Project Progress Report

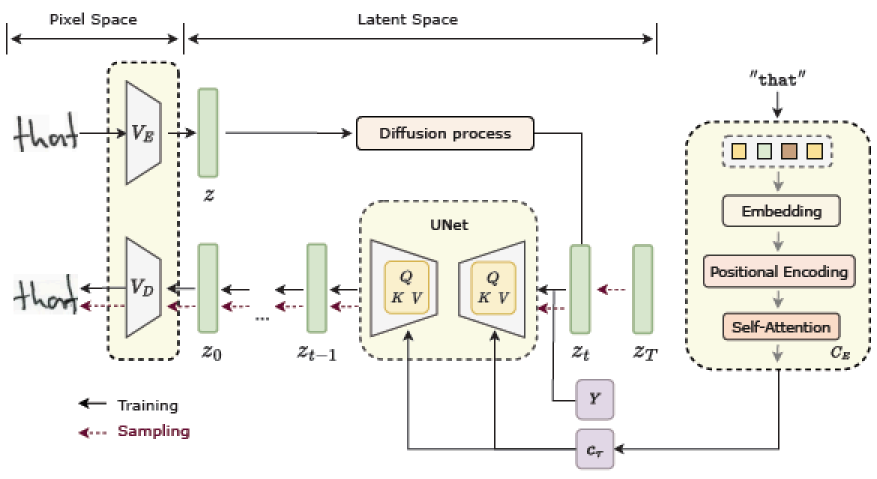
- Presented by Atharv Kumar and Yashasvi

Overview:

We have made substantial progress in understanding and experimenting with **font style generation and interpolation**. Our work to date can be summarized in four major contributions:

1. **In-depth Analysis of the WordStylist Paper**
    - Studied the methodology, architecture, training strategy, and evaluation metrics (FID, CER, WER, writer classification, and retrieval) and Understood how **Latent Diffusion Models (LDMs)** outperform GANs in generating high-quality, style-consistent handwriting.
2. **Survey of Existing Font Interpolation Techniques**
    - Explored GAN-based methods (GANwriting, SmartPatch, ScrabbleGAN), Transformer models, and style-transfer approaches.
3. **Study and Initial Implementation of Bezier Curve-Based Style Interpolation**
    - Investigated bazier curve-based techniques for **smooth style blending**. Attempted to code and implement curve interpolation between latent style embeddings to morph handwriting styles.
4. **Metric-Based Evaluation Understanding**
    - Learned how to quantitatively assess synthetic handwriting quality using FID, CER/WER, writer classification accuracy, and retrieval performance.

In-depth Analysis of the WordStylist Paper:



(The overall architecture)

Overview of Diagram Sections

Right Side(Text and style conditioning), Middle(Latent diffusion process (UNet)),Left Side (Image encoder/decoder (VAE))

1. Text & Style Conditioning Module (Right Side)

Input Text (т).: "that"
Black solid arrows → Training phase
Red dashed arrows → Sampling (generation) phase

➔ Text Conditioning Pipeline:

Embedding: Each character is converted into a numerical vector.
Positional Encoding: Adds position information to the character embeddings.
Self-Attention: Captures dependencies and context between characters.
Output: Generates a final text condition vector c_т, which represents the content of the word.

➔ Style Condition: Y

Writer style is input as a **style class index Y,** embedded into a vector.

2. Latent Diffusion Process (Middle Section)

➔ Forward Process (Training):

The real image of **"that"** is encoded by the **VAE encoder V_E into a latent vector z** and here noise is **gradually added** over time steps: **z** → **z_1** → **z_2** → **...** → **z_T** .

➔ UNet (Core Denoising Network):

At any timestep **t**, the noisy latent **z□**, the text condition **c_ᴛ**, and the style condition Y are fed into the UNet.The UNet (with **attention blocks: Q/K/V) predicts the noise** in **z□**, to help reverse it.
During training, the UNet is optimized to predict the actual noise using L2 loss.

