Teaching an Agent to Sketch One Part at a Time

Xiaodan Du*[1], Ruize Xu*[2], David Yunis*[1],
Yael Vinker[3], and Greg Shakhnarovich[1]

[1] TTI-Chicago, Chicago, IL, USA
{xdu,dyunis,greg}@ttic.edu
[2] University of Chicago, Chicago, IL, USA
richard1xur@uchicago.edu
[3] MIT CSAIL, Cambridge, MA, USA
yaelvink@mit.edu

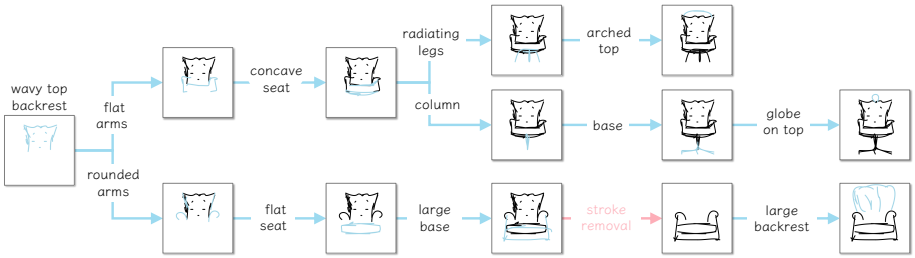

Fig. 1: Progressive vector sketch generation using our VLM agent. Trained on our new dataset via SFT + RL training, our agent generates sketches part-by-part, conditioned on text and the evolving canvas. It produces diverse, structurally plausible sketches and supports localized editing via arbitrary stroke removal and replacement.

Abstract. We develop a method for producing vector sketches one part at a time. To do this, we train a multi-modal language model-based agent using a novel multi-turn process-reward reinforcement learning following supervised fine-tuning. Our approach is enabled by a new dataset we call ControlSketch-Part, containing rich part-level annotations for sketches, obtained using a novel, generic automatic annotation pipeline that segments vector sketches into semantic parts and assigns paths to parts with a structured multi-stage labeling process. Our results indicate that incorporating structured part-level data and providing agent with the visual feedback through the process enables interpretable, controllable, and locally editable text-to-vector sketch generation. Project page: https://xiaodan.io/teaching-an-agent-to-sketch/

Keywords: sketch synthesis · VLM · reinforcement learning

* Equal contribution.

1 Introduction

Sketching provides a structured abstraction of visual content and enables rapid ideation and concept exploration in domains from industrial design to digital art. Sketches represented as vector graphics offer many advantages over rasterized canvases, making them useful in creative workflows: infinite scalability, structured visual elements, support for precise and localized modifications, and more. Automatic text-to-vector sketch generation has been widely explored [6–10, 13, 25, 36, 39, 42]. However, the majority of existing works generate the full sketch at once, overlooking the progressive, step-by-step nature of the sketching process.

Having the strokes in a sketch grouped into meaningful parts makes the sketch more easily **editable**: parts could be removed, replaced or modified in isolation from the rest of the sketch more efficiently than by modifying individual strokes. It also makes the sketch more **interpretable** to a human user. Finally, one-shot generation from long, compositional prompts may lead to failures that are localized but difficult to mitigate. In contrast, incorporating the notion of parts into the generation pipeline provides the designer part-level **control**: if a generated sketch of a part is not right, it can be replaced, and multiple choices can be explored at any intermediate stage before proceeding to other parts.

Most prior work on text-to-vector sketch generation does not allow for part-by-part generation. The one exception is SketchAgent [36] which relies on closed-source vision-language model (VLM) as backend (and thus is not easily adaptable to desired domain or style) and produces simplistic, icon-style outputs.

This leaves a gap: existing methods are unable to achieve free-text guided part-by-part generation and highly detailed vector sketch with a unified model, struggling to support human-friendly workflows which include branching possibilities and creative exploration. In contrast, our method makes these possible, as illustrated in Fig. 1.

We believe that a necessary element for closing this gap is the right training data. Work like SketchAgent has demonstrated the potential of VLMs in iteratively generating sketches conditioned on text. However, large language models (LLMs) are known to be extremely data-hungry [16, 33] and collecting a large amount of high-quality part-annotated datasets for vector sketch created by professionals can be costly and difficult to scale. To overcome this obstacle, we propose a scalable pipeline for annotating parts in vector sketches. Our pipeline relies on a multi-stage labeling process for part decomposition and path assignment. It includes proposal, critique, and revision stages. This pipeline is generic and can be applied to any vector sketch data.

We apply the aforementioned data collection pipeline on the ControlSketch dataset [1]. We call the resulting part-annotated dataset (which we will release as one of our contributions) *ControlSketch-Part*, and use it to train a VLM on the text-guided part-by-part generation task. The training uses a two-stage supervised fine-tuning (SFT) and reinforcement learning (RL) framework, where the SFT stage teaches format and initializes the sketching policy for a single

turn, and an innovative **multi-turn process-reward GRPO** RL training stage aligns multi-turn rollouts using intermediate-state rewards [29, 30].

Our automatic data pipeline and the proposed training strategy enable free-text guided multi-turn interactive sketch generation. We show, quantitatively using automated metric and user studies, and qualitatively in the paper and the supplementary, that our results significantly improve on prior work.

In summary, our contributions are:

- A generic scalable pipeline for automated VLM-based part annotation of vector sketches, yielding a short overall caption, a set of semantic part descriptions, and a complete path-to-part assignment for any vector sketch.
- A high-quality part-annotated sketch dataset, ControlSketch-Part, and an associated new benchmark for multi-turn text-to-vector sketch generation.
- A novel multi-turn process-reward GRPO algorithm for training, enabling us to train a sketching agent with novel capabilities: multi-turn vector sketch generation and progressive editing of sketches with text guidance.

The qualitative and quantitative experiment results show the potential of our data pipeline combined with VLM in the field of text-to-vector sketch synthesis.

2 Related Works

2.1 Text-to-Vector Sketch Synthesis

Previous works on text-to-vector sketch generation fall into two main categories: learning-based approaches and test-time optimization-based approaches.

