# Generating Sketches in a Hierarchical Auto-Regressive Process for Flexible Sketch Drawing Manipulation at Stroke-Level

**Sicong Zang*, Shuhui Gao, Zhijun Fang**

School of Information and Intelligent Science, Donghua University, Shanghai 201620, China
sczang@dhu.edu.cn, 2242994@mail.dhu.edu.cn, zjfang@dhu.edu.cn

## Abstract

Generating sketches with specific patterns as expected, i.e., manipulating sketches in a controllable way, is a popular task. Recent studies control sketch features at stroke-level by editing values of stroke embeddings as conditions. However, in order to provide generator a global view about what a sketch is going to be drawn, all these edited conditions should be collected and fed into generator simultaneously before generation starts, i.e., no further manipulation is allowed during sketch generating process. In order to realize sketch drawing manipulation more flexibly, we propose a hierarchical auto-regressive sketch generating process. Instead of generating an entire sketch at once, each stroke in a sketch is generated in a three-staged hierarchy: 1) predicting a stroke embedding to represent which stroke is going to be drawn, and 2) anchoring the predicted stroke on the canvas, and 3) translating the embedding to a sequence of drawing actions to form the full sketch. Moreover, the stroke prediction, anchoring and translation are proceeded auto-regressively, i.e., both the recently generated strokes and their positions are considered to predict the current one, guiding model to produce an appropriate stroke at a suitable position to benefit the full sketch generation. It is flexible to manipulate stroke-level sketch drawing at any time during generation by adjusting the exposed editable stroke embeddings.

**Code** — https://github.com/SCZang/Sketch-HARP
**Extended version** — https://arxiv.org/abs/2511.07889

## Introduction

Free-hand sketches are valuable tools to convey messages and express emotions. Generative models are trained to produce sketches with specific patterns as expected, i.e., manipulating sketches in a controllable way. Several applications have been introduced to verify whether a model could manipulate sketches accurately and robustly. Controllable sketch synthesis (Zang, Tu, and Xu 2021) requires generate sketches to preserve both categorical and stylistic patterns from the input sketches. Sketch healing (Su et al. 2020) targets to restore missing details from corrupted sketches. Sketch analogy (Ha and Eck 2018) aims to generate sketches with hybrid patterns collected from multiple sketches given. These applications are instance-level sketch manipulations, i.e., the adjustments are applied on the full sketch instead of partial components. As a result, it is not flexible enough to precisely control some specific local contents (e.g., adjusting the shape of a single stroke) in a reliable manner.

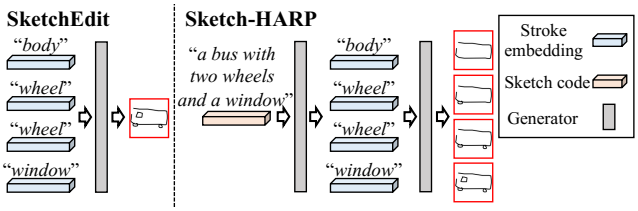


Figure 1: Manipulating sketches at stroke-level by SketchEdit (Li, Tu, and Xu 2024) and the proposed Sketch-HARP. For SketchEdit, the editable stroke embeddings are simultaneously collected and fed into the generator before generation starts, i.e., no manipulation is allowed during sketch generating process. Sketch-HARP generates sketches in a hierarchical process, i.e., with a given sketch code, a group of stroke embeddings are firstly generated, which are secondly translated into drawing actions and positioned on the canvas. The exposed stroke embeddings are editable and could be flexibly adjusted during sketch drawing to enable stroke-level sketch drawing manipulation.

Recently, SketchEdit (Li, Tu, and Xu 2024) is proposed to manipulate sketches at stroke-level. Each sketch is separated by a group of strokes with paired 2D coordinates to locate them on the canvas, and it is able to control features and locations of strokes by editing their captured embeddings' values. However, as shown in Fig. 1(left), when manipulating sketches by SketchEdit, in order to provide its generator a global view about what a sketch is going to be drawn, all the adjusted stroke embeddings should be simultaneously collected and fed into its generator before generation starts. As a result, no further manipulation is allowed during sketch generating process.

We propose a hierarchical auto-regressive sketch generating process for flexible sketch drawing manipulation at stroke-level. On one hand, our generative process is in a hierarchical manner. Instead of producing the entire sketch at

*Corresponding author.

once, we generate stroke embeddings to bridge the starting sketch code with the final generated sketch, shown in Fig. 1(right). More specifically, each stroke is generated in three stages: firstly predicting a stroke embedding to represent which stroke is going to be drawn, and secondly determining the stroke's position on the canvas, and finally translating stroke embedding to a sequence of drawing actions to form the full sketch. Such a generating process imitates human sketching procedure well: 1) realizing which sketch component is going to be drawn, and 2) considering where to settle the pen, and 3) moving the pen to draw a stroke. Moreover, as stroke embeddings are exposed during sketch generating process, we can edit them at any time during generation to realize flexible stroke-level sketch drawing manipulation.

On the other hand, the separated stroke embedding prediction, position determination and embedding translation are all auto-regressively processed. When generating a stroke, the recently produced strokes along with their positions are both considered as references, which requires model to have a global view about what the current unfinished sketch would be like and further to generate an appropriate stroke at a suitable location to benefit sketch generation.

To realize these above, we propose a method, namely sketch generation in a Hierarchical Auto-Regressive Process (Sketch-HARP) for flexible sketch drawing manipulation at stroke-level. A stroke encoder and a position encoder are employed to capture embeddings for strokes and their starting positions on the canvas, respectively, and a relationship encoder is for incorporating strokes' features before sent into a sketch encoder to obtain sketch codes. The sketch code is fed into a hierarchical auto-regressive generator to recursively produce paired stroke embeddings, starting positions, and drawing actions step-by-step in a queue. To summarize, we make the following contributions:

1. We introduce Sketch-HARP to generate sketches in a hierarchical process to enable flexible sketch drawing manipulation at stroke-level.
2. We propose an auto-regressive generator to recursively generate sketch strokes and their positions on the canvas by turns, producing appropriate strokes at suitable locations to benefit sketch generation.
3. Experimental results indicate that Sketch-HARP achieves accurate and robust performance on many sketch manipulating applications, such as stroke replacement, stroke erasion, stroke expansion, etc.