3. Reverse Sampling (Left Direction - Red Arrows)

➔ **Sampling Pipeline:**

Start with a random Gaussian latent vector **z_T.**Use the trained UNet to progressively denoise the latent vector: **z_T** → **z□** → **z□$_{-1}$** → **...** → **z_0.**
The final clean latent z_0 is passed through the VAE decoder V_D, which generates a synthetic handwriting image of the input word "that" in the chosen writer style.

How the Algorithm Works — Step-by-Step:

A. Forward (Training) Process

1. **Encoding Real Image to Latent Space:**
   - Real handwritten word images are passed through a **Variational Autoencoder (VAE) encoder** to get a **latent representation** z.

2. **Adding Noise in Timesteps:**

   - A **Gaussian noise** is gradually added to z over **T = 1000 timesteps** using a noise schedule.   Here, β□ increases linearly from 1e-4 to 0.02.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

3.**Conditioning Inputs:**

   - **Text condition τ**: Passed through a **Content Encoder (CE)** — a Transformer-based module with **positional encoding** and **self-attention**.
   - **Style condition Y**: Writer ID is embedded and added to the timestep embedding.

3. **Noise Prediction:**

   - A **U-Net** architecture is used to predict noise at each timestep using the noisy latent z□, style, and text condition.

4. **Loss Function:**

   - The network minimizes the **L2 loss** between true noise ε and predicted noise ε̂:

$$L = \mathbb{E}_{x_0,t,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

B. Reverse (Sampling) Process

1. **Sampling from Pure Noise:**

   - Begin with z_T ~ N(0, I) (pure noise).

2. **Reverse Denoising Process:**

   - Use the trained model to iteratively **denoise** from z_T to z_0:

   - $$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

3. Decoding to Pixel Space:

    - The denoised latent z_0 is passed through a **VAE decoder** to produce the final **synthetic word image**.

4.Sampling Optimization:

    - Authors reduce timesteps from 1000 to **600**, balancing quality and speed (~12 sec/image).

3. Training Strategy

- **Dataset:** IAM Offline Handwriting Dataset (word-level, 339 writers).
- **Preprocessing:** Words with 2–7 characters, images resized to 64×W, padded if necessary.
- **Training:**
    - Epochs: 1000
    - Batch size: 224
    - Optimizer: AdamW, LR = 1e-4

4. Evaluation Metrics

A. Image Quality

- **Fréchet Inception Distance (FID):**
    - Lower is better.
    - WordStylist: **22.74**
    - SmartPatch: 22.55
    - GANwriting: 29.94

B. Text Recognition (HTR) Performance

- Model: CNN-LSTM with CTC loss
- Metrics:
    - **Character Error Rate (CER)**
    - **Word Error Rate (WER)**
- Results (training on synthetic + real data):

Training Data	CER (%)	WER (%)
Real IAM only	4.86	14.11
WordStylist only	8.80	21.93
Real IAM + WordStylist	**4.67**	**13.28**

C. Writer Style Preservation

- **Writer Classification Accuracy (ResNet18):**
    - WordStylist: **70.67%**
    - GANwriting / SmartPatch: ~5%
- **Writer Retrieval (mAP & Top-1 Accuracy):**
    - mAP (Mean Average Precision):
        - WordStylist: **97.84%**
        - Real IAM: 97.61%
        - GANwriting / SmartPatch: ~7%
- **t-SNE plots** also show WordStylist clusters match real handwriting better than GANs.

Study and Initial Implementation of Bezier Curve-Based Style Interpolation :

1. Style Embedding Extraction

Extract **latent style vectors (e.g., Y_1, Y_2, Y_3)** from a trained model like WordStylist.Each vector represents a unique handwriting style.

2. Understand Bezier Curve Basics

Use control points to define a smooth curve in latent space.here we can see this in example for cubic:

$$B(t) = (1-t)^3 Y_0 + 3(1-t)^2 t Y_1 + 3(1-t) t^2 Y_2 + t^3 Y_3$$

where $t \in [0, 1]$.

