Technical Assignment Report

Sketch Generation via Diffusion Models using Sequential Strokes

1 Introduction

The purpose of this report is to present the development process and findings from a project focused on sequential stroke generation. The primary objective was to implement and train diffusion models capable of producing hand-drawn sketches for three classes—cat, bus, and rabbit—in a sequential, stroke-by-stroke manner, rather than generating the entire image at once. A supplementary Jupyter notebook *(Technical_Assignment.ipynb)* contains the full technical implementation, including the model architectures, data preprocessing, and training steps.

To achieve this goal, the project began with a literature review to identify state-of-the-art techniques (Section 2). Following this, three different diffusion architectures were implemented and trained (Section 3). While the models training steps are build successfully, they encountered significant challenges common in complex generative tasks, which prevented the final visual outputs from converging. The final section of this report provides a detailed analysis of these challenges and outlines potential improvements for future work, demonstrating a thorough exploration of a research-level problem .

However, despite testing these various approaches, the model's final outputs did not reach the level of producing successful and recognizable sketches.

2 Related Works

Diffusion models are a powerful class of generative models based on the principle of iteratively denoising a random signal to produce data. This architecture quickly surpassed Generative Adversarial Networks (GANs) in image generation, particularly in training stability and sample quality. While most of the research is in the image generation field, there are some works including text and point generation.

The Quick, Draw![1] dataset contains over 50 million drawings across 345 different categories, collected from Google's AI game. Unlike typical image datasets, each sketch is saved as a series of vectors that show the pen's movement over time, rather than as pixels. This structure provides a unique resource for modeling the action of drawing itself.

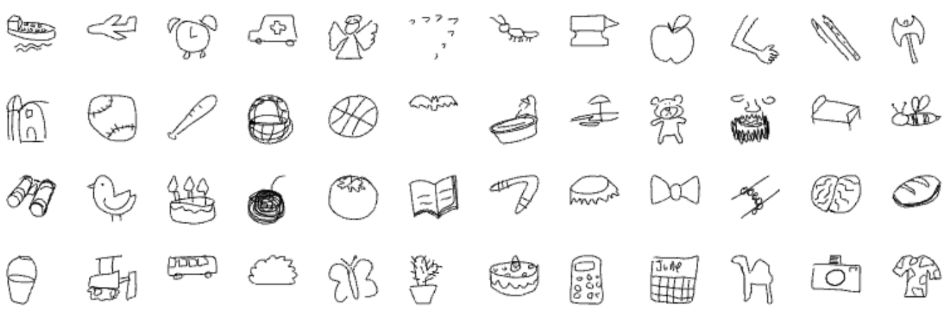

Figure 1 Sample from Quick, Draw! Dataset

The first approach in this area is SketchRNN[1]. The paper introduces a model that learns how to draw by treating sketches as a sequence of pen movements. The architecture uses a Sequence-to-Sequence autoencoder with an RNN to first encode an entire sketch into a single vector, and then decode it back into the original sequence of strokes. For training, the model is given a sketch and tries to predict the next pen movement (Δx, Δy, and pen state) at each step, making it learn how to complete and even create new drawings from scratch.

Another key piece of related work is SketchKnitter[2], an approach that shares several similarities with this project. The paper introduces an advanced model that generates sketches using a Diffusion U-Net, which is adapted for sequential data with 1D Convolutional layers. Similar to one of my own architectures, it takes noisy delta values as input and predicts the delta values and pen states separately through independent output heads.

Given the relevance of SketchKnitter, I attempted to train the official implementation on a single class of data using the default parameters. However, the model did not converge to producing meaningful results, an issue that has also been noted by others in the project's public code repository. Due to these replication challenges, I decided not to investigate this specific implementation further. Instead, the final architectures I propose in this report are novel hybrids that combine core concepts from both the SketchKnitter and the original SketchRNN papers.

3 Proposed Method

Following the literature review, I developed three distinct approaches to achieve the project's goal. This section will detail the architecture of each proposed method.