## Related Work

### Sketch Generation

Learning accurate sketch representations is a common approach to improve sketch generation. Sketches are represented by sequences of drawing actions (Ha and Eck 2018; Wang et al. 2024; Li, Tu, and Xu 2024), raster images (Chen et al. 2017; Li et al. 2024; Li, Tu, and Xu 2025), or both (Su et al. 2020; Zang and Fang 2025) for learning contextual information among strokes and visual patterns from canvas. Besides, self-organizing sketch representations into specific structures in the latent space could also regularize networks to benefit sketch learning. For example, constructing a Gaussian mixture model (GMM) distributed latent space (Zang, Tu, and Xu 2021, 2024) would encourage sketches with similar patterns to be clustered together, further contributing to accurate and robust sketch pattern representation.

Employing powerful network structures is another effective approach. The long short-term memory (LSTM) (Graves and Graves 2012) based generator proposed by (Ha and Eck 2018) is widely used to produce a sequence of drawing actions to describe sketches (Zang, Tu, and Xu 2023; Li, Tu, and Xu 2024; Su et al. 2020). Its generating process imitates how humans draw a sketch stroke-by-stroke. Diffusion model (Ho, Jain, and Abbeel 2020) based architectures are also utilized to morph and locate strokes for constructing recognizable sketches (Wang et al. 2023; Das et al. 2023).

In this paper, we propose a new sketch generating process by predicting the drawn strokes step-by-step before generating a full sketch, which benefits stroke-level sketch drawing manipulation in a flexible way.

### Sketch Manipulation

Sketch manipulation aims to generate sketches with expected patterns in a controllable way. Controllable sketch synthesis (Zang, Tu, and Xu 2021), sketch healing (Su et al. 2020), sketch reorganization (Wang et al. 2024), sketch analogy (Ha and Eck 2018; Zang, Tu, and Xu 2024) and sketch edit (Li, Tu, and Xu 2024) are sketch manipulating applications. Recent studies manipulate sketches by adjusting values of sketch codes or stroke embeddings, and the edited features are fed into generators to generate sketches as expected. Their manipulating operations (e.g., editing stroke embeddings) cannot be applied during sketch drawing.

The proposed Sketch-HARP divides sketch generating process into stroke embedding prediction, position determination and embedding translation. We are able to adjust stroke embeddings via replacement, expansion, erasion, etc. during sketch drawing, driving to flexible sketch drawing manipulation at stroke-level.

## Methodology

Fig. 2 offers the network structure of Sketch-HARP. A sketch is separated into sequence-formed strokes with their starting positions on the canvas, whose captured embeddings are incorporated to obtain a sketch code. Sketch generation is conditional on the learned code in a hierarchical auto-regressive way. The code is sent into a stroke decoder to generate a group of stroke embeddings recursively, which are fed into a sequence decoder and a position decoder to produce their corresponding positions and to translate them into drawing actions to form a full sketch, respectively.

### Translating Sketches into Paired Strokes and Starting Positions

In QuickDraw (Ha and Eck 2018), a sequence-formed sketch consists of a series of drawing actions $(x, y, l_0, l_1, l_2)$ to record how it is drawn step-by-step. $(x, y)$ is a two-dimensional coordinate to store the position of pen on the canvas, and $(l_0, l_1, l_2)$ is a one-hot vector to indicate three

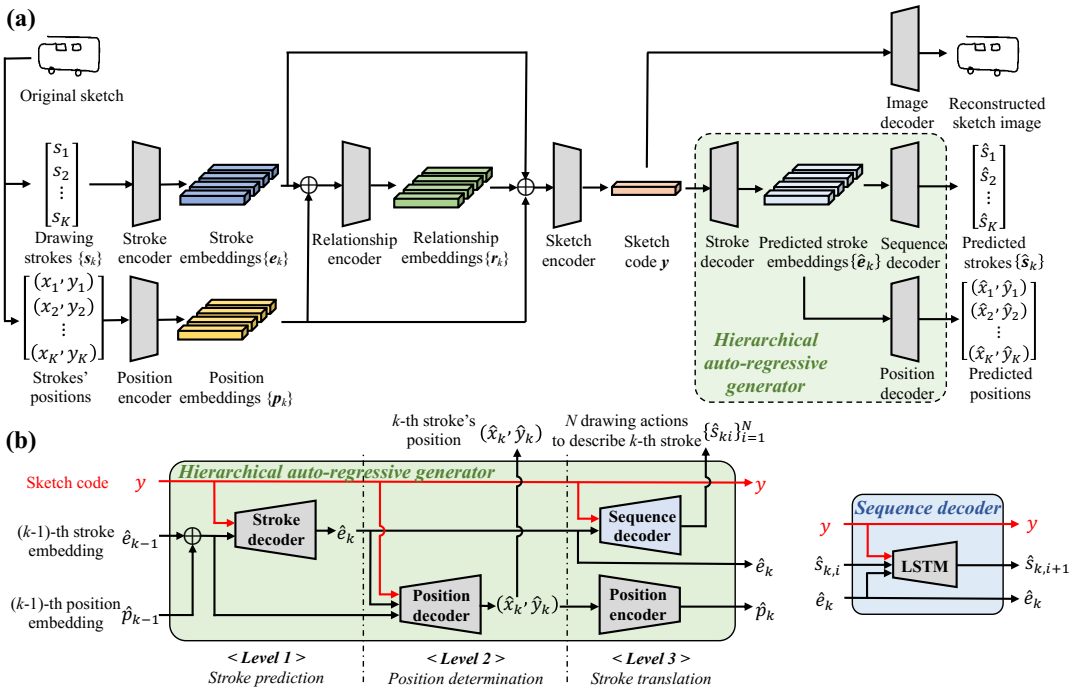


Figure 2: Generating sketches in a hierarchical auto-regressive process by Sketch-HARP. (a) The network structure. We learn relationships among strokes and incorporate them with features from strokes and starting positions to obtain a sketch code, which is fed into a hierarchical auto-regressive generator to produce sketches. (b) Generating sketches in a hierarchical auto-regressive process. Each stroke is generated by three stages: 1) predicting which stroke is going to be drawn, 2) determining where to locate the predicted stroke, and 3) translating stroke embeddings into drawing actions to finally form a full sketch.

pen states (pen down, pen lifting and end-of-drawing). As a stroke ends when $l_1 = 1$, we could split a sketch with $K$ strokes into $K$ groups of drawing actions. The $k$-th stroke $\boldsymbol{s}_k$ ($k = 1, 2, \cdots, K$) collects $j$ drawing actions $\{\boldsymbol{s}_{k1}, \boldsymbol{s}_{k2}, \cdots, \boldsymbol{s}_{kj}\}$, as it takes $j$ steps to finish its drawing. Besides, we mark the 2D coordinate of the first point of $\boldsymbol{s}_k$ as $(x_k, y_k)$, namely the starting position of stroke $\boldsymbol{s}_k$. With the cooperation between $\boldsymbol{s}_k$ and $(x_k, y_k)$, we can organize strokes on the canvas to construct a full sketch.