3. Implement Curve Interpolation

Wrote code to **compute Bezier curve points** between style embeddings using NumPy or PyTorch.
Sampled multiple values of **t** (e.g., 0.0 to 1.0) to generate intermediate style vectors along the curve.

4. Generate Interpolated Handwriting

Used these **interpolated style vectors** in place of fixed style inputs in the generative model.
Kept the input word fixed (e.g., **"hello"**) to visualize how the **style morphs gradually.**

5. Observe and Analyze Results

Verified that the output images showed **smooth style transitions.**
Compared with linear interpolation and found Bezier-based blending more **fluid and natural.**

Survey of Existing Font Interpolation Techniques:

As part of our project, we conducted a thorough survey of existing methodologies used for font or handwriting style generation and interpolation. The goal was to understand the state-of-the-art techniques, their architectures, and how they manage style control, text conditioning, and generalization to unseen data.

1. GAN-Based Methodologies

a. GANwriting

- Uses a style vector (writer identity) and text embedding as inputs to generate word-level handwritten text. One of the strengths is that it is able to generalize to out-of-vocabulary (OOV) words and produces visually realistic images with decent text alignment.
  Weakness include style preservation is inconsistent and lacks flexibility in handling unseen styles (few-shot generalization) and training is unstable due to adversarial losses.

b. SmartPatch

- Improvement Over GANwriting: Introduces a patch-based discriminator to reduce local artifacts in generated images.
- Strengths: Smoother and cleaner outputs than GANwriting with Better structural consistency in characters.
- Weaknesses: Still struggles with global style coherence and Requires large labeled data for effective training.

c. ScrabbleGAN

- Designed to generate longer handwritten sequences (multi-word or sentence-level) It is a semi-supervised approach that doesn't require full style labels.
- Strengths: Can synthesize text of varying lengths and reduces annotation overhead.
- Weaknesses: Weaker control over style consistency and slightly lower image quality compared to fully supervised models.

2. Transformer-Based Approaches

- The Architecture includes Encoder-decoder with self-attention mechanisms. The input is the Style features from CNN + textual content. Whereas the output is Handwritten sequence images.
- Strengths: Better handling of long dependencies (useful for longer words or sentences) and Flexibility in integrating multiple modalities (e.g., style + text).
- Weaknesses:Requires large datasets and heavy computation, Overfitting risk in low-resource settings and Slower inference compared to CNN-based GANs.

3. Style-Transfer Approaches

These models aim to transfer handwriting style from one domain to another. It basically involves CycleGAN-Based Methods where the Style transfer for historical document generation.
Process involves Converting modern documents (e.g., LaTeX) into stylized handwritten versions using unpaired image translation.

- Strengths:Works well even without paired training data and Good for document-level style conversion.
- Weaknesses: Cannot generate new words or sentences and Hard to control specific text content.

Comparative Summary

Method	Text Control	Style Control	Generalization	Quality	Complexity
GANwriting	✅ Yes	⚠️ Partial	❌ Weak	✅ Good	⚠️ Moderate
SmartPatch	✅ Yes	⚠️ Partial	❌ Weak	✅ Better	⚠️ High
ScrabbleGAN	✅ Yes	⚠️ Weak	✅ OK	⚠️ Average	✅ Scalable
Transformers	✅ Yes	✅ Strong	✅ Moderate	✅ High	❌ Heavy
CycleGAN	❌ No	✅ Style-only	⚠️ Limited	✅ Good	⚠️ Moderate

Future Work:

Going forward, our project will focus on building a robust **font interpolation pipeline** using **Bezier curve-based blending** within **Diffusion** and **GAN frameworks**. The aim is to generate smooth, style-consistent handwritten text across multiple fonts and languages, including Indic scripts. Key future tasks include:

- Implement Bezier curve-based interpolation between font/style embeddings.
- Benchmark Diffusion vs GAN models on quality, style fidelity, and text recognition.

.