Learning-based approaches Sketch-RNN [13] is one of the first works to explore this task by learning to generate polylines autoregressively. BézierSketch [7] improves upon Sketch-RNN by replacing polylines with Bézier curves for better smoothness. SketchODE [8] further extends autoregressive stroke generation by modeling sketches as continuous-time functions via Neural ODEs. More recently, inspired by the success of score-based generative modeling [15, 31], methods including ChiroDiff [9] and StrokeFusion [42] apply diffusion models to sketch synthesis and denoise all strokes simultaneously, offering no progressive, part-level control over the generation process. These methods are conditioned on predefined discrete class/attribute labels rather than free-form natural language, greatly limiting their real-world applicability. They also fall short of producing complex, high-fidelity sketches.

Test-time optimization-based approaches These methods take a longer time to produce an output but offer greater flexibility and higher visual quality. CLIPDraw [10] pioneers text-guided vector sketch synthesis by optimizing SVG paths with a CLIP-based [26] objective. Later works utilize CLIP-based optimization for versatile image-to-sketch generation [34, 35]. DiffSketcher [39], AutoSketch [6], and SketchDreamer [25] leverage a wider range of supervision, such as LPIPS [41] loss and score distillation sampling (SDS) [24] loss, to achieve higher visual quality. However, these methods optimize all strokes jointly for a single text input, producing sketches without meaningful stroke ordering or semantic part structure.

The most directly relevant work to part-aware, text-guided sketch synthesis is SketchAgent [36], which uses a closed-source Claude Sonnet model in a zero-shot prompting framework to perform text-guided sequential sketching. SketchAgent's zero-shot nature constrains it to doodle-style outputs that cannot be adapted to higher visual fidelity or specific domains. It also exhibits low spatial grounding accuracy.

2.2 Reinforcement Learning for Large Language Models

Reinforcement learning (RL) has long provided a principled framework for optimizing sequential decision-makers in Markov decision processes (MDPs). This MDP perspective is increasingly relevant for modern LLMs, since autoregressive token generation itself can be interpreted as a sequential decision process and, more broadly, many agentic applications expose the model to multi-step environments where errors can accumulate over time. Recently, DeepSeekMath introduced Group Relative Policy Optimization (GRPO), which has been efficient and successful in various reasoning tasks [30].

Multimodal RL and dense credit assignment Extensions of GRPO training go beyond text-only reasoning. In the domain of vector graphics, for example, *Reason-SVG* [38] proposes a two-stage scheme (SFT followed by GRPO) to generate SVG sketches via a hybrid reward combining programmatic correctness and visual similarity. A complementary line, *Rendering-Aware Reinforcement Learning* [27], uses rendering feedback to compute a visual reward: the similarity between a rendered SVG output and a target image guides policy improvement through GRPO. However, these methods do not use intermediate states of the generation process, whereas we leverage intermediate state representations (i.e., partial sketch) to provide dense credit assignment.

3 Automated Part Annotation

We start with an existing dataset of sketches. Our goal is to enrich each sketch, given as a vector graphics (e.g., SVG) file, with detailed part information:

- A short caption describing the entire sketch;
- A set of part descriptions, on a semantic level related to the sketch content;
- A path-to-part assignment for each path.

3.1 Data Collection Pipeline

We design a multi-step automatic data annotation pipeline that progressively derives semantic structure from the raw SVG input. An overview of the pipeline is presented in Fig. 2.

(1) Initial part decomposition The input sketch is rendered into a raster image. Based on this rendering, a VLM proposes a semantic decomposition as a small set of parts. Each part is written as a concise textual description of a

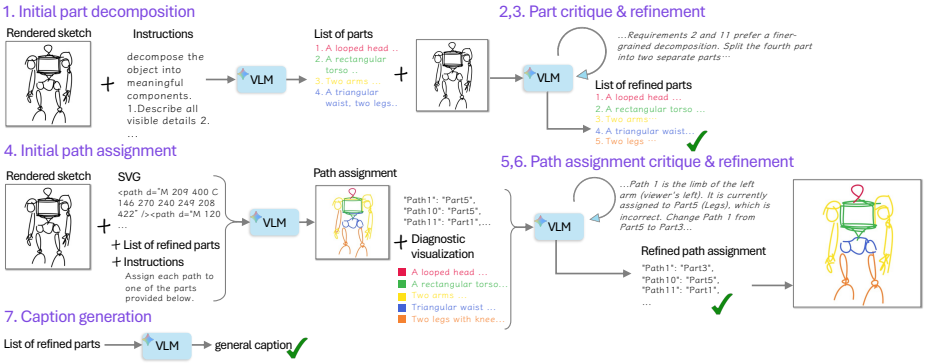

Fig. 2: An illustration of our automated part annotation pipeline. The same VLM is used to produce part designations and assignments in some stages and to critique these assignments and suggest improvements in other stages. Green check marks indicate outputs retained in the final dataset.

distinct object component. The VLM prompt (see Supplementary for all prompt details) instructs it to output non-overlapping yet collectively exhaustive parts.
(2) Part critique Like others [14, 20], we find that even current state-of-the-art VLMs struggle to follow all rules in a complicated task. Therefore, we run an improvement step: the VLM (acting now as a critic) audits the current set of parts against all the instructions from Step 1 (and the rendered sketch) and returns a structured list of issues, enforced by a schema [12]. Each issue contains "type of violation", "severity", "reasoning" and "suggested fix". The critique also contains an overall "summary" of the issues and a boolean "should revise" flag.
(3) Part refinement If the flag is "should revise", the VLM is instructed to revise the provided previous part decomposition using the critique from (2) and the sketch rendering. The output format is the same as that in (1).
(4) Initial path assignment Based on the refined parts, the sketch's SVG text, and the sketch rendering, we instruct the VLM to assign every path to one part. The output is schema-constrained so that:

- parts are assigned part labels as "Part1", "Part2",...;
- each path index ("Path1", "Path2",...) is assigned to exactly one part;
- each part contains at least one path.