## Learning Sketch Representations

For a single stroke $\boldsymbol{s}_k$, we firstly capture its stroke and position embeddings to restore its trajectory of pen moving and its location on the canvas, respectively. In practice, a bidirectional LSTM-based stroke encoder $q_{\text{sok}}$ is adopted to capture stroke embedding $\boldsymbol{e}_k \in \mathbb{R}^{128\times 1}$ from $\boldsymbol{s}_k$. A fully connected layer, working as a position encoder $q_{\text{pos}}$, is employed to project 2D coordinates into position embeddings $\boldsymbol{p}_k \in \mathbb{R}^{128\times 1}$.

$$\boldsymbol{e}_k = q_{\text{sok}}(\boldsymbol{s}_k), \quad \boldsymbol{p}_k = q_{\text{pos}}((x_k, y_k)). \tag{1}$$

Such captured embeddings $\boldsymbol{e}_k$ and $\boldsymbol{p}_k$ only describe what and where a stroke is, but could hardly tell which semantic component it would be to construct a full sketch. For example, a tiny round-shaped stroke might be a wheel or a window in a bus. Actually, the semantic meaning of a stroke for constructing a sketch is determined by not only its shape, but also the relationships among strokes. For example, we might realize a tiny circle to be a wheel when we figure out that it locates beneath a large rectangle as the body of a bus. Such relationships between strokes might cover: 1) spatial relationships, indicating the relative positions of strokes on the canvas (e.g., nose is always positioned above mouth in a pig); 2) contextual relationships, reporting the temporal distances between strokes in the drawing order (e.g., two eyes of a pig are drawn closely in the drawing order); and 3) semantic relationships, revealing sketch-component-level dependencies among strokes for constructing a recognizable sketch (e.g., exact two ears are drawn to complete a pig).

Thus, it is beneficial to equip a stroke embedding with information passing from the other strokes. When realizing what other strokes are by $\{\boldsymbol{e}_k\}$ and where they are located by $\{\boldsymbol{p}_k\}$, Sketch-HARP would have a global view about how strokes are organized in a full sketch, further learning comprehensive stroke representations.

We employ a gMLP block (Liu et al. 2021) as relationship encoder $q_{\text{rel}}$ to learn relationship embedding $\boldsymbol{r}_k \in \mathbb{R}^{128\times 1}$. $\boldsymbol{r}_k$ is further incorporated to $\boldsymbol{e}_k$ to yield the final stroke representation $\tilde{\boldsymbol{e}}_k$.

$$\tilde{\boldsymbol{e}}_k = \boldsymbol{e}_k + \boldsymbol{r}_k, \quad \boldsymbol{r}_k = q_{\text{rel}}(\{\boldsymbol{e}_k + \boldsymbol{p}_k\}_{k=1}^{K}). \tag{2}$$

Finally, we employ an LSTM-based sketch encoder $q_{\text{skc}}$ to obtain the final sketch code $\boldsymbol{y} \in \mathbb{R}^{128\times 1}$.

$$\boldsymbol{y} = q_{\text{skc}}(\{\tilde{\boldsymbol{e}}_k + \boldsymbol{p}_k\}_{k=1}^{K}). \tag{3}$$

The sequential information about stroke drawing order would be restored in code $\boldsymbol{y}$.

## Generating Sketches via a Hierarchical Auto-Regressive Process

When generating sketches, a sketch code is fed into a series of decoders to produce the predicted stroke embeddings, their starting positions, and further drawing actions recursively in a hierarchical process, shown in Fig. 2(b).

**(I) Predicting stroke embeddings**

In the first stage of generation, we utilize an LSTM-based stroke decoder $p_{\text{sok}}$ to predict stroke embeddings $\{\hat{\boldsymbol{e}}_k\}_{k=1}^{K}$.

$$\hat{\boldsymbol{e}}_k, \hat{\boldsymbol{\eta}}_k = p_{\text{sok}}(\boldsymbol{y}, \tilde{\boldsymbol{e}}_{k-1} + \boldsymbol{p}_{k-1}). \tag{4}$$

We feed $p_{\text{sok}}$ with a given sketch code $\boldsymbol{y}$ and the $(k-1)$-th enriched stroke embedding $\tilde{\boldsymbol{e}}_{k-1}$ along with its position embedding $\boldsymbol{p}_{k-1}$ to predict $\hat{\boldsymbol{e}}_k$. The stroke decoder receives information about which sketch is being drawn (from $\boldsymbol{y}$), which stroke has been drawn previously (from $\tilde{\boldsymbol{e}}_{k-1}$) and where it locates (from $\boldsymbol{p}_{k-1}$), and further predicts $\hat{\boldsymbol{e}}_k$ as an appropriate stroke to be drawn currently. Note that a pair of tokens $\tilde{\boldsymbol{e}}_0$ and $\boldsymbol{p}_0$, regarded as starting markers with their values by $\tilde{\boldsymbol{e}}_0 = \boldsymbol{p}_0 = -\mathbf{1}$, are introduced to predict the first stroke embedding $\hat{\boldsymbol{e}}_1$.