3.1 Dataset Preparation

To prepare the raw sketch data for the model, each drawing is transformed from a list of strokes with absolute (x, y) coordinates into a single, continuous 5D vector sequence. This process involves calculating the relative offset (Δx, Δy) between each point and the one preceding it. A 3-element one-hot vector is then appended to represent the pen's state: pen down, pen up (end of stroke), or end of drawing.

3.2 Diffusion Transformer

Given the sequential structure of the sketch data, I chose to implement a Transformer-based architecture rather than one relying on traditional Convolutional Layers. Transformers were originally developed for natural language processing, but their powerful self-attention mechanism, which excels at capturing long-range dependencies, has led to their successful adaptation for visual tasks as well. Today, this architecture remains a core component of many state-of-the-art approaches, including Large Language Models and modern Diffusion Models.

I proposed two different approaches. My first approach is designed for generating strokes unconditionally. The other approach is conditional, it takes pen state as a condition.

The model was trained using the AdamW optimizer with an initial learning rate of 1e-4. To manage this rate, a Cosine Annealing Scheduler was implemented, which gradually decreases the learning rate to a minimum of 1e-6 over the course of the training. For the diffusion process itself, the DDIM Scheduler was used, configured with 100 training steps with a sampler during training.

3.1.1 Unconditional Diffusion Transformer

In this approach, the model learns to generate a complete sketch holistically. The architecture is designed with a multi-task learning objective, where the continuous delta values and the discrete pen states are predicted separately from a shared internal representation.

Step-by-step process is as follows;

1. The delta values are noised and fed into the Transformer backbone.
2. The model's final hidden state is passed through two separate output "heads": one predicts the noise for the Δx, Δy values, and the other predicts the logits for the pen_state.

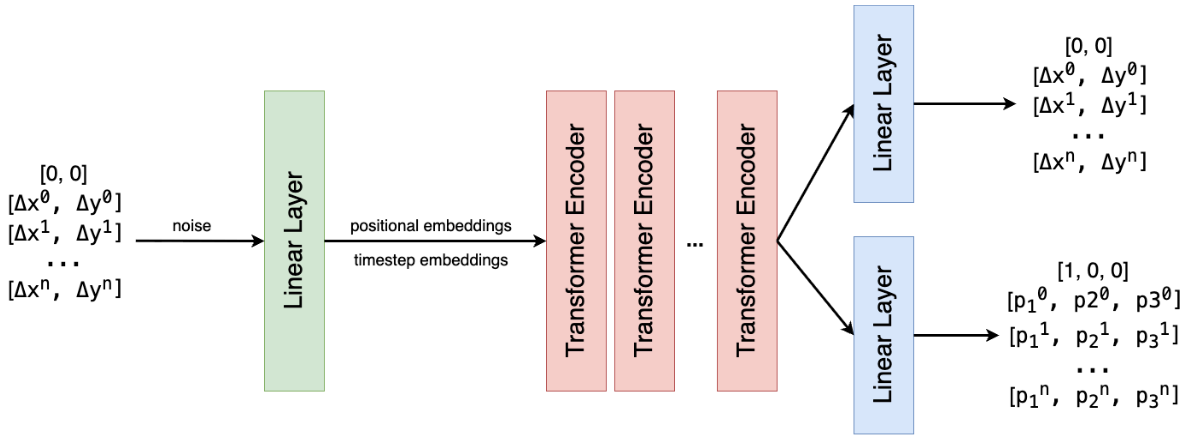

Figure 2 Unconditional Diffusion Transformer Architecture

This design, inspired by multi-headed outputs in models like SketchKnitter[2], allows the network to learn both continuous motion and discrete state predictions simultaneously. The training process utilizes a hybrid loss function: Mean Squared Error (MSE) is applied to the denoising task for the delta values, while Cross-Entropy Loss is used for the pen state classification.

3.1.2 Conditional Diffusion Transformer

This approach reframes the task as an inpainting problem, where the sequence of pen states serves as a known conditioning signal. The model's objective is to generate the correct movements (Δx, Δy) that correspond to this predefined sequence of actions.