(5) Path assignment critique with diagnostic visualization. We critique the path assignment similarly to (2), with the addition of a diagnostic visualization (shown in Fig. 2), as input to the VLM critic. First, we assign each part label a unique color from a pre-defined color palette, and build two panel. In the left panel of the diagnostic visualization, we render a color marker, the part label, and the part description text for each part, in the corresponding color. Thus each part description has an unambiguous visual identity. In the right panel, we recolor the sketch by rendering each path in the color of its assigned part. The two color-coded panels (descriptions and sketch) are concatenated side-by-side, making it easier for the VLM to capture the correspondence between the part de-

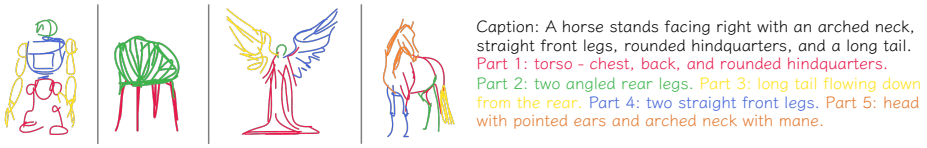

Fig. 3: Examples from the ControlSketch-Part dataset. We show part decomposition for 4 sketches with various objects and number of parts. The actual caption and part descriptions are shown for the rightmost sketch. The black text is the overall caption. The color-coded part descriptions and stroke groups demonstrate the part-level semantic annotations.

scription and the path assignment. The VLM receives the original sketch image, the diagnostic image, the previous path assignment, and the task instructions for (4), and is asked to identify incorrect path assignments and provide concrete correction suggestions. The output schema is exactly the same as that of (2).
(6) Path assignment refinement During this step, a refinement pass receives the sketch rendering, the sketch paths, the refined parts from (3), and initial part assignments from (4) along with step (4) instruction, and the path assignment critique from (5). It updates the path assignment with necessary edits under the same schema constraints as (4).
(7) Caption generation Finally, we use the VLM to generate a short general caption that summarizes the object based solely on the refined parts. This ensures the overall text caption remains consistent with part-level semantics.

3.2 Our Dataset: ControlSketch-Part

The procedure described above is designed to generalize to *any* sketch dataset with SVG (or vector-convertible) sketches. To reduce the data gap discussed in Sec. 1, we apply it to a complex, realistic-looking sketch dataset: ControlSketch.

ControlSketch is a professional-quality dataset that consists of 35,000 image-sketch pairs [1] generated by SDXL [23] and the SDS [24] loss-based optimization algorithm. It contains sketches for 15 object categories; we do not use or refer to the category labels in any way in training, and only mention them for reference when organizing examples in this paper.

We construct a schema so that the number of parts of a sketch is between 2 and 5, and apply our pipeline using Gemini 3.0 Pro as the VLM. We call the resulting dataset with the newly added captions, part descriptions, and path-to-part assignment **ControlSketch-Part**. An illustration of examples of ControlSketch-Part data can be found in Fig. 3.

4 Method

We aim to have a VLM agent generate a vector sketch iteratively: draw a part → look and reason → draw the next part. An overview of our method's pipeline

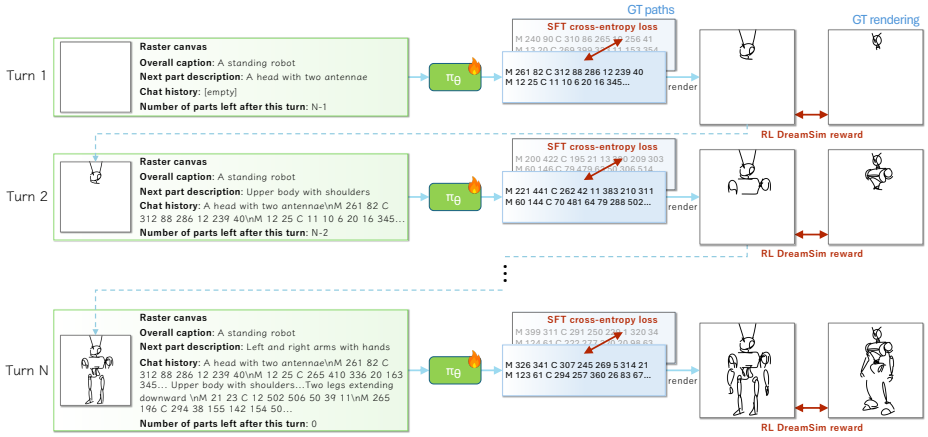

Fig. 4: The visualization of the training pipeline. The task of generating vector sketches based on text prompts is split into multiple turns. Blue arrows: sequential computation; red arrows: loss. Cross-entropy loss and DreamSim reward are used as training signal at SFT and RL stages, respectively. π_θ is the policy model, i.e., our VLM.

can be found in Fig. 4. At each turn, the VLM receives: (1) the rendering of the current canvas, (2) an overall short caption of the object it is drawing, (3) description of the next part, (4) descriptions of previously drawn parts of the sketch along with their corresponding vector paths, and (5) the number of parts left to sketch after the current turn.

The output is a sequence of paths (strokes) each coded as a curve. Since all strokes share the same set of SVG attributes (width, opacity, etc.) we instruct the model to output only the eight coordinates that define a cubic Bézier curve along with the SVG command letters `M` and `C`. The paths are separated by newline `\n`. For example, a sequence of two paths will be presented as

```
M 212 146 C 6 89 303 88 322 14\nM 213 17 C 213 269 18 157 218 32\n    (1)
```

Our method consists of two training stages: (1) supervised fine-tuning stage in which the model learns the correct output format and sketching policy for a single turn, and (2) multi-turn process-reward GRPO training to improve the visual quality of output.

4.1 Stage 1: Supervised Fine-Tuning

We conduct SFT on the VLM agent using the standard cross-entropy loss (next token prediction) on input/output pairs. We augment our dataset by randomly sampling a maximum of 20 part permutations per sketch, yielding for each permutation the corresponding sequence of part descriptions, incomplete sketches (as strokes) and the associated incomplete renderings. For instance, suppose a sketch has parts A,B,C,D and E. A permutation of these might be C,B,D,E,A.