Furthermore, in Eq. (4), a two-dimensional vector $\hat{\boldsymbol{\eta}}_k = [\hat{\eta}_{k0}, \hat{\eta}_{k1}]$ ($\hat{\eta}_{k0} + \hat{\eta}_{k1} = 1$) is also produced simultaneously with the prediction of $\hat{\boldsymbol{e}}_k$. $\hat{\boldsymbol{\eta}}_k$ is a marker to reveal whether the currently predicted stroke would be the last one, i.e., guiding Sketch-HARP to realize when to stop stroke production. $[\hat{\eta}_{k0}, \hat{\eta}_{k1}]$ is set as parameters of a categorical distribution to model the ground truth $\boldsymbol{\eta}_k$. $\boldsymbol{\eta}_k = [1, 0]$, if the $k$-th stroke exists, and $k = \min\{i \,|\boldsymbol{\eta}_i = [0, 1]\}$ indicates that the $k$-th stroke is the last one. Thus, we could use $\hat{\boldsymbol{\eta}}$ to determine the number of strokes in sketch generation.

**(II) Anchoring strokes on the canvas**

A group of ordered stroke embeddings $\{\hat{\boldsymbol{e}}_k\}_{k=1}^{K}$ has been predicted, but where to position them on the canvas has not been determined yet. Inspired by (Ha and Eck 2018), we model a probability distribution function (P.D.F.) to predict 2D coordinates as starting positions for each $\hat{\boldsymbol{e}}_k$. More specifically, the P.D.F. is a bi-variate Gaussian distribution $\mathcal{N}(x, y|\mu^{px}, \mu^{py}, \sigma^{px}, \sigma^{py}, \rho^{p})$, where $\mu^{px}$, $\mu^{py}$, $\sigma^{px}$, $\sigma^{py}$ and $\rho^{p}$ denote the means, the standard deviations and the correlation parameter, respectively. When anchoring a stroke on the canvas, we sample a 2D coordinate from the distribution as the predicted position. In practice, we use an LSTM-based position decoder $p_{\text{pos}}$ to predict the distribution parameters.

$$\mu_k^{px}, \mu_k^{py}, \sigma_k^{px}, \sigma_k^{py}, \rho_k^{p} = p_{\text{pos}}(\boldsymbol{y}, \hat{\boldsymbol{e}}_{k-1} + \boldsymbol{p}_{k-1}, \hat{\boldsymbol{e}}_k). \tag{5}$$

When modeling the P.D.F. of the $k$-th stroke's position, the position decoder is always fed with three inputs. The first one is a sketch code $\boldsymbol{y}$, which reminds $p_{\text{pos}}$ what a sketch is currently being drawn. The second one is $\hat{\boldsymbol{e}}_{k-1} + \boldsymbol{p}_{k-1}$, which offers the information about which stroke has already been drawn recently and where it has been located. The final input is $\hat{\boldsymbol{e}}_k$, whose position is being determined currently. Besides, when anchoring the first stroke, we also introduce a token $\hat{\boldsymbol{e}}_0 = -\mathbf{1}$ as a starting marker.

**(III) Generating sequenced-formed strokes**

We have produced a group of stroke embeddings as representations of strokes, along with their 2D coordinates on the canvas. To finish sketch drawing, the final step is translating these predicted stroke embeddings $\{\hat{\boldsymbol{e}}_k\}$ into sequence-formed strokes by drawing actions. We adopt the LSTM-based sequence decoder $p_{\text{seq}}$ from (Ha and Eck 2018) to generate a sequence of drawing actions $\hat{\boldsymbol{s}}_k = \{[\Delta\hat{x}_{ki}, \Delta\hat{y}_{ki}, \hat{\boldsymbol{l}}_{ki}]\}_{i=1}^{N_k}$ for each stroke, where $N_k$ denotes the length of the $k$-th stroke. More specifically, when generating the $i$-th drawing action for $\hat{\boldsymbol{s}}_k$, $p_{\text{seq}}$ produces: 1) a mixture of $M$ bi-variate Gaussian distributions $\sum_{m=1}^{M} \alpha_{kim} \cdot \mathcal{N}(\Delta x_{ki}, \Delta y_{ki}|\mu_{kim}^{x}, \mu_{kim}^{y}, \sigma_{kim}^{x}, \sigma_{kim}^{y}, \rho_{kim})$ with mixing weights $\{\alpha_{kim}\}_{m=1}^{M}$ to model the P.D.F. of offset distance $(\Delta x_{ki}, \Delta y_{ki})$ between the $i$-th and the $(i-1)$-th drawing actions in the $k$-th stroke, and 2) a categorical distribution with parameter $\hat{\boldsymbol{l}}_{ki}$ to predict its pen state for alerting model to stop drawing.

$$\{\boldsymbol{\alpha}_{km}, \boldsymbol{\mu}_{km}^{x}, \boldsymbol{\mu}_{km}^{y}, \boldsymbol{\sigma}_{km}^{x}, \boldsymbol{\sigma}_{km}^{y}, \boldsymbol{\rho}_{km}\}_{m=1}^{M}, \hat{\boldsymbol{l}}_k = p_{\text{seq}}(\boldsymbol{y}, \hat{\boldsymbol{e}}_k). \tag{6}$$

In addition, as the relationships among strokes have been modeled into the predicted stroke embeddings $\{\hat{\boldsymbol{e}}_k\}$ via their auto-regressive generating processes, we could decode one stroke embedding without the attendance of other strokes. Hence, it is possible to translate multiple $\hat{\boldsymbol{e}}_k$ simultaneously into drawing actions.

Note that error accumulation via the auto-regressive process by Sketch-HARP is slight, since each decoder hardly generates long sequences. We generate a sketch as: 1) producing a sequence of stroke embeddings with positions by stroke and position decoders (stroke level), and 2) translating each embedding into a sequence of drawing actions individually (drawing action level). The lengths of sequence generated by stroke/position decoders are the number of strokes per sketch, which is around 7 for sketches in Quickdraw (Ha and Eck 2018). The length for sequence decoder is the number of drawing actions per stroke, which is around 10.

