant-in-Aid JP23K16921, Japan, the National Science and Technology Council (NSTC), Taiwan (under NTSC 114-2221-E-002-218-MY3, 113-2634-F-002-007, and 114-2218-E-002-006) and the Center of Data Intelligence: Technologies, Applications, and Systems at National Taiwan University (NTU) (114L900902) funded through the Featured Areas Research Center Program and NTU Core Consortium Project (114L8922), under the Higher Education Sprout Project by the Ministry of Education (MOE), Taiwan.

References

Itamar Berger, Ariel Shamir, Moshe Mahler, Elizabeth Carter, and Jessica Hodgins. 2013. Style and abstraction in portrait sketching. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

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.

Kevin Frans, Lisa Soros, and Olaf Witkowski. 2022. CLIPDraw: Exploring Text-to-drawing Synthesis Through language-Image Encoders. *Advances in Neural Information Processing Systems* 35 (2022), 5207–5218.

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. 2018. A Neural Representation of Sketch Drawings. In *Proc. ICLR*.

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).

Ajay Jain, Amber Xie, and Pieter Abbeel. 2023. VectorFusion: Text-to-SVG by Abstracting Pixel-Based Diffusion Models. In *Proc. CVPR*. 1911–1920.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*. http://arxiv.org/abs/1412.6980

Honghua Li, Hao Zhang, Yanzhen Wang, Junjie Cao, Ariel Shamir, and Daniel Cohen-Or. 2013. Curve Style Analysis in a Set of Shapes. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 77–88.

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 *Proc. CVPR*. 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).

Haoran Mo, Xusheng Lin, Chengying Gao, and Ruomei Wang. 2024. Text-based Vector Sketch Editing with Image Editing Diffusion Prior. In *Proc. ICME*. IEEE, 1–6.

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

Zhiyu Qu, Tao Xiang, and Yi-Zhe Song. 2023. SketchDreamer: Interactive Text-Augmented Creative Sketch Ideation. In *Proc. BMVC*.

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, Abhay Puri, Shubham Agarwal, Issam H Laradji, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2025. StarVector: Generating Scalable Vector Graphics Code from Images and Text. In *Proc. AAAI*, Vol. 39. 29691–29693.

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).

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

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. doi: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, and Jing Liao. 2024. Chat2SVG: Vector Graphics Generation with Large Language Models and Image Diffusion Models. *arXiv preprint arXiv:2411.16602* (2024).

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In *Proc. ICCV*. 3836–3847.

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).