The corresponding set of input/output pairs will include empty canvas + description of C (with the output being the ground truth strokes for C); canvas with C rendered + description of B (with the output consisting of the strokes for B); canvas with C+B rendered + description of D (with the output consisting of D), etc. See Fig. 4 for visualization. All permutation for a given sketch share the same "global" caption. This approach provides the agent with example of completing a sketch with arbitrary ordering of parts/turns.

The main purpose of the SFT stage is to train the agent to produce valid paths, and to train the model to generate a single part to extend an existing ground truth partial sketch (which will prepare it to the second stage in which it learns multi-turn generation).

4.2 Stage 2: RL with Multi-turn Process-Reward GRPO

After the SFT stage, the agent is capable of progressive generation when applied autoregressively (generate the first part, then generate the second part conditioned on observing the just generated first part, etc.) However, this creates a gap between the SFT training regime, in which the agent has only seen "oracle" intermediate states sampled from ground truth during SFT, and the inference time when it is given its own generations from previous steps. Indeed we observe a resulting deterioration in visual quality as the generation progresses.

To bridge this gap, we further train our agent with a reinforcement learning algorithm, Group Relative Policy Optimization (GRPO) [30]. GRPO computes the mean reward over multiple sampled trajectories (a group) as the baseline, replacing the need for an additional value function approximation model, which is usually of comparable size as the policy model [30]. This makes GRPO more efficient than its predecessors like [22, 29].

GRPO preliminary We call a *trajectory* a sampled sequence of responses $o^1, \ldots, o^T$ for a given input $q \sim P(Q)$. In our case, the trajectory is a sequence of sketch parts each adding to the previously generated parts. Assuming the group size (number of sampled trajectories for a given problem) is G and the length of steps for trajectory g is T_g, then the collection of all the rewards for a group $\left\{ \{o_g^t\}_{t=1}^{T_g} \right\}_{g=1}^{G}$ can be expressed as:

$$\mathbf{R} = \left\{ \{r_g^t\}_{t=1}^{T_g} \right\}_{g=1}^{G}. \tag{2}$$

The standard GRPO normalizes the rewards with the mean and standard deviation of the *entire* $\mathbf{R}$, i.e., $\tilde{r}_g^t = \left(r_g^t - \text{mean}(\mathbf{R})\right) / \text{std}(\mathbf{R})$. The advantage $\hat{A}_g^t$ of the current step is calculated as the sum of the normalized rewards from the current and the following steps,

$$\hat{A}_g^t = \sum_{t' \geq t}^{T_g} \tilde{r}_g^{t'}. \tag{3}$$

Process-reward calculation In iterative sketch generation the number of steps (parts) is fixed for a given sketch. Moreover, the ground truth of any intermediate state in a trajectory is also available to the reward model by simply assembling the ground truth paths for each previous parts together. Therefore, we can estimate intermediate rewards more precisely. Since all trajectories in a group have identical lengths (the number of parts), let us denote it by $T = T_1 = T_2 = \cdots = T_G$. Thus, the reward collection in (2) becomes

$$\mathbf{R} = \left\{ \{r_g^t\}_{t=1}^{T} \right\}_{g=1}^{G}. \tag{4}$$

Instead of estimating a unified baseline with all rewards in $\mathbf{R}$, we compute normalized rewards and advantages within each step. Let $\mathbf{R}^t = \left\{r_{g'}^t\right\}_{g=1}^{G}$, then

$$\tilde{r}_g^t = \frac{r_g^t - \text{mean}(\mathbf{R}^t)}{\text{std}(\mathbf{R}^t)}, \tag{5}$$

$$\text{and} \quad \hat{A}_g^t = \tilde{r}_g^t. \tag{6}$$

We use two rewards to supervise GRPO training: DreamSim reward intended to capture visual quality, and path count reward encouraging appropriate brevity.

DreamSim reward In each step, we render the current canvas with CairoSVG [19], a lightweight rendering engine, and measure its (image-to-image) similarity to ground truth rendering at the same step. For this we use DreamSim [11] pre-trained ensemble model to compute cosine similarity between two images in embedded space.

DreamSim is a learned perceptual similarity metric for images that aligns better with how humans judge visual similarity compared to CLIP [26], DINO [4] and LPIPS [41]. Let $\text{dreamsim}(I)$ be the embedding of an image I. The *DreamSim reward* is

$$r_{\text{dreamsim}} = \cos\left(\text{dreamsim}(I_{\text{gen}}), \text{dreamsim}(I_{\text{gt}})\right), \tag{7}$$

where I_{gen} is the current generated rendering and I_{gt} is the current ground truth rendering for the same set of parts.

Path count reward As identified by Liu et al. [21], GRPO objective induces a bias towards longer trajectories. To keep the response length close to the distribution of the training data, we introduce the *path count reward*:

$$r_{\text{pc}} = \max(0, 1 - |N_{\text{gt}} - N|/N_{\text{gt}}), \tag{8}$$

where N is the number of paths in the *final* output and N_{gt} is the number of paths in the *final* ground truth. We only regularize the agent on the final number of paths rather than the number of paths for each individual part because empirically we find per-part path count signal to be too noisy.

The combined reward is a weighted combination of the two rewards:

$$r_g^t = r_{\text{dreamsim}} + \lambda r_{\text{pc}}. \tag{9}$$

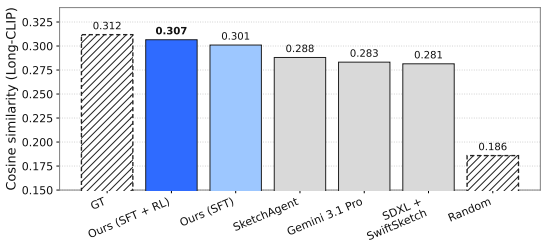

Fig. 5: The Long-CLIP cosine similarity across all tested models. The Ground Truth (GT) value and the Random value are the cosine similarity scores of text to the ground truth sketches from ControlSketch-Part and sketches of randomly sampled paths.

Before computing the rewards, we run the responses through a validity verifier. Any response that does not conform with the format will be assigned with $\min(r_g^t)$ and its trajectory will be terminated at the current step; in such a case, N and N_{gt} are the cumulative path count up to the last successful step.