## Training a Sketch-HARP

Our training objective is computed via sketch reconstruction, and it carries five terms as follows:

We adopt the sketch reconstruction loss from (Ha and Eck 2018), which measures the drawing-action-level differences between the generated sketch and the input, shown as $\mathcal{L}_{\text{seq}}$.

$$\mathcal{L}_{\text{seq}} = -\sum_{k=1}^{K}\sum_{i=1}^{N_k} \log \sum_{m=1}^{M} \alpha_{ikm} \cdot \beta_{ikm} - \sum_{k=1}^{K}\sum_{i=1}^{N_k} \boldsymbol{l}_{ki} \log \hat{\boldsymbol{l}}_{ki},$$
$$\beta_{ikm} = \mathcal{N}(\Delta x_{ki}, \Delta y_{ki}|\mu_{kim}^{x}, \mu_{kim}^{y}, \sigma_{kim}^{x}, \sigma_{kim}^{y}, \rho_{kim}), \tag{7}$$

where $N_k$ denotes the number of drawing actions in the $k$-th stroke. $(\Delta x_{ki}, \Delta y_{ki})$ is the ground truth offset distance on the canvas between the $(i-1)$-th and the $i$-th drawing actions in the $k$-th stroke. $\sum_{m=1}^{M} \alpha_{ikm} \cdot \beta_{ikm}$ is a P.D.F. modeled by sequence decoder in Eq.(6).

The first term in $\mathcal{L}_{\text{seq}}$ is a negative log-likelihood, describing a probability that the ground truth pen positions would be sampled from our modelled mixture distribution. A small value of the first term indicates that the predicted pen positions are close to the ones in ground truth. The second term is a cross entropy, measuring whether the predicted pen states $\hat{\boldsymbol{l}}$ is consistent with its corresponding ground truth $\boldsymbol{l}$.

Furthermore, we introduce $\mathcal{L}_{\text{pos}}$ to anchor the generated strokes at their corresponding 2D coordinates on the canvas.

$$\mathcal{L}_{\text{pos}} = -\sum_{k=1}^{K} \log \mathcal{N}\left(x_k, y_k | \mu_k^{px}, \mu_k^{py}, \sigma_k^{px}, \sigma_k^{py}, \rho_k^{p}\right). \quad (8)$$

Here we also adopt the measurement in $\mathcal{L}_{\text{seq}}$ to describe whether the modeled P.D.F. could predict starting positions $\{(x_k, y_k)\}_{k=1}^{K}$ accurately.

As stroke decoder generates a sketch stroke-by-stroke, it is necessary to produce a marker to report when the last stroke has been drawn. For an input sketch, we utilize a sequence of vectors $\{\boldsymbol{\eta}_k\}_{k=1}^{K}$ as markers. Each $\boldsymbol{\eta}_k$ is a two-dimensional one-hot vector. $\boldsymbol{\eta}_k = [1, 0]$ or $[0, 1]$ indicate that the $k$-th stroke exists or not, respectively. We calculate the cross entropy between ground truth $\{\boldsymbol{\eta}_k\}$ and their corresponding predictions $\{\hat{\boldsymbol{\eta}}_k\}$, ensuring no redundant strokes would be drawn on the canvas.

$$\mathcal{L}_{\text{stp}} = -\sum_{k=1}^{K} \boldsymbol{\eta}_k \log \hat{\boldsymbol{\eta}}_k. \quad (9)$$

$\mathcal{L}_{\text{seq}}$ drives the generated strokes to be consistent with the original input in drawing action level. We also introduce $\mathcal{L}_{\text{sok}}$ to assist stroke reconstruction in feature level.

$$\mathcal{L}_{\text{sok}} = \sum_{k=1}^{K} \|\hat{\boldsymbol{e}}_k - \text{sg}(\tilde{\boldsymbol{e}}_k)\|_2^2, \quad (10)$$

where $\text{sg}(\cdot)$ denotes stop gradient operation, and $\|\cdot\|_2^2$ represents L2 norm function. $\mathcal{L}_{\text{sok}}$ is a regularization term to push the generated stroke embedding $\hat{\boldsymbol{e}}_k$ towards its target $\tilde{\boldsymbol{e}}_k$.

Moreover, we adopt a convolutional neural network (CNN) decoder $p_{\text{img}}(\boldsymbol{I}|\boldsymbol{y})$ to reconstruct the image-formed sketch $\boldsymbol{I} \in \mathbb{R}^{128\times128\times1}$ by its code $\boldsymbol{y}$. The reconstructed sketch image $\hat{\boldsymbol{I}}$ is sent into a regularization loss $\mathcal{L}_{\text{img}}$ to encourage $\boldsymbol{y}$ to carry more features from the input sketch.

$$\mathcal{L}_{\text{img}} = \|\hat{\boldsymbol{I}} - \boldsymbol{I}\|_2^2. \quad (11)$$

Finally, our objective is to minimize

$$\mathcal{L} = \mathcal{L}_{\text{seq}} + \mathcal{L}_{\text{pos}} + \mathcal{L}_{\text{stp}} + 5 \cdot \mathcal{L}_{\text{sok}} + 0.5 \cdot \mathcal{L}_{\text{img}}. \quad (12)$$

We give $\mathcal{L}_{\text{sok}}$ a large weight to predict accurate stroke embeddings $\hat{\boldsymbol{e}}_k$ to further improve sketch generation.

# Experiments and Analysis

## Preparations

**Datasets**. Two datasets from QuickDraw (Ha and Eck 2018) are selected. Dataset 1 (DS1), adopted from (Li, Tu, and Xu 2024; Su et al. 2020), contains 17 categories (airplane, angel, alarm clock, apple, butterfly, belt, bus, cake, cat, clock, eye, fish, pig, sheep, spider, umbrella, the Great Wall of China). The sketches in DS1 are easily to be recognized in category. Dataset 2 (DS2), adopted from (Zang, Tu, and Xu 2021; Li, Tu, and Xu 2024), collects 5 categories (bee, bus, flower, giraffe and pig). DS2 could evaluate whether a method is powerful on learning sketch representation, as some categories contain multiple non-categorical features (e.g., giraffes orienting left or right). Each category contains $70K$ training, $2.5K$ valid and $2.5K$ test sketches ($1K = 1000$).

**Baselines**. We compare Sketch-HARP with 7 baseline models: RPCL-pix2seq (Zang, Tu, and Xu 2021), RPCL-pix2seqH (Zang, Tu, and Xu 2024), SketchHealer (Su et al. 2020), Lmser-pix2seq (Li et al. 2024), SP-gra2seq (Zang, Tu, and Xu 2023), DC-gra2seq (Zang and Fang 2025) and SketchEdit (Li, Tu, and Xu 2024). Note that SketchEdit (Li, Tu, and Xu 2024), which is designed for sketch edit, is selected to make a direct comparison with Sketch-HARP on sketch manipulation.