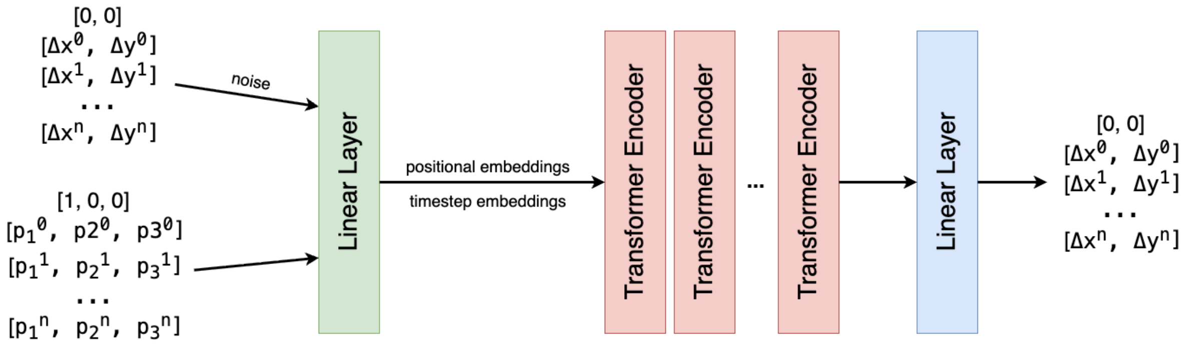

Figure 3 Conditional Diffusion Transformer Architecture

During the training process, the diffusion noise is applied only to the Δx, Δy values, while the pen_state vectors are kept clean. The model receives a composite input of both the noised deltas and the clean pen states. Its denoising objective is then to predict the noise solely for the delta values, using the clean pen states as a guiding context. This method explicitly teaches the model how a stroke should be shaped based on whether the pen is on the paper or being lifted.

This time only Mean Squared Error (MSE) is applied to the denoising task for the delta values.

3.2 Stroke History Conditioned Diffusion Transformer

For the final experiment, I developed a more advanced hierarchical and autoregressive architecture to better model the sequential nature of drawing. This approach pairs a history encoder with a conditional generator:

1. First, the stroke history is fed into an LSTM-based Stroke History Encoder. This network uses a bidirectional LSTM to process the sequence in both directions, allowing it to extract richer contextual relationships between strokes. A padding mask is used to prevent the model from forming associations with meaningless padded areas.
2. The resulting feature vector, which serves as a summary of the past, is then used to condition a Diffusion Transformer via cross-attention. This is a powerful method for guiding a generative process based on external information—in this case, the drawing's own history.
3. Finally, the Diffusion Transformer's task is to denoise the next target stroke. Guided by the context from the encoder, it predicts the noise using two separate output heads for the continuous delta values and the discrete pen states.

The loss function of this approach is similar to the Uncondition Transformer approach. Mean Squared Error (MSE) is applied to the denoising task for the delta values, while Cross-Entropy Loss is used for the pen state classification.