Learning algorithm Our multi-turn process-reward GRPO learning objective builds on DeepSeekMath [30]. Let $\rho_{g,k}^t(\theta) = \frac{\pi_\theta(o_{g,k}^t|q,o_{g,<k}^t)}{\pi_{\theta_{old}}(o_{g,k}^t|q,o_{g,<k}^t)}$ be the token level ratios where $\pi_\theta, \pi_{\theta_{old}}$ are the current and the old policy models during policy update (our VLM agent) and q, o are questions and outputs sampled from the question dataset and the old policy $\pi_{\theta_{old}}$. k indicates the token position of the response and $o_{g,<k}^t$ indicates the conditional generation process of g-th trajectory's t-th step response conditioned on its first $< k$ tokens. The learning objective, multi-turn and thus different from [30], is

$$\begin{aligned}\mathcal{J}_{GRPO}(\theta) = \mathbb{E}\Big[q \sim P(Q), \{\{o_g^t\}_{t=1}^T\}_{g=1}^G \sim \pi_{\theta_{old}}(O|q)\Big] \\ \frac{1}{GT}\sum_{g=1}^{G}\sum_{t=1}^{T}\frac{1}{|o_g^t|}\sum_{k=1}^{|o_g^t|}\Big\{\min\Big[\rho_{g,k}^t(\theta)\,\hat{A}_{g,k}^t, \text{clip}\left(\rho_{g,k}^t(\theta), 1-\epsilon, 1+\epsilon\right)\,\hat{A}_{g,k}^t\Big] - \beta\mathbb{D}_{KL}\left[\pi_\theta||\pi_{ref}\right]\Big\},\end{aligned} \tag{10}$$

where ϵ and β are hyper-parameters, and $\hat{A}_{g,k}^t$ is the k-th token level advantage calculated above broadcasted at token position k. We estimate the KL divergence with the following unbiased estimator [28]:

$$\mathbb{D}_{KL}\left[\pi_\theta||\pi_{ref}\right] = \nu_{i,k}^t(\theta) - \log\nu_{i,k}^t(\theta) - 1, \tag{11}$$

where $\nu_{g,k}^t(\theta) = \frac{\pi_{ref}(o_{g,k}^t|q,o_{g,<k}^t)}{\pi_\theta(o_{g,k}^t|q,o_{g,<k}^t)}$, where $\pi_{\theta_{ref}}$ is the reference model. We present the pseudocode in the supplementary materials.

5 Experiments

We experimentally assess generation quality across both the step-by-step sketching procedure and the final output, comparing against state-of-the-art methods. We further validate the contribution of our multi-turn process-reward GRPO training through ablation studies. Evaluation is conducted using both automatic metrics and user studies.

5.1 Experimental Setup

Training data We follow an established practice in two-stage LLM training pipelines [5, 37] that uses separate data for SFT and RL to prevent imitation

bias, which has been found [18] to reduce exploration potential at the RL stage. We reserve the high-quality (and relatively costly to create) ControlSketch-Part dataset for RL, and prepare an alternative dataset for the SFT stage. The SFT training dataset is obtained with the same pipeline described in Sec. 3.1 but annotated with Gemini 2.5 Flash, a model 6.7× cheaper than Gemini 3.0 Pro.

Implementation details We fine-tune Qwen3-VL-30B-A3B [2] as the backbone of our sketching agent. For both stages, LoRA [17] with rank 64 is used for fine-tuning. We run SFT training with a learning rate of 2e-4 and a batch size of 128 for 5400 steps. RL training takes an additional 1000 steps with a batch size of 8, a group size of 8 and a learning rate of 3e-6. We use the reward proposed in Eq. (9) with $\lambda = 1.0$ and turn off KL divergence loss for RL training. Adam with $\beta_1 = 0.9, \beta_2 = 0.95$ and $\epsilon = 1e-8$ is used throughout the entire training. We use Thinking Machines Lab's Tinker [32] for both training stages. Bézier curve coordinates are rounded to the nearest ten for SFT training, while the original integer coordinates are retained for RL training.

Baseline methods We benchmark our method against three methods: SketchAgent [36], Gemini 3.1 Pro and SDXL [23]+SwiftSketch [1]. SketchAgent is a Claude Sonnet-based sequential sketch generation method through zero-shot prompting. The original paper uses Claude Sonnet 3.5, which is no longer available, so we switch to the more recent Claude Sonnet 4.5. We also compare against Gemini 3.1 Pro, one of the latest general-purpose VLMs at the time of writing, used as a direct whole sketch generator. SwiftSketch is an image-to-sketch diffusion model. Since it requires an image as input, we first use SDXL, a text-to-image diffusion model, to generate images from text, and then apply SwiftSketch to convert them into sketches. For methods that require a single text caption, we concatenate all part descriptions.

Evaluation metrics Both automatic metrics and user studies are used to assess the visual quality of the generated sketches. Because the lengths of the concatenated text captions often exceed the maximum input length of CLIP [26], we use Long-CLIP [40] with a maximum input length of 248 tokens, to evaluate how faithful the final sketch is to the text caption. For each sketch rendering, we compute the cosine similarity of image embedding to the embedding of the concatenated part descriptions, as a measurement of faithfulness to text input. Note that this uses different embeddings, and thus is a different metric, than DreamSim used in our reward mechanism (Eq. (7)).

We also conducted double-blind forced choice user preference studies between our method and baselines. These include two questions. In the first question, we ask the user to pick one from a pair of (whole) sketches based on the overall visual quality according to the associated text caption. In the second question, we present a looping animation showing part-by-part generation of a pair of sketches, and ask users to choose the sketch whose *generation procedure* better matches the part descriptions. We ask the first question for comparison with all methods and ask both questions for comparison with SketchAgent, the only baseline capable of part-by-part generation.

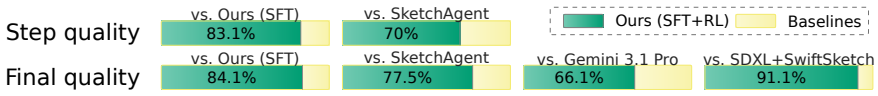

Fig. 6: Pairwise preference studies conducted between our final model (SFT + RL) and the baselines. The title for each bar describes the baseline method that we are comparing against. The first column is the ablation between our final method (SFT + RL) versus the SFT only variant, which demonstrates the effectiveness of RL.

