DEEP LEARNING — ASSIGNMENT 3 REPORT

Saurabh Chavan	Akshat Patidar	Ram Prasad
B22EE061	B22EE087	B22EE054

Colab Link

SKETCH GENERATION MODEL REPORT

1. OBJECTIVE:

The goal of this project was to develop a sequence-to-sequence generative model capable of producing recognizable sketches based on a given class name. The model should learn the sequential nature of drawing strokes from the Google Quick, Draw! dataset and generate new stroke sequences step-by-step, mimicking human drawing behavior.

2. DATASET AND PREPROCESSING:

- **Dataset Source:** Google Quick, Draw! Dataset.
- **Classes Used:** The model was trained on 5 distinct classes: ['cat', 'clock', 'tent', 'house', 'nail']. The label_map assigned integer indices: {'cat': 0, 'clock': 1, 'tent': 2, 'house': 3, 'nail': 4}.
- **Dataset Size:** 10,000 recognized drawings were loaded per class, resulting in a total dataset size of **50,000** drawings.
- **Data Split:** The dataset was split into training, validation, and test sets using a 70:15:15 ratio:
    - Training Set: 35,000 samples
    - Validation Set: 7,500 samples
    - Test Set: 7,500 samples
- **Stroke Representation (Tokenization):**
    - Raw Quick, Draw! data consists of lists of (x, y) coordinate lists for each segment.
    - The strokes_to_npy function converted this into a sequence of delta movements [dx, dy, pen_state], where dx, dy are the changes in coordinates from the previous point, and pen_state is a binary flag (0: pen down, 1: pen up after this delta).

- The convert_npy_to_sequences function further processed this into the final 5-dimensional format used by the model: [dx, dy, p_down, p_up, eos].
    - p_down: 1 if the pen remains down for the *next* stroke, 0 otherwise.
    - p_up: 1 if the pen lifts *after* this stroke, 0 otherwise.
    - eos: 1 if this is the final stroke of the drawing, 0 otherwise.
- A **Start-of-Sequence (SOS)** token [0, 0, 1, 0, 0] was prepended to the input sequence.
- An **End-of-Sequence (EOS)** token [0, 0, 0, 0, 1] was appended to the target sequence.

- **Sequence Length:** Sequences longer than MAX_SEQ_LEN = 150 strokes (after conversion) were truncated.
- **Normalization:**
    - The dx and dy values across the entire training dataset were used to calculate mean and standard deviation.
    - Logs report: dx(mean=3.559, std=29.091), dy(mean=2.977, std=31.959).
    - The dx and dy components in both the input sequences (excluding SOS) and target sequences (excluding EOS) were normalized using standard scaling (subtract mean, divide by standard deviation). This helps stabilize training by bringing features to a similar scale.

3. MODEL ARCHITECTURE:

The model (SketchGenerator) is a conditional Recurrent Neural Network (RNN), specifically using LSTMs, designed for sequence generation.

- **Class Name Embedding:**
    - An nn.Embedding layer (class_embedding) converts the integer class label (0-4) into a dense vector representation.
    - Embedding Dimension: **128** (EMBEDDING_DIM).
- **LSTM Core:**
    - A multi-layer LSTM (lstm) processes the input sequence step-by-step.
    - Hidden Dimension: **512** (HIDDEN_DIM) per LSTM layer.
    - Number of Layers: **3** (NUM_LSTM_LAYERS).

- Dropout: A dropout rate of **0.3** (DROPOUT_PROB) was applied between LSTM layers to prevent overfitting.

- **Conditional Input:**
  - The 128-dimensional class embedding is **concatenated** with the 5-dimensional stroke input at *each timestep*.
  - Therefore, the actual input dimension to the LSTM is 5 (stroke) + 128 (class) = 133. This provides continuous conditioning information to the LSTM throughout sequence processing.
- **Initialization:**
  - The initial hidden state (h_0) and cell state (c_0) of the LSTM are generated by passing the class embedding through separate linear projection layers (init_h_proj, init_c_proj). This primes the LSTM with information about the target class from the very beginning.
- **Output Layers:**
  - The LSTM output at each timestep (512 dimensions) is passed through two separate linear layers:
    - fc_xy: Predicts the normalized dx and dy values (output size 2).
    - fc_pen: Predicts the logits for the three pen states (p_down, p_up, eos) (output size 3).
- **Total Parameters:** The model has **5,927,045** trainable parameters.

4. STROKE GENERATION MECHANISM:

The model generates strokes auto-regressively:

1. Provide the target class_label tensor.
2. Generate the class embedding and initialize the LSTM hidden/cell states.
3. Start with the SOS token [0, 0, 1, 0, 0] as the first input stroke.
4. **Loop:**
   a. Concatenate the current input stroke and the class embedding.
   b. Feed the combined input to the LSTM, updating the hidden state.
   c. Pass the LSTM output through fc_xy and fc_pen to get dx, dy predictions and pen state logits.
   d. **Sampling:**
   * Use the predicted dx, dy (after denormalization).