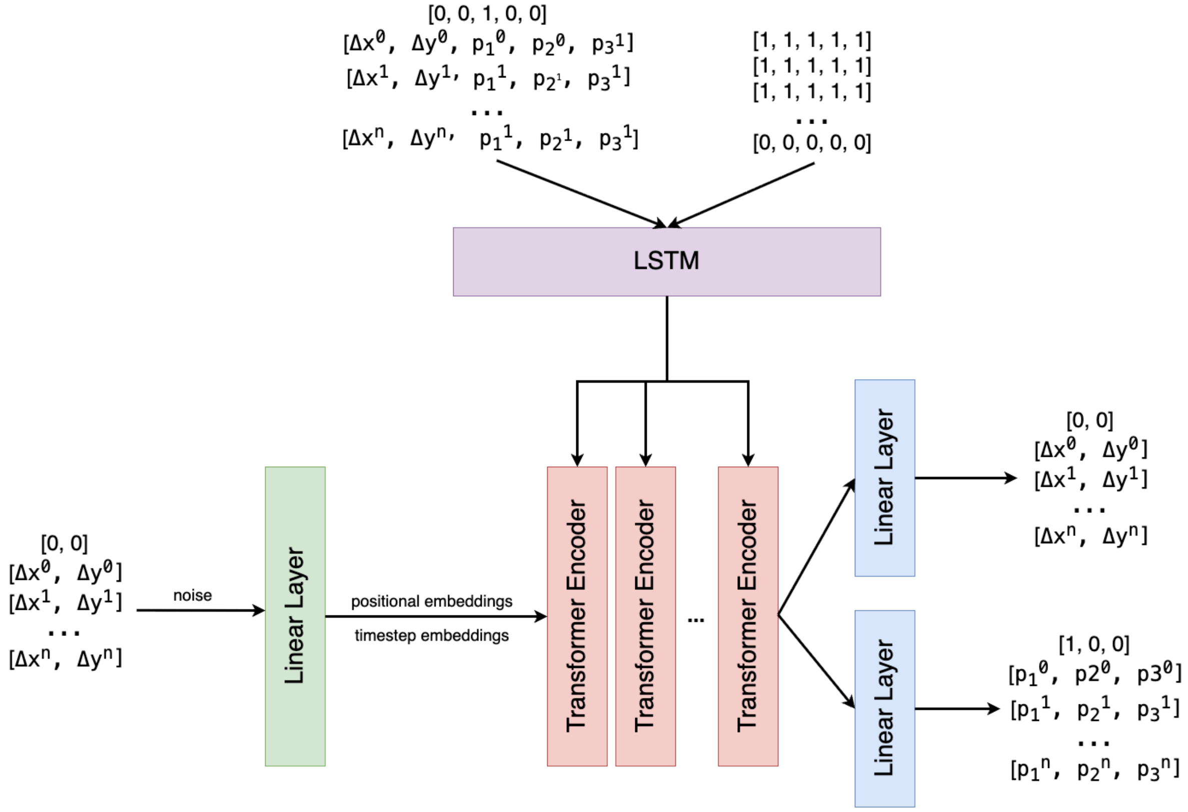

Figure 4 Stroke History Conditioned Diffusion Transformer Architecture

While this hierarchical method offers a step-by-step generation process, its primary drawback is the computational cost of inference. Because it operates autoregressively—generating one stroke at a time—the sampling speed scales linearly with the number of strokes, making it significantly slower than holistic approaches that generate the entire sketch at once.

4 Future Improvements

As stated previously, the models developed in this project did not converge to a state of producing successful, recognizable sketches. The outputs were generally composed of either blank canvases or incoherent strokes. Consequently, this report does not include a final section on qualitative or quantitative results, as such metrics would not be meaningful at this stage.

An initial experiment involved training the official SketchKnitter[3] implementation on the cat dataset. This model also failed to converge, which aligns with similar issues reported by other users of that repository. Assuming the results presented in the original paper are reproducible, one primary hypothesis for this challenge is the nature of single-class training. The original SketchKnitter model was trained on the entire Quick, Draw! dataset, which may allow it to learn a more generalized and robust representation of strokes before specializing in any one class. It's possible that training on the complete dataset first is crucial for stable convergence.

To mitigate this potential issue, I simplified the model architectures and experimented with various hyperparameter sets to reduce complexity. However, these attempts did not resolve the convergence problem.

As a direction for future work, a more effective strategy would likely be to pre-train the diffusion model on the entire dataset and then fine-tune it separately for each class. This would provide the model with a strong foundational understanding of drawing dynamics before it specializes. Unfortunately, this large-scale experiment was not possible for me due to time and computational constraints.

Finally, it is worth noting that the fundamental structure of this problem is highly analogous to Next Word Prediction, the core task used to train Large Language Models. Just as a language model predicts the next word based on the preceding text, a sketch generation model must predict the next stroke based on the history of previous strokes. This suggests that a purely autoregressive framework is a very natural and powerful approach for this task, even if it presents its own unique training challenges.

References

[1] Quick, Draw!
[2] Sketch-RNN
[3] SketchKnitter