5.2 Experiment Results and Analysis

Long-CLIP cosine similarity Fig. 5 reports the Long-CLIP cosine similarity scores of all methods and reference baselines including ground truth (GT) and randomly generated sketches (Random). The GT value is the mean Long-CLIP cosine similarity between concatenated part descriptions and the corresponding GT sketch in ControlSketch-Part, which can be viewed as the upper bound of the performance. The Random baseline is the mean Long-CLIP cosine similarity between concatenated part descriptions and sketches with randomly sampled strokes, where the number of cubic Bézier curves is sampled uniformly from [0, 32] and curve coordinates are sampled uniformly from [0, 512]. It establishes a lower bound on the metric.

Our full model (SFT + RL) achieves the best performance across all methods, surpassing the SFT-only variant, validating the contribution of both training stages. Among prior methods, SketchAgent performs best, suggesting that progressive, part-by-part generation holds a meaningful advantage over holistic approaches. Gemini 3.1 Pro, despite its general strength, falls short of specialist agents, highlighting that text-to-vector sketch generation remains a challenging domain where task-specific training still matters. SDXL + SwiftSketch trails all baselines, as errors from the text-to-image stage compound in the subsequent image-to-sketch generation. While the numerical range for all the non-random methods is fairly compressed, our user studies and qualitative results confirm that the metric differences correspond to meaningful visual quality distinctions.

User studies We conduct user studies via Prolific, an online crowdsourcing platform. For step quality preference, we collect 426 responses per baseline comparison from 142 participants, and for final output quality preference, 560 responses per baseline comparison from 146 participants, totalling 3,092 pairwise comparisons. Fig. 6 reports the percentage of comparisons in which our method was preferred. Across both evaluation settings and all baselines, participants consistently favored our results.

Qualitative results Fig. 7 shows our generated sketches alongside those of the baselines. Our sketches tend to contain smooth paths, and have a natural style with identifiable, meaningful parts. SketchAgent produces relatively clean part structure but favors simple geometric primitives and symmetric layouts, which limits visual quality. In a portion of outputs it produces misplaced or distorted components (e.g., car, dog, rabbit). Gemini 3.1 Pro shares a similar preference for simple geometries and symmetric layouts, and occasionally fails to produce a

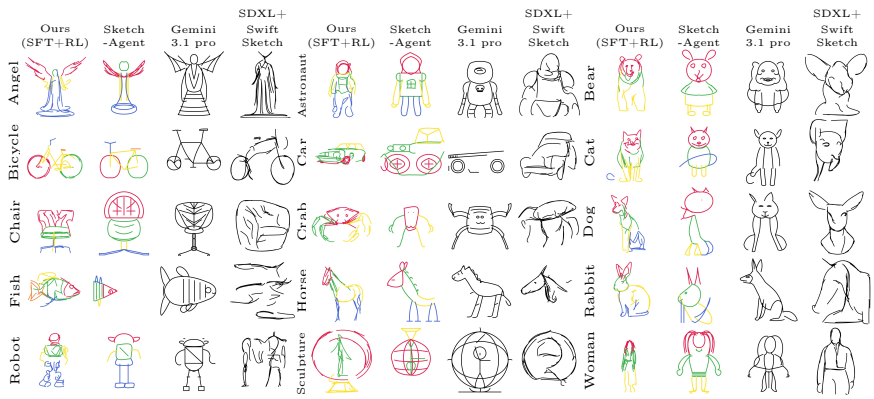

Fig. 7: Qualitative comparison. Part-by-part generated samples are color-coded to illustrate different parts. One-shot generations are rendered in black. Samples in each group are generated with the same text input. Our model and training process do not rely on the class labels in any way, and we only show these for reference.

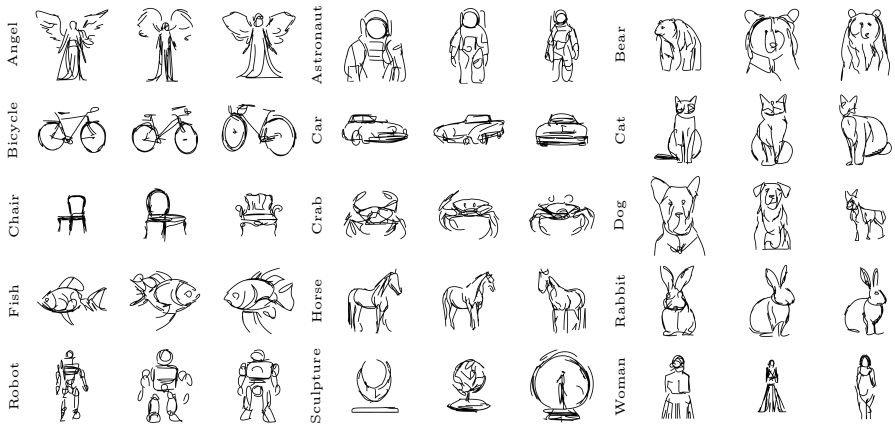

Fig. 8: Example outputs from Ours (SFT + RL) trained on ControlSketch-Part. Our model and training process do not rely on the class labels in any way, and we only show these for reference.

complete object (e.g., car). It also struggles to capture the distinguishing features of certain animals (e.g., bear, cat, and dog). SDXL + SwiftSketch can produce smooth, naturalistic sketches (e.g., bike and car), but is hindered by SDXL's lack of ability to adhere to long text inputs, which often causes it to miss details or misinterpret compositional relationships. SwiftSketch further degrades when the image generated by SDXL is low quality or lacks a clear foreground subject. Note that class labels are shown purely for reference and are not used by our model or training process in any way. We include more sketch examples of Ours(SFT + RL) in Fig. 8. More progressive editing examples can be found in Fig. 9

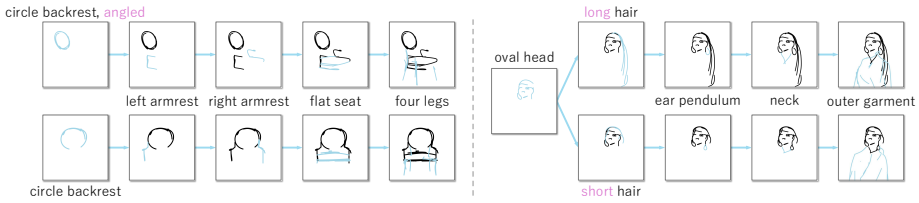

Fig. 9: Additional progressive editing examples. Left: identical part descriptions with different initial canvasses lead to different outputs. Right: changing the description for an early part but keeping the subsequent part descriptions same produces two sketches with significant differences localized to the affected part.