**Implementation details**. The LSTM-based $q_{\text{sok}}$ and $q_{\text{skc}}$ are with hidden state size as 512, and $p_{\text{sok}}$, $p_{\text{pos}}$ and $p_{\text{seq}}$ are set at 1024. $q_{\text{rel}}$ is a stack of two gMLP blocks with $d_{\text{model}} = 128$ and $d_{\text{ffn}} = 512$. And $p_{\text{img}}$ consists of a fully connected layer (with output dimension at 2048, which are reshaped to $128 \times 4 \times 4$ before fed into CNN layers) and five convolutional layers (with channel numbers as 128, 128, 128, 128 and 1, kernel size $4 \times 4$, stride 2 and padding 1) followed by batch normalization and ReLU activation function (the last activation function is Tanh). When training a Sketch-HARP, the mixture number $M$ in GMM distribution, the max number of strokes $N_{\text{max}}^{\text{num}}$, the max length of a stroke $N_{\text{max}}^{\text{len}}$ and the mini-batch size are fixed at 20, 25, 32, 128, respectively.

## Stroke-Level Sketch Manipulation

**Stroke replacement**. This application aims to replace a target stroke from a sketch with a source stroke, which could be selected from another sketch given. For example, in Fig. 3, the red stroke in "target stroke" column is replaced by the red one in "source stroke" column, and the generated sketches should preserve details from the composed sketches.

Fig. 3 shows some stroke replacement results. Sketch-HARP is powerful to maintain the features from source stroke in generated sketches, though no visual patterns are utilized in sketch code learning, compared with some baselines (e.g., Lmser-pix2seq and DC-gra2seq) whose model inputs are sketch images. Furthermore, our generated sketches are harmonious in appearance, even if the introduced source stroke and the target sketch are collected from different categories. For example, in the 2-nd row, the pair of wings of the generated angel are different in shape, but are about the same size. These experimental results demonstrate that Sketch-HARP could restore stroke embeddings into drawn strokes well, and is able to customize strokes to produce reasonable sketches.

**Stroke erasion**. This application erases stroke(s) from a sketch, and the adjusted sketch is fed into models as a condition for sketch generation. Models are required to preserve the details from the adjusted sketches.

Fig. 4 shows some stroke erasion results. Sketch-HARP could prevent drawing redundant strokes when the currently generated sketch contains enough details about the input sketch. For example, only a single leg is drawn in the sheep, though it may not be common to represent a sheep. Furthermore, Sketch-HARP could always anchor strokes at their right positions, e.g., the bus wheels are closely connected with its body, and only the leftmost candle remains on the

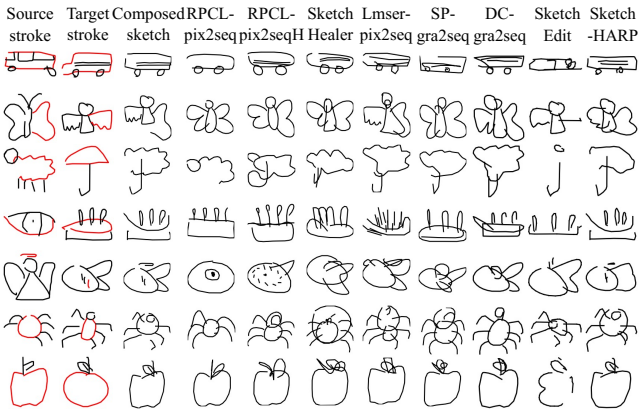


Figure 3: Comparisons on stroke replacement. For each sketch in "target stroke" column, its stroke in red is replaced by the highlighted one in "source stroke" column, with the composed sketches listed in "composed sketch" column.

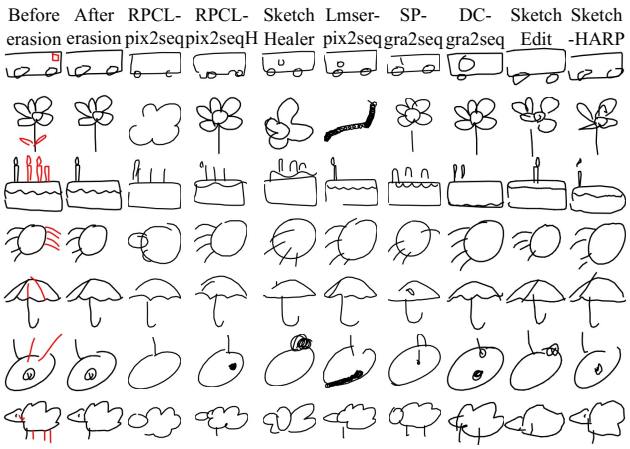


Figure 4: Comparisons on stroke erasion.

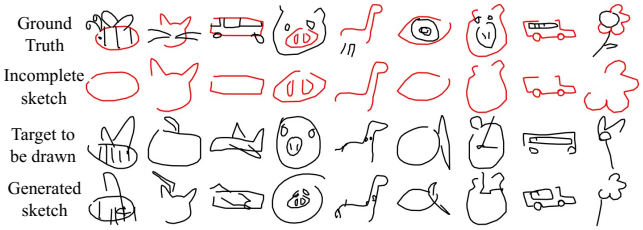


Figure 5: Exemplary stroke expansion results by Sketch-HARP. Sketch-HARP is required to continue drawing from an incomplete sketch, and the generated sketch should contain features from another source sketch.

cake. These results demonstrate that Sketch-HARP could precisely manipulate stroke drawing in determining strokes' positions and controlling an appropriate number of strokes.