* Apply softmax to the pen state logits (potentially adjusted by temperature) and sample one of the three states (down, up, EOS).
e. Construct the predicted 5D stroke [dx_norm, dy_norm, p_down, p_up, eos].
f. Store the generated stroke (denormalized dx,dy).
g. Use the *normalized predicted* 5D stroke as the input for the next timestep.
h. Stop if EOS is sampled or MAX_SEQ_LEN is reached.

5. TRAINING PROCEDURE:

- **Loss Functions:**
    - A composite loss was used, calculated by calculate_loss:
        - **Mean Squared Error (MSE) Loss:** Applied to the predicted dx, dy coordinates compared to the target dx, dy. Calculated using nn.MSELoss.
        - **Cross-Entropy (CE) Loss:** Applied to the predicted pen state logits compared to the one-hot encoded target pen state (p_down, p_up, eos). Calculated using nn.CrossEntropyLoss.
    - The final loss was a sum of the MSE and CE losses, with equal weighting (LOSS_WEIGHT_XY = 1.0, LOSS_WEIGHT_PEN = 1.0).
    - Loss calculation was masked to exclude padding tokens based on original sequence lengths.
- **Optimizer:**
    - **AdamW** (optim.AdamW) was used, which is a variant of Adam with improved weight decay handling.
    - Initial Learning Rate: **0.001**.
    - Weight Decay: **1e-5**.
- **Gradient Clipping:** Gradient norms were clipped to a maximum value of **1.0** (GRADIENT_CLIP) during backpropagation to prevent exploding gradients, common in RNN training.
- **Learning Rate Scheduling:**
    - ReduceLROnPlateau scheduler was used, monitoring the validation loss (mode='min').
    - If the validation loss didn't improve for patience=3 epochs, the learning rate was reduced by a factor=0.5.

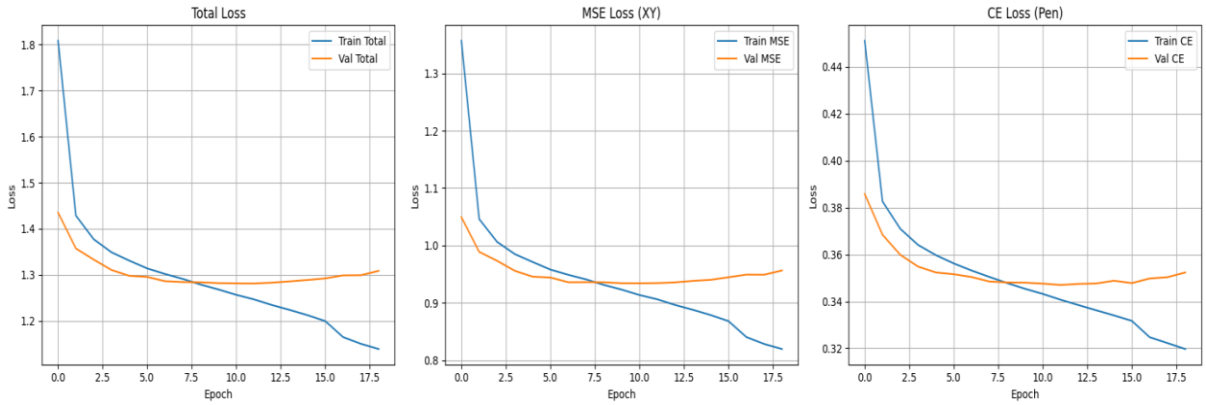
- The logs show the LR was reduced from 0.001 to 0.0005 at Epoch 16.

- **Early Stopping:**
  - Training was stopped early if the validation loss did not improve for patience=7 consecutive epochs.
  - The logs show training stopped after **19 epochs** because the validation loss (Val Loss) did not improve beyond the best value of **1.2814** (achieved at Epoch 12) for 7 epochs.
  - The model weights from the epoch with the best validation loss (Epoch 12) were saved and reloaded for final use.
- **Hardware:** Training was performed on a CUDA-enabled GPU (Using device: cuda).

6. RESULTS AND ANALYSIS (TRAINING CURVES):

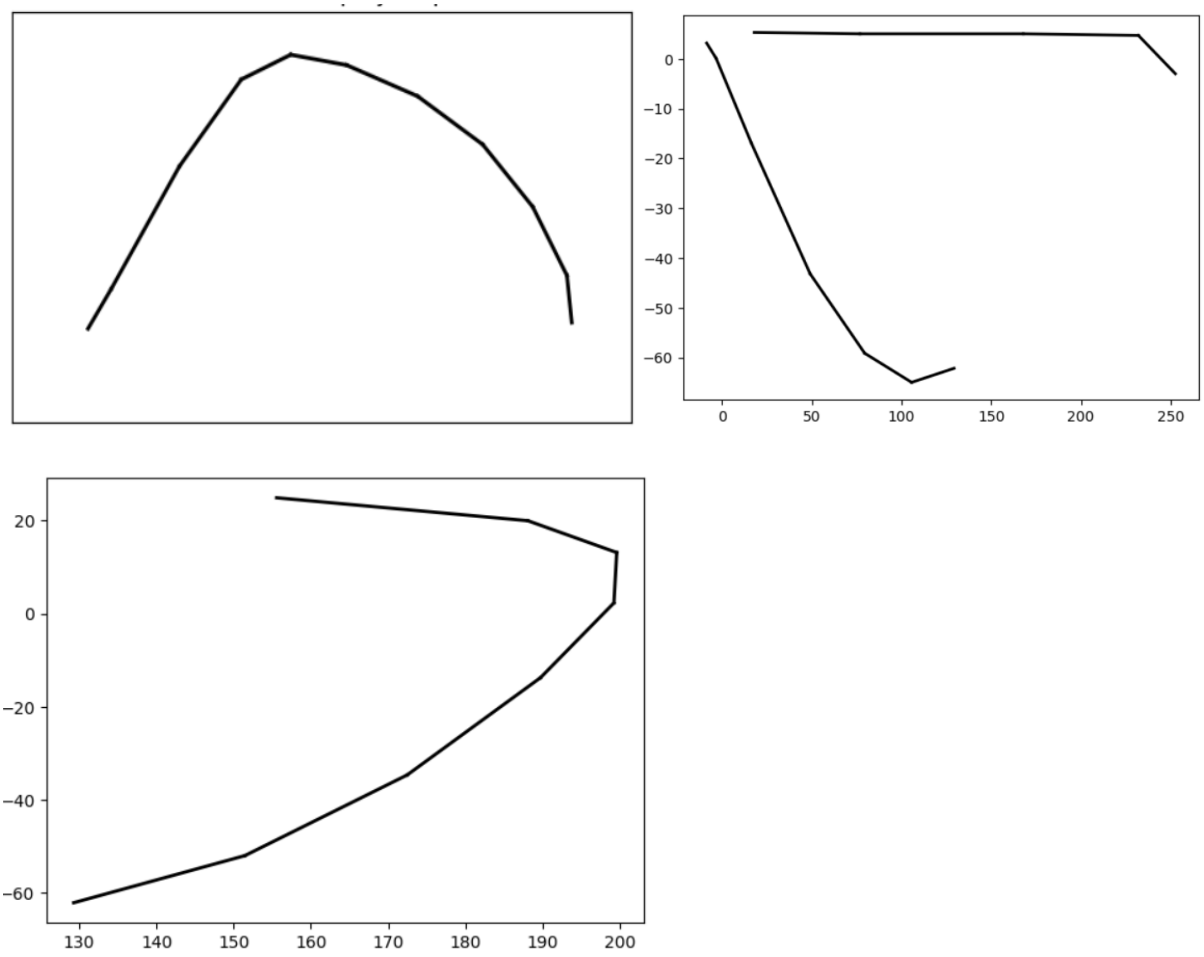
The provided training plots show the evolution of Total Loss, MSE Loss (XY coordinates), and CE Loss (Pen State) for both training (blue) and validation (orange) sets over the 19 epochs.

- **General Trend:** All losses decrease significantly in the initial epochs, indicating the model is learning effectively. The curves flatten out in later epochs.
- **Overfitting:** A clear gap emerges between the training and validation loss curves after approximately Epoch 7-10. The training loss continues to decrease steadily, while the validation loss plateaus and even starts to slightly increase towards the end (Epochs 13-19). This divergence is a classic sign of **overfitting**, where the model learns patterns specific to the training data that do not generalize well to the unseen validation data.
- **Justification for Early Stopping:** The plateauing/slight increase in validation loss while training loss continued decreasing strongly justifies the use of early stopping. Stopping at Epoch 19 prevented further overfitting and retained the model state (from Epoch 12) that performed best on unseen data.
- **Loss Components:** Both MSE and CE losses show similar trends, suggesting the model was learning both coordinate prediction and pen state prediction, but eventually started overfitting on both aspects simultaneously. The validation CE loss seems to plateau earlier and more definitively than the validation MSE loss.

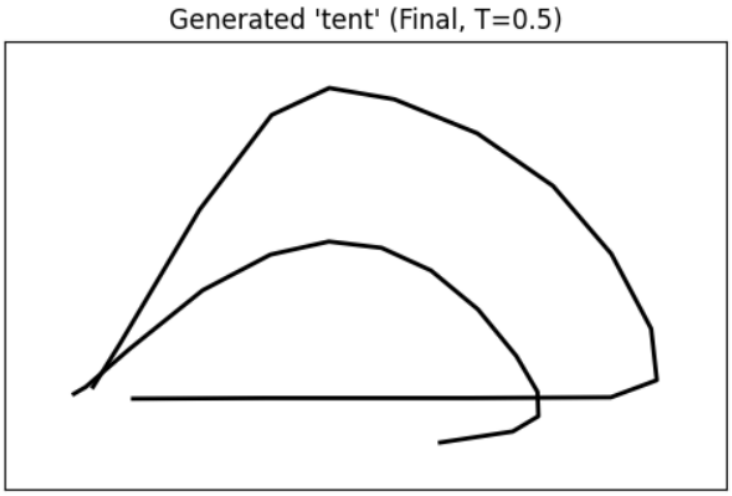


Generated Stroke-wise sketches of all classes-

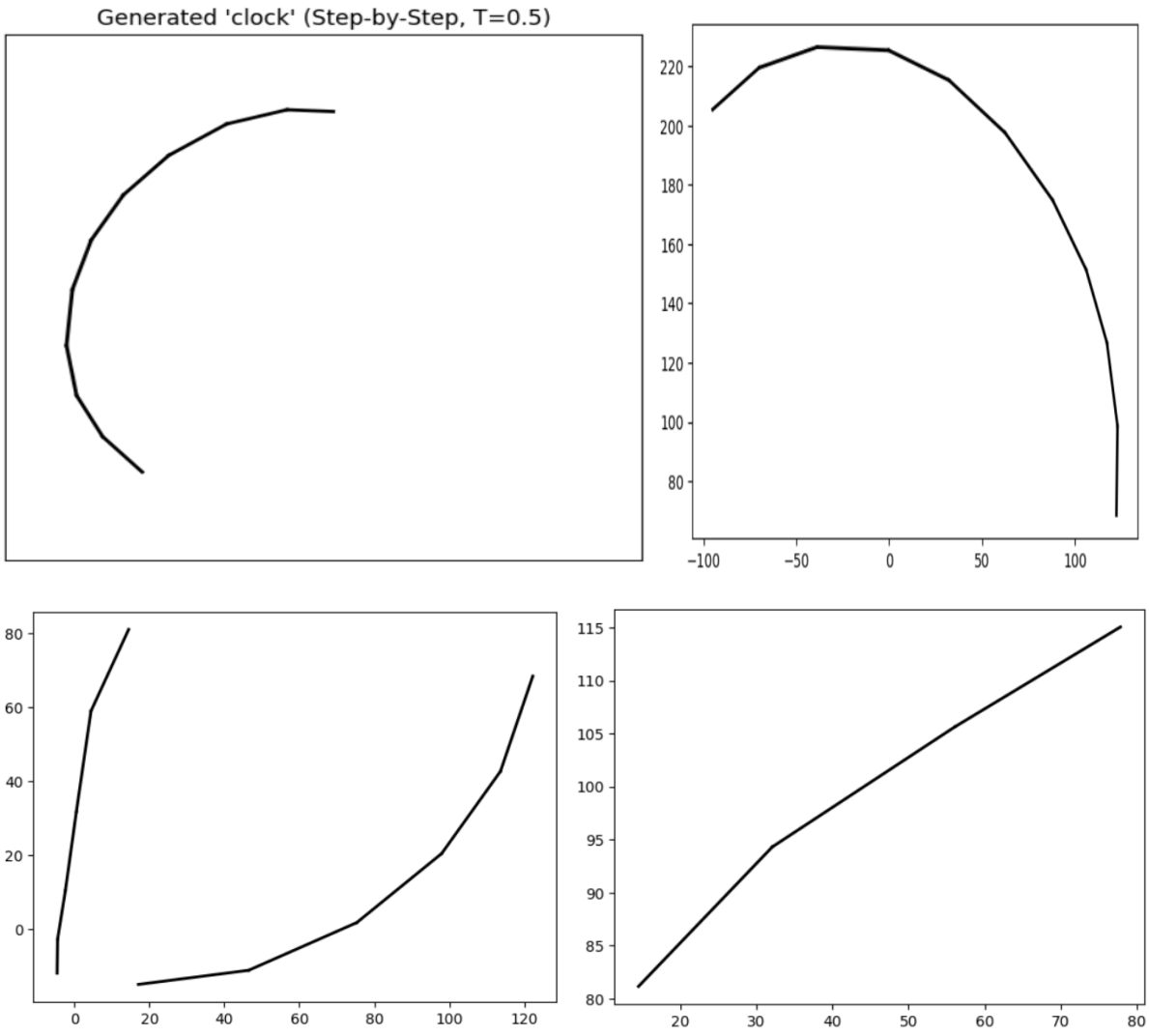
1. Tent



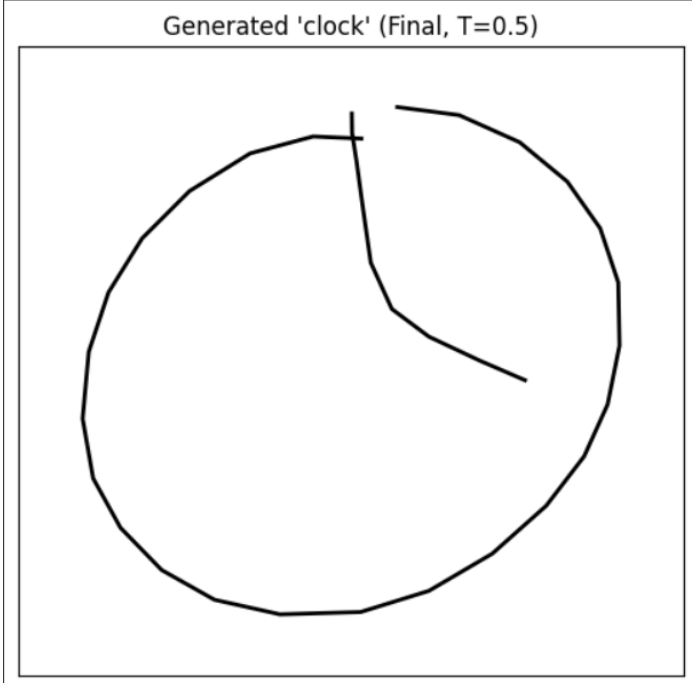
```
Final plot generated for 'Generated 'tent' (Final, T=0.5)'.
```



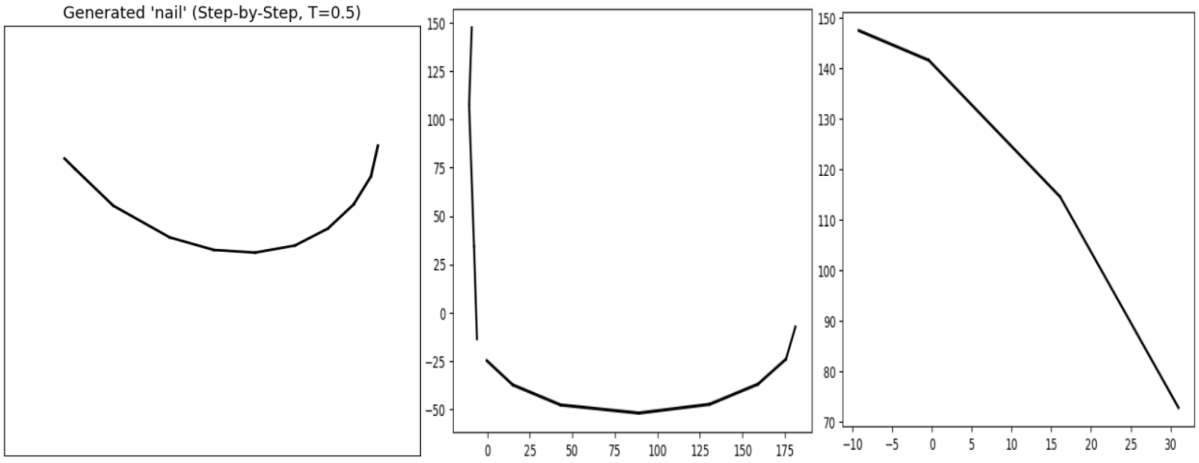
2. Clock



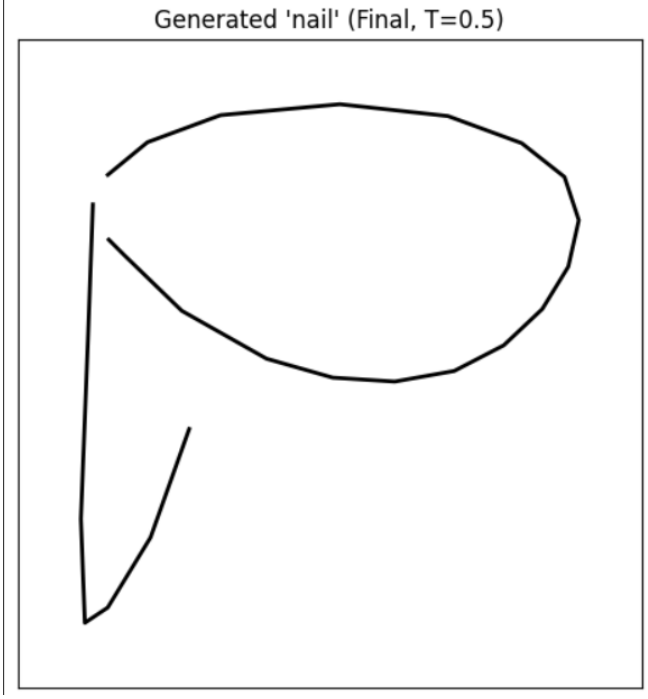
```
Final plot generated for 'Generated 'clock' (Final, T=0.5)'.
Saved final plot to plots/generated_clock_final_v3_final.png
```



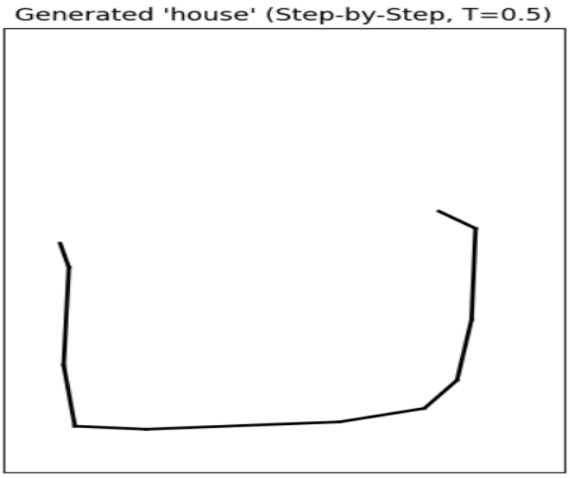
3. Nail

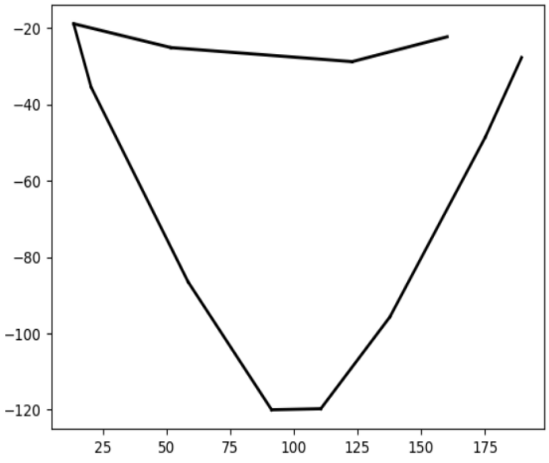


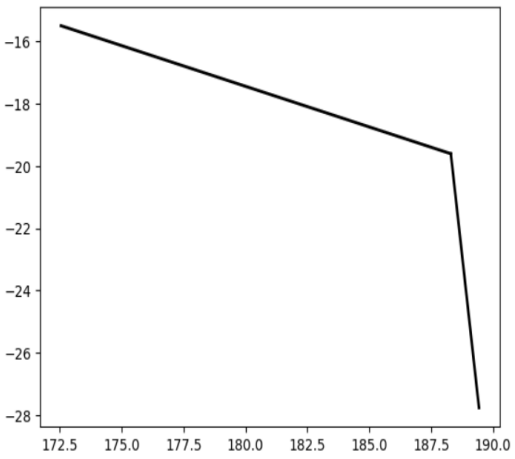
```
Final plot generated for 'Generated 'nail' (Final, T=0.5)'.
```



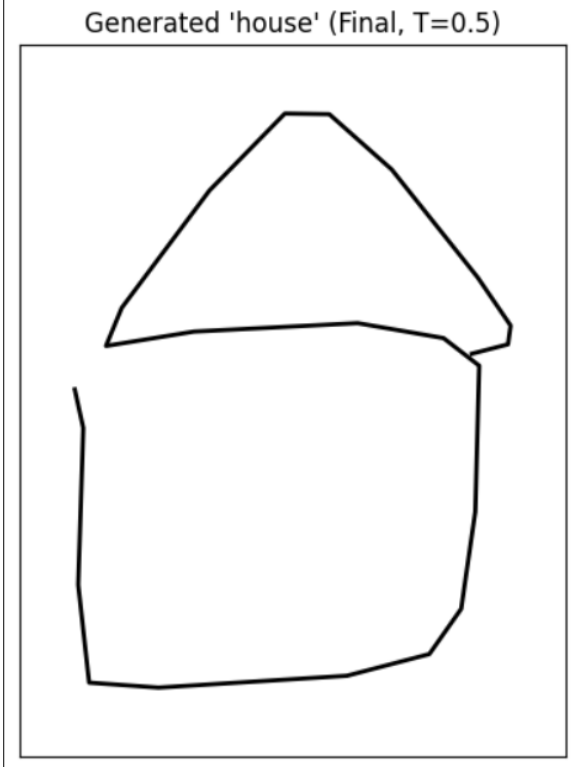
4. House



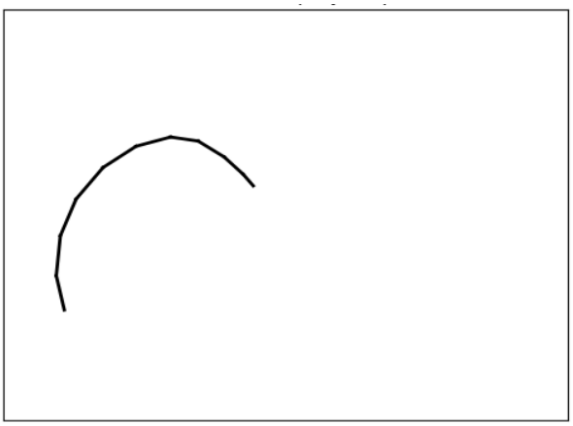


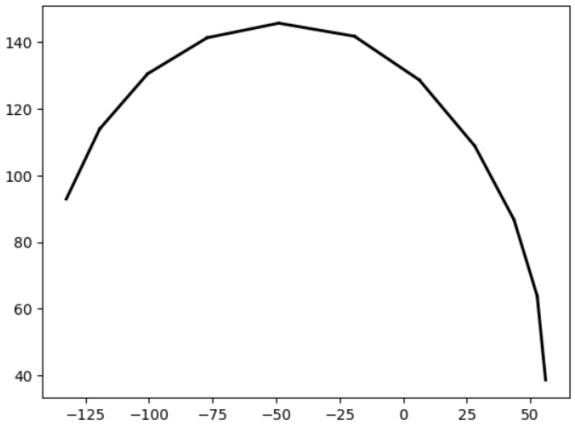


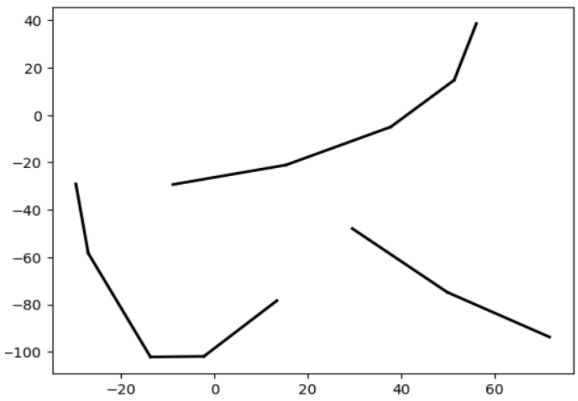
Final plot generated for 'Generated 'house' (Fin

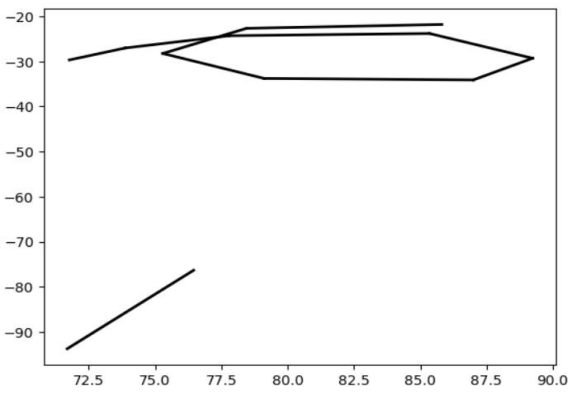


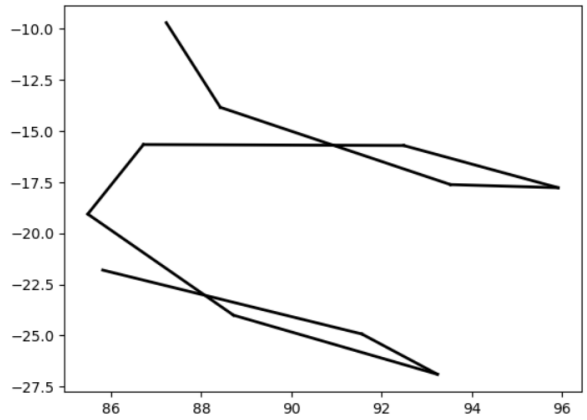
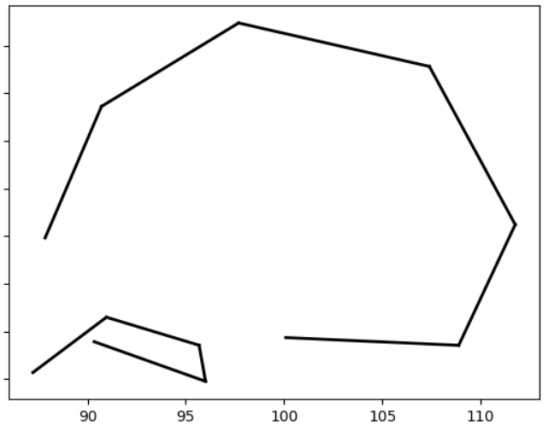
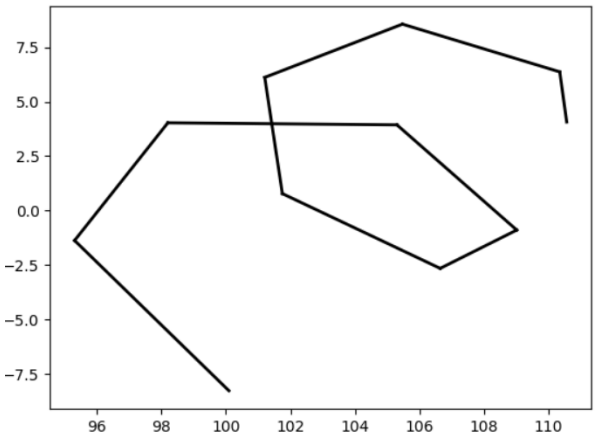
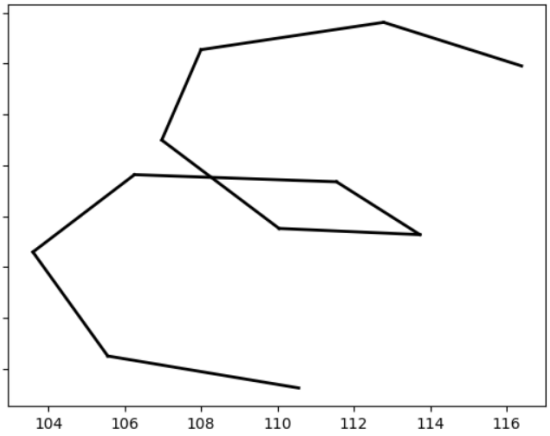
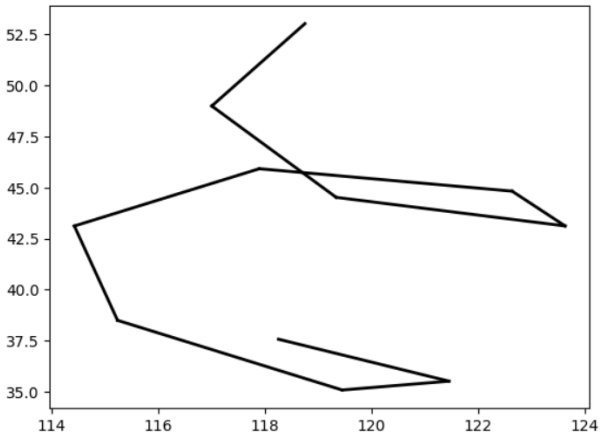
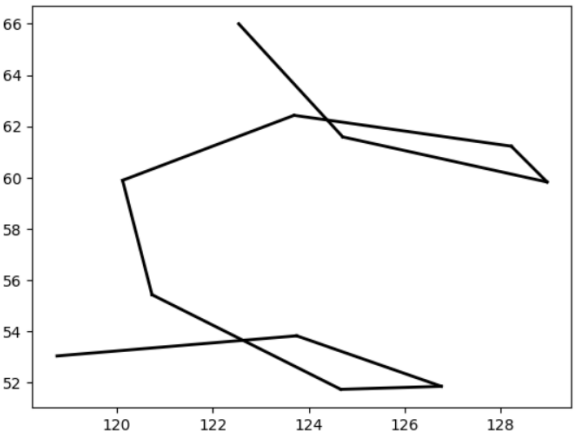
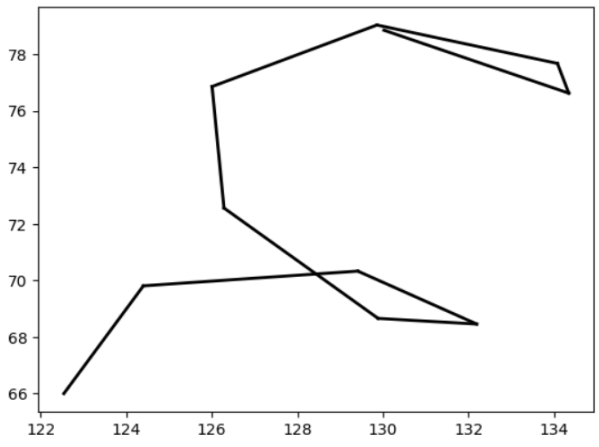
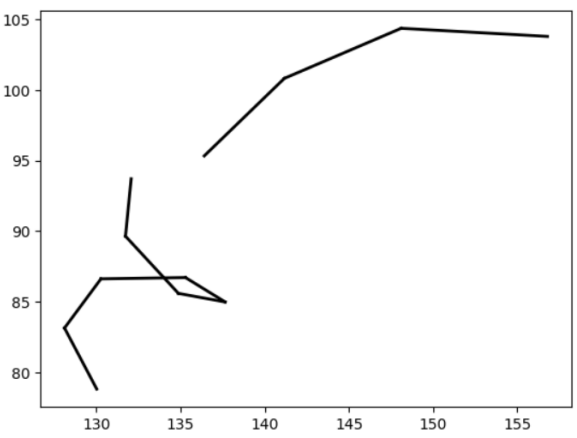
5. Cat



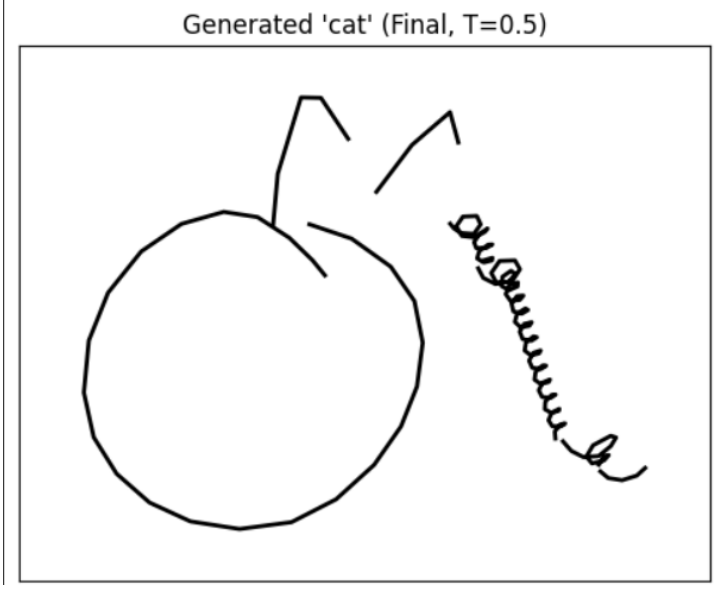






```
Final plot generated for 'Generated 'cat' (Final, T=0.5)'.
```



7. CONCLUSION:

The implemented conditional LSTM model successfully learned to generate stroke sequences from class labels. Key improvements included increasing model capacity (embedding size, hidden dimensions, LSTM layers) and incorporating continuous conditioning by concatenating the class embedding at each timestep. Data preprocessing involved careful tokenization into a 5D stroke format and normalization. Standard training techniques like AdamW, gradient clipping, learning rate scheduling, and early stopping were employed effectively, with early stopping preventing excessive overfitting as evidenced by the loss curves. While the model demonstrates learning, the quality of generated sketches (assessed visually in previous steps) might still require further improvements, potentially through more data, architectural refinements (like attention or GMM outputs), or hyperparameter tuning.