Supplemental Material for

Generating Handwriting via Decoupled Style Descriptors

Atsunobu Kotani, Stefanie Tellex, and James Tompkin

Brown University

A Table of Variables

We include a brief table of the key variables used throughout the main manuscript and in this supplemental manuscript (Table 1).

Table 1: Brief explanation of key variables used throughout these manuscripts.

	Name	Shape	Note
$\mathbf{x}$	Input data	(N, 3)	A handwriting sample; a time sequence of 2D points.
$\mathbf{x}^*$	Encoded input	(N, 256)	A raw output from $f_\theta^{\text{enc}}(\mathbf{x})$.
s	Sentence	(M)	A string label for $\mathbf{x}$ (e.g., *hello*).
c_t	Substring	(t)	A substring of s (e.g., *he*).
$\mathbf{c}_t$	Character vector	(87×1)	A one-hot vector denoting the t-th character in s .
$\mathbf{c}_t^{\text{raw}}$	Encoded character	(256×1)	An output from $g_\theta^{\text{FC1}}(\mathbf{c}_t)$. Input for g_θ^{LSTM}.
$\mathbf{c}_{c_t}^{\text{raw}}$	Encoded substring	(256×1)	An output from $g_\theta^{\text{LSTM}}(\mathbf{c}_t^{\text{raw}})$.
$\mathbf{w}$	Writer-DSD	(256×1)	Content-independent handwriting style for a writer A.
$\mathbf{C}_{c_t}$	Character-DSD	(256×256)	An encoded character matrix for a substring c_t.
$\mathbf{w}_{c_t}$	Writer-Character-DSD	(256×1)	An encoded drawing representation for c_t, extracted from $\mathbf{x}^*$.
f_θ^{enc}	Sequence encoder		Outputs a list of Writer-Character-DSDs $\mathbf{w}_{c_t}$ from an input drawing $\mathbf{x}$.
f_θ^{dec}	Sequence decoder		Outputs a drawing $\mathbf{x}$ from a list of $\mathbf{w}_{c_t}$.
g_θ	Character encoder		Outputs a character matrix $\mathbf{C}_{c_t}$. Simplified function used as shorthand.
g_θ^{FC1}			Outputs a vector $\