**Stroke expansion**. Stroke expansion aims to insert some introduced strokes during sketch drawing. A special case of stroke expansion is that the introduced strokes are inserted at the start of drawing. For example, in Fig. 5, Sketch-HARP

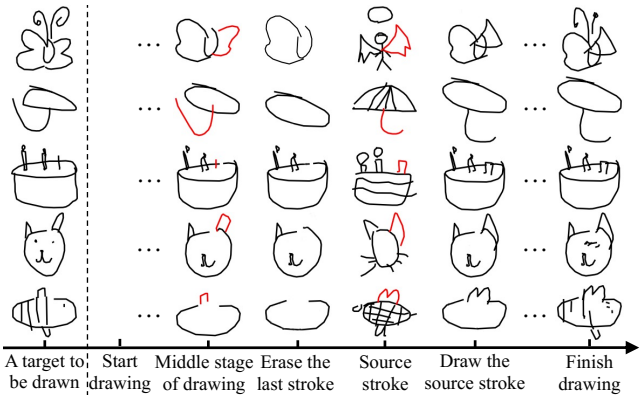


Figure 6: Manipulating sketch drawing at stroke-level by Sketch-HARP. Before applying stroke erasion, the last-drawn-stroke for each sketch is highlighted in red.

| Model | Replacement | Erasion |
|---|---|---|
| RPCL-pix2seq | 0.21±0.59 | 0.38±0.76 |
| RPCL-pix2seqH | 0.29±0.71 | 0.46±0.76 |
| SketchHealer | 0.36±0.73 | 0.53±0.94 |
| Lmser-pix2seq | 0.64±0.93 | 0.65±1.02 |
| SP-gra2seq | 0.52±0.79 | 0.40±0.78 |
| DC-gra2seq | 0.44±0.90 | 0.24±0.65 |
| SketchEdit | 1.45±1.27 | 1.04±1.14 |
| Sketch-HARP | **2.09±1.06** | **2.30±0.96** |

Table 1: Human scoring results (mean±std.) on stroke replacement and stroke erasion.

is required to continue drawing from the unfinished sketches in 2-nd row, and patterns of the generated sketches are conditional on another references (sketches in 3-rd row).

Fig. 5 reports some stroke expansion results by Sketch-HARP. Without visual features extracted, Sketch-HARP could still generate reasonable sketches to not only fit the categorical patterns from source sketches but also preserve detailed patterns from incomplete sketches. For example, in the 7-th column, Sketch-HARP could draw hands of a clock on a face contour borrowed from a pig, to create a pig-head-shaped clock. The features extracted from the drawn strokes could be referenced by Sketch-HARP to guide sketch drawing for creating unique hybrid sketches.

**Manipulating sketch drawing at stroke-level**. As Sketch-HARP generates sketches in a hierarchical process with exposed stroke embeddings, it is possible to edit stroke embeddings during sketch drawing to manipulate sketch drawing at stroke-level. More specifically, during sketch drawing, two basic operations: 1) erasing the stroke which was drawn recently, and 2) inserting a given stroke into generation before the current stroke drawing, could be applied to manipulate sketch drawing. And it is able to realize stroke replacement by applying stroke erasion and expansion in a queue.

Fig. 6 reports some results by Sketch-HARP. When a sketch is being drawn at a middle stage, it is free to erase the last-drawn-stroke (highlighted in red in the 2-nd column). Moreover, we could select a source stroke from another

| Dataset | Model | FID ↓ | LPIPS ↓ | CLIP-S ↑ | $Rec$ ↑ | $Ret$ ↑ | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | @1 | @10 | @50 |
| DS1 | RPCL-pix2seq (Zang, Tu, and Xu 2021) | 22.74 | 0.37 | 88.57 | 81.80 | 28.80 | 59.05 | 77.52 |
| | RPCL-pix2seqH (Zang, Tu, and Xu 2024) | 17.30 | 0.38 | 88.66 | 87.82 | 81.22 | 92.15 | 94.90 |
| | SketchHealer (Su et al. 2020) | 27.72 | 0.39 | 88.17 | 87.03 | 68.52 | 82.37 | 86.57 |
| | Lmser-pix2seq (Li et al. 2024) | 18.94 | 0.36 | 89.74 | **90.50** | 80.56 | 86.68 | 89.88 |
| | SP-gra2seq (Zang, Tu, and Xu 2023) | 10.80 | 0.38 | 88.72 | 89.83 | 94.05 | 98.72 | 99.57 |
| | DC-gra2seq (Zang and Fang 2025) | 12.83 | 0.30 | **93.84** | 90.41 | 96.27 | **99.47** | **99.81** |
| | Sketch-HARP | **9.96** | **0.28** | 91.60 | 89.90 | **97.96** | 99.42 | 99.77 |
| DS2 | RPCL-pix2seq (Zang, Tu, and Xu 2021) | 36.56 | 0.28 | 85.47 | 74.94 | 26.72 | 48.80 | 63.13 |
| | RPCL-pix2seqH (Zang, Tu, and Xu 2024) | 26.90 | 0.34 | 88.13 | 84.40 | 74.63 | 90.09 | 94.65 |
| | SketchHealer (Su et al. 2020) | 31.02 | 0.29 | 91.46 | 74.93 | 32.09 | 57.35 | 73.63 |
| | Lmser-pix2seq (Li et al. 2024) | 10.10 | 0.32 | 91.10 | 85.02 | 90.23 | 93.05 | 94.67 |
| | SP-gra2seq (Zang, Tu, and Xu 2023) | 34.06 | 0.45 | 85.17 | 76.89 | 56.27 | 80.26 | 90.56 |
| | DC-gra2seq (Zang and Fang 2025) | 11.01 | 0.30 | **94.46** | 85.67 | 95.27 | 99.09 | 99.65 |
| | Sketch-HARP | **6.97** | **0.22** | 93.31 | **87.71** | **96.00** | **99.78** | **99.93** |

Table 2: Sketch reconstruction performance (%). "@$k$" indicates the top-$k$ accuracy.

sketch and inject its stroke embedding into Sketch-HARP's generator to draw the source stroke as a stroke expansion. After that, Sketch-HARP is released to finish sketch drawing freely. As shown in Fig. 6, our model could preserve both features from the target sketch and from the selected stroke to yield recognizable sketches after manipulation.

**Human perception quality**. In order to investigate human subjective rating of the quality of sketch manipulation, we administer a questionnaire with 10 sketch replacement and 10 sketch erasion questions. For each question, a human subject was required to rank the manipulated sketches generated by Sketch-HARP and 7 baselines, and we only value the best three models with scores at 3, 2 and 1, respectively, leaving the rest models with 0 score. 51 volunteers participated in the evaluation, and the scoring results in Table 1 reports that Sketch-HARP outperforms all baselines. We also raise t-tests in which the p-values against our Sketch-HARP with SketchEdit are $6.04 \times 10^{-13}$ and $1.23 \times 10^{-56}$ for sketch replacement and erasion, respectively, revealing that the improvement contributed by Sketch-HARP compared with SketchEdit is significant.