Table 1: Average Long-CLIP scores across different ablation configurations, using Qwen2.5-VL-3B.

Method	Long-CLIP ↑
Single-turn RL	0.281
Multi-turn outcome-reward	0.286
Multi-turn process-reward (ours)	**0.298**

5.3 Ablation Study

The performance gap (Fig. 5, Fig. 6) between Ours (SFT + RL) and Ours (SFT) shows the benefits of RL training. We further investigate the usefulness of certain training strategies of our RL training. We ablate our multi-turn process-reward GRPO formulation against a single-turn GRPO baseline and a multi-turn outcome-reward GRPO baseline. The single-turn baseline treats the entire sketching process as a single completion with only a terminal reward on the final rendering. The multi-turn outcome-reward GRPO baseline uses the reward of the final rendering to compute advantages for all the steps, whereas our multi-turn process-reward GRPO setup uses intermediate-state rewards at each step, enabling dense credit assignment over the evolving canvas. We run the controlled ablation on Qwen2.5-VL-3B [3], and observed that multi-turn GRPO setups outperform single-turn GRPO while process-reward outperforms outcome-reward, as shown in Table 1. The ablation study suggests that both multi-turn formulation and dense process-level rewards are important contributors to our model's final performance, each providing complementary benefits.

5.4 Limitations

Detailed discussion of our method's limitations, failure modes and avenues for future work can be found in the supplementary material. Here we provide a brief summary. First, data is a significant bottleneck toward a more general sketching agent. Our work is limited in its coverage of object categories beyond those present in the ControlSketch dataset. For unseen categories, the agent often fails to produce meaningful sketches altogether. When the input caption and part descriptions contain semantic concepts closely related to those seen during training, the agent tends to output sketches of "nearest-neighbor" categories rather than

the intended objects. Beyond object-level generalization, the agent also struggles to generalize to out-of-domain *parts*. A higher-level limitation stems from the inherent difficulty of large language models in predicting precise numerical coordinates, which occasionally yields sketches with drastically degraded stroke placement. In practice, such degradation is infrequent and represents only edge cases. Finally, our method currently only supports *part-level* editing, and not a finer-grained *stroke-level* editing.

6 Conclusion

We argue that part-level semantic annotation is a missing but critical ingredient for learning text-to-vector sketch generation. To close this gap, we developed a scalable automatic annotation pipeline and applied it to produce ControlSketch-Part, a dataset that enriches vector sketches with semantic part decompositions, per-part text descriptions, and path-to-part assignments. With this data in hand, we trained a VLM agent using a two-stage SFT+RL framework: SFT grounds the agent in output format and initializes the sketching policy for a single turn, while a novel multi-turn process-reward GRPO stage optimizes visual quality via intermediate visual rewards, closing the distribution gap between oracle training states and free-form inference. The resulting agent can generate structured sketches one part at a time. It outperforms prior methods across automatic metrics and user studies, and naturally supports localized editing operations (such as removal and replacement of strokes) with high visual quality. We expect that ControlSketch-Part and the proposed training framework will serve as useful resources for future research on structured multi-turn processes that benefit from visual feedback.

Acknowledgments

This work was partially supported by Hyundai Motor Co/MIT Agreement dated 2/22/2023, Hasso Plattner Foundation/MIT Agreement dated 11/02/2022, and IBM/MIT Agreement No. W1771646. The sponsors had no role in the experimental design or analysis, the decision to publish, or manuscript preparation. The authors have no competing interests to report. We gratefully acknowledge the generous gift from Adobe Research that supported our research. We also thank Professor Karen Livescu for generously providing the Gemini credits that made this research possible.

References

1. Arar, E., Frenkel, Y., Cohen-Or, D., Shamir, A., Vinker, Y.: Swiftsketch: A diffusion model for image-to-vector sketch generation. In: Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers. SIGGRAPH Conference Papers '25, Association for Computing Machinery, New York, NY, USA (2025)
2. Bai, S., Cai, Y., Chen, R., Chen, K., Chen, X., Cheng, Z., Deng, L., Ding, W., Gao, C., Ge, C., et al.: Qwen3-vl technical report. arXiv preprint arXiv:2511.21631 (2025)
3. Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., Zhong, H., Zhu, Y., Yang, M., Li, Z., Wan, J., Wang, P., Ding, W., Fu, Z., Xu, Y., Ye, J., Zhang, X., Xie, T., Cheng, Z., Zhang, H., Yang, Z., Xu, H., Lin, J.: Qwen2.5-vl technical report (2025)
4. Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., Joulin, A.: Emerging properties in self-supervised vision transformers. In: Proceedings of the International Conference on Computer Vision (ICCV) (2021)
5. Chen, H., Tu, H., Wang, F., Liu, H., Tang, X., Du, X., Zhou, Y., Xie, C.: Sft or rl? an early investigation into training r1-like reasoning large vision-language models. arXiv preprint arXiv:2504.11468 (2025)
6. Chin, H.Y., Shen, I.C., Chiu, Y.T., Shamir, A., Chen, B.Y.: Autosketch: Vlm-assisted style-aware vector sketch completion. In: Proceedings of the SIGGRAPH Asia 2025 Conference Papers. pp. 1–11 (2025)
7. Das, A., Yang, Y., Hospedales, T., Xiang, T., Song, Y.Z.: Béziersketch: A generative model for scalable vector sketches. In: European conference on computer vision. pp. 632–647. Springer (2020)
8. Das, A., Yang, Y., Hospedales, T., Xiang, T., Song, Y.Z.: Sketchode: Learning neural sketch representation in continuous time. In: International Conference on Learning Representations (2021)
9. Das, A., Yang, Y., Hospedales, T., Xiang, T., Song, Y.Z.: Chirodiff: Modelling chirographic data with diffusion models. arXiv preprint arXiv:2304.03785 (2023)
10. Frans, K., Soros, L., Witkowski, O.: Clipdraw: Exploring text-to-drawing synthesis through language-image encoders. Advances in Neural Information Processing Systems **35**, 5207–5218 (2022)
11. Fu, S., Tamir, N., Sundaram, S., Chai, L., Zhang, R., Dekel, T., Isola, P.: Dreamsim: Learning new dimensions of human visual similarity using synthetic data. In: Advances in Neural Information Processing Systems. vol. 36, pp. 50742–50768 (2023)
12. Geng, S., Cooper, H., Moskal, M., Jenkins, S., Berman, J., Ranchin, N., West, R., Horvitz, E., Nori, H.: Jsonschemabench: A rigorous benchmark of structured outputs for language models. arXiv preprint arXiv:2501.10868 (2025)
13. Ha, D., Eck, D.: A neural representation of sketch drawings. In: International Conference on Learning Representations (2018)
14. Harada, K., Yamazaki, Y., Taniguchi, M., Kojima, T., Iwasawa, Y., Matsuo, Y.: Curse of instructions: Large language models cannot follow multiple instructions at once. OpenReview (2024)
15. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in neural information processing systems **33**, 6840–6851 (2020)

16. Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L.A., Welbl, J., Clark, A., et al.: Training compute-optimal large language models. In: Proceedings of the 36th International Conference on Neural Information Processing Systems. pp. 30016–30030 (2022)
17. Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al.: Lora: Low-rank adaptation of large language models. ICLR **1**(2), 3 (2022)
18. Kang, F., Kuchnik, M., Padthe, K., Vlastelica, M., Jia, R., Wu, C.J., Ardalani, N.: Quagmires in sft-rl post-training: When high sft scores mislead and what to use instead. arXiv preprint arXiv:2510.01624 (2025)
19. Kozea: Cairosvg: Convert your svg files to pdf and png (https://cairosvg.org/) (2025), https://cairosvg.org/, last accessed 27 Jun 2026
20. Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. Transactions of the association for computational linguistics **12**, 157–173 (2024)
21. Liu, Z., Chen, C., Li, W., Qi, P., Pang, T., Du, C., Lee, W.S., Lin, M.: Understanding r1-zero-like training: A critical perspective. arXiv preprint arXiv:2503.20783 (2025)
22. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback. Advances in neural information processing systems **35**, 27730–27744 (2022)
23. Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., Rombach, R.: Sdxl: Improving latent diffusion models for high-resolution image synthesis. In: The Twelfth International Conference on Learning Representations (2024)
24. Poole, B., Jain, A., Barron, J.T., Mildenhall, B.: Dreamfusion: Text-to-3d using 2d diffusion. arXiv preprint arXiv:2209.14988 (2022)
25. Qu, Z., Xiang, T., Song, Y.Z.: Sketchdreamer: Interactive text-augmented creative sketch ideation. arXiv preprint arXiv:2308.14191 (2023)
26. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al.: Learning transferable visual models from natural language supervision. In: International conference on machine learning. pp. 8748–8763. PmLR (2021)
27. Rodriguez, J., Zhang, H., Puri, A., Pramanik, R., Feizi, A., Wichmann, P., Mondal, A., Samsami, M.R., Awal, R., Taslakian, P., et al.: Rendering-aware reinforcement learning for vector graphics generation. Advances in Neural Information Processing Systems **38**, 60496–60534 (2026)
28. Schulman, J.: Approximating kl divergence (2020), http://joschu.net/blog/kl-approx.html, last accessed 27 Jun 2026
29. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
30. Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al.: Deepseekmath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300 (2024)
31. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. arXiv preprint arXiv:2011.13456 (2020)
32. Thinking Machines Lab: Tinker (2025), https://thinkingmachines.ai/tinker/, last accessed 27 Jun 2026

33. Villalobos, P., Ho, A., Sevilla, J., Besiroglu, T., Heim, L., Hobbhahn, M.: Position: Will we run out of data? limits of llm scaling based on human-generated data. In: Forty-first International Conference on Machine Learning (2024)
34. Vinker, Y., Alaluf, Y., Cohen-Or, D., Shamir, A.: Clipascene: Scene sketching with different types and levels of abstraction. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 4146–4156 (October 2023)
35. Vinker, Y., Pajouheshgar, E., Bo, J.Y., Bachmann, R.C., Bermano, A.H., Cohen-Or, D., Zamir, A., Shamir, A.: Clipasso: Semantically-aware object sketching. ACM Trans. Graph. **41**(4) (2022)
36. Vinker, Y., Shaham, T.R., Zheng, K., Zhao, A., E Fan, J., Torralba, A.: Sketchagent: Language-driven sequential sketch generation. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 23355–23368 (2025)
37. Wang, H., Unsal, M., Lin, X., Baksys, M., Liu, J., Santos, M.D., Sung, F., Vinyes, M., Ying, Z., Zhu, Z., et al.: Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. arXiv preprint arXiv:2504.11354 (2025)
38. Xing, X., Guan, Y., Zhang, J., Xu, D., Yu, Q.: Reason-svg: Hybrid reward rl for aha-moments in vector graphics generation. arXiv preprint arXiv:2505.24499 **4** (2025)
39. Xing, X., Wang, C., Zhou, H., Zhang, J., Yu, Q., Xu, D.: Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. Advances in Neural Information Processing Systems **36**, 15869–15889 (2023)
40. Zhang, B., Zhang, P., Dong, X., Zang, Y., Wang, J.: Long-clip: Unlocking the long-text capability of clip. In: European conference on computer vision. pp. 310–325. Springer (2024)
41. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR (2018)
42. Zhou, J., Zhou, Y., Yang, H., Xu, P., Huang, H.: Strokefusion: Vector sketch generation via joint stroke-udf encoding and latent sequence diffusion. arXiv preprint arXiv:2503.23752 (2025)