

## Module 2: Survey of Technologies

This project focuses on **handwriting synthesis**: generating human-like handwriting from text input by producing **stroke sequences** (e.g.,  $(x, y, pen\_state)$  tuples). The overall approach described in the report is:

1. sequence modeling using **RNNs (LSTM/GRU)** with an **MDN/GMM-style output head** to model continuous pen movement,
2. optional **style conditioning** using embeddings (to control writing style / few-shot style cloning),
3. optional **reinforcement learning (RL) fine-tuning** to optimize high-level style and legibility goals,
4. **rendering/export** of generated strokes to SVG/PNG for use in documents.

### 2.1 Front-End Technologies

#### 2.1.1 User Interface Frameworks

- **Next.js (Primary Frontend)**: A production-ready React framework providing SSR/SSG, API routes, and modern tooling for a professional user experience. Enables real-time handwriting generation, style upload/preview, parameter tuning (bias/temperature), and sample gallery browsing.
- **Command Line Interface (CLI)**: A CLI is well-suited for training and generation runs (e.g., `--text`, `--style`, `--steps`, `--output`). It is simple, reproducible, and easy to integrate into other scripts and batch workflows.
- **Notebook-based exploration**: Jupyter notebooks can be used to explore stroke data, visualize generated handwriting, and compare styles during experimentation and research validation.

#### 2.1.2 Front-End Design Tools

- **Matplotlib / Seaborn**: Visualization of stroke trajectories, loss curves, and qualitative comparisons.
- **PlantUML**: Used for software engineering diagrams (class, activity, sequence) to document the system.
- **VS Code**: Primary development environment for scripts, configs, and documentation.

### 2.2 Back-End Technologies

#### 2.2.1 Programming Language

- **Python (>= 3.10)**: Suitable for ML research/prototyping, data preprocessing, model training, and integration with deep learning frameworks.

## 2.2.2 Machine Learning Libraries

- **TensorFlow (sequence modeling reference)**: Commonly used for RNN-based handwriting synthesis implementations, including attention mechanisms.
- **PyTorch (alternative implementation option)**: Also widely used to build RNN/MDN generators and RL fine-tuning loops.
- **NumPy / SciPy**: Efficient handling of variable-length stroke sequences, normalization, filtering/smoothing, and resampling.
- **Scikit-learn**: Dataset splitting and supporting utilities.
- **Visualization libraries**: Matplotlib (and optionally Seaborn) for trajectory plots and results inspection.
- **Logging/monitoring**: TensorBoard-style logging for training curves and evaluation metrics.

## 2.2.3 Database Systems

- **PostgreSQL**: Primary database for user accounts, API credentials, style library management, and credits/usage tracking. Provides ACID guarantees and scalability for production deployments.
- **File-based storage**: Training datasets and model checkpoints remain filesystem-based (or cloud storage like S3):
  - **Numpy arrays / NPZ** for processed stroke datasets,
  - **model checkpoints** for trained weights,
  - **JSON/YAML** for configs and run metadata,
  - **SVG/PNG** for generated handwriting outputs.

## 2.3 AI/ML Technologies

### 2.3.1 Data Collection Tools

- **Online handwriting datasets**: Datasets such as **IAM-OnDB** (and similar public handwriting corpora) provide pen trajectories suitable for stroke-based modeling.
- **Text sources / label extraction**: ASCII transcripts paired with stroke data enable supervised learning (text-to-stroke alignment via attention).
- **Few-shot style samples**: Small user-provided stroke samples (short phrases) can be used to extract/condition a style embedding.
- **Synthetic augmentation (optional)**: Synthetic stroke perturbations (noise, skew, stretch) can improve robustness and style diversity.

### 2.3.2 Data Pre-processing Methods

- **Stroke representation**: Convert raw pen streams into sequences of  $(\Delta x, \Delta y, pen\_state)$  or  $(x, y, pen\_state)$ .
- **Normalization**: Centering/scale normalization (and optional de-skew) to reduce writer-specific global offsets.

- **Sequence trimming & padding:** Handle variable lengths via padding + masking (for batching) and limit max lengths for stable training.
- **Denoising / smoothing:** Optional smoothing filters to remove jitter and improve stability.
- **Train/validation/test splitting:** Split by writer or by sample, depending on whether the goal is writer-general generation or writer-specific style modeling.
- **Rendering for inspection:** Render intermediate stroke sequences to quickly verify preprocessing correctness.

### 2.3.3 Algorithms / Models Used

- **Sequence model (generator):**
  - **RNN (LSTM/GRU)** to model temporal dependencies in pen strokes.
  - **Attention mechanism** to align text character sequences with stroke generation.
- **Output distribution modeling:**
  - **Mixture Density Network (MDN) / Gaussian Mixture Model (GMM) head** to generate continuous-valued coordinates and pen-state probabilities.
- **Style conditioning:**
  - **Style embeddings** to control handwriting appearance and enable limited style cloning from few examples.
- **Reinforcement learning fine-tuning (optional):**
  - Policy-gradient style optimization using rewards like stroke smoothness, legibility, spacing consistency, and similarity to a target style.
- **Rendering:**
  - Convert strokes to **SVG** (vector) and **PNG** (raster) for export.

### Summary (Technology Stack at a Glance)

- **Frontend:** Next.js (React, SSR/SSG, API routes), CLI, Jupyter notebooks
- **Front-End Tools:** Matplotlib/Seaborn, PlantUML, VS Code
- **Backend Language:** Python (>= 3.10)
- **ML Libraries:** TensorFlow / PyTorch, NumPy/SciPy, Scikit-learn, Matplotlib, TensorBoard
- **Database:** PostgreSQL (user/credits/styles), filesystem/S3 (datasets/checkpoints)
- **Data:** IAM-OnDB and similar handwriting corpora, text transcripts, synthetic augmentation
- **Preprocessing:** Stroke normalization, sequence padding, denoising, train/val/test splits
- **Modeling:** RNN (LSTM/GRU) + Attention, MDN/GMM output head
- **Fine-tuning:** Optional RL with rewards (smoothness, legibility, style similarity)
- **Outputs:** SVG/PNG rendered handwriting