## Sketch Representation and Reconstruction

We also make comparisons on sketch representation and reconstruction. If the generated sketches preserve more details from the inputs, that model is powerful in both sketch representation learning and conditional sketch generation. We utilize the Fréchet Inception Distance (FID) score (Heusel et al. 2017), the learned perceptual image patch similarity (LPIPS) computed by ControlNet (Zhang, Rao, and Agrawala 2023) and the contrastive language-image pre-training score (CLIP-S) (Hessel et al. 2021) to evaluate the generative performance on sketches. We also use $Rec$ and $Ret$ (Zang, Tu, and Xu 2021) to measure whether the generated sketch could preserve more categorical or stylistic features from the input, and we trained two sketch-a-nets (Yu et al. 2015) on DS1 and DS2, respectively, for computing $Rec$. Since SketchEdit represents a sketch by a group of stroke embeddings instead of a single sketch code, its generator receives about 10 times of information from the input sketch than all the other models. To make the comparison fair, we do not include it in this quantitative comparison.

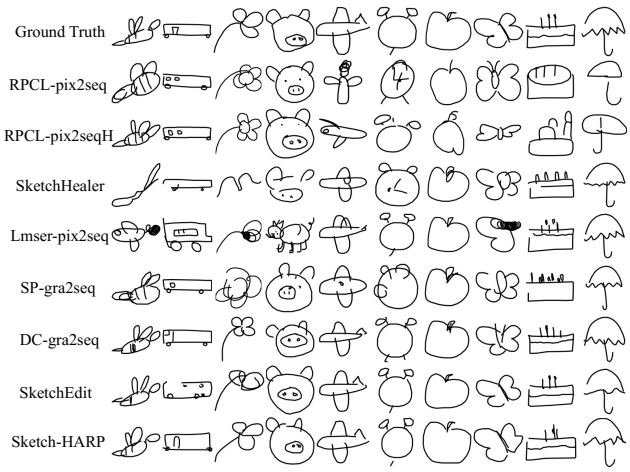


Figure 7: Qualitative comparisons on sketch reconstruction.

Table 2 reports quantitative results on two datasets. Sketch-HARP achieves comparable sketch reconstruction performance with state-of-the-art baselines. DC-gra2seq learns sketch codes from sketch images cooperated with contextual information from drawing orders. The visual patterns of sketches, which are never utilized by Sketch-HARP, improves DC-gra2seq's reconstruction performance. Fig. 7 reports qualitative comparisons on sketch reconstruction.

## Conclusions

We have presented Sketch-HARP to generate sketch sequences in a three-staged hierarchical auto-regressive generating process. By separating a stroke drawing into stroke embedding prediction, stroke position determination and drawing actions translation, the stroke embeddings, introduced as editable accesses in generating process, are exposed for manipulation. Thus, we can control strokes' features, their locations on the canvas, and the number of strokes, i.e., manipulating sketch drawing in a controllable way.

## Acknowledgments

This work was supported by the National Natural Science Foundation of China (62406064) and the Fundamental Research Funds for the Central Universities (2232024D-28).

## References

Chen, Y.; Tu, S.; Yi, Y.; and Xu, L. 2017. Sketch-pix2seq: a model to generate sketches of multiple categories. *arXiv preprint arXiv:1709.04121*.

Das, A.; Yang, Y.; Hospedales, T.; Xiang, T.; and Song, Y.-Z. 2023. Chirodiff: Modelling chirographic data with diffusion models. *arXiv preprint arXiv:2304.03785*.

Graves, A.; and Graves, A. 2012. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, 37–45.

Ha, D.; and Eck, D. 2018. A neural representation of sketch drawings. In *International Conference on Learning Representations*.

Hessel, J.; Holtzman, A.; Forbes, M.; Bras, R. L.; and Choi, Y. 2021. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*.

Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; and Hochreiter, S. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33: 6840–6851.

Li, T.; Tu, S.; and Xu, L. 2024. SketchEdit: editing freehand sketches at the stroke-level. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 4461–4469.

Li, T.; Tu, S.; and Xu, L. 2025. SketchMLP: effectively utilize rasterized images and drawing sequences for sketch recognition. *Machine Learning*, 114(3): 1–18.

Li, T.; Zang, S.; Tu, S.; and Xu, L. 2024. Lmser-pix2seq: Learning stable sketch representations for sketch healing. *Computer Vision and Image Understanding*, 240: 103931.

Liu, H.; Dai, Z.; So, D.; and Le, Q. V. 2021. Pay attention to mlps. *Advances in neural information processing systems*, 34: 9204–9215.

Su, G.; Qi, Y.; Pang, K.; Yang, J.; and Song, Y.-Z. 2020. SketchHealer: a graph-to-sequence network for recreating partial human sketches. In *Proceedings of The 31st British Machine Vision Virtual Conference*, 1–14. British Machine Vision Association.

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

Wang, X.; Li, T.; Zang, S.; Tu, S.; and Xu, L. 2024. Self-supervised learning for enhancing spatial awareness in free-hand sketch. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 5117–5125.

Yu, Q.; Yang, Y.; Song, Y.-Z.; Xiang, T.; and Hospedales, T. 2015. Sketch-a-net that beats humans. In *British Machine Vision Conference*.

Zang, S.; and Fang, Z. 2025. Equipping sketch patches with context-aware positional encoding for graphic sketch representation. *Computer Vision and Image Understanding*, 104385.

Zang, S.; Tu, S.; and Xu, L. 2021. Controllable stroke-based sketch synthesis from a self-organized latent space. *Neural Networks*, 137: 138–150.

Zang, S.; Tu, S.; and Xu, L. 2023. Linking Sketch Patches by Learning Synonymous Proximity for Graphic Sketch Representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 11096–11103.

Zang, S.; Tu, S.; and Xu, L. 2024. Self-Organizing a Latent Hierarchy of Sketch Patterns for Controllable Sketch Synthesis. *IEEE Transactions on Neural Networks and Learning Systems*, 35(10): 14506–14518.

Zhang, L.; Rao, A.; and Agrawala, M. 2023. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, 3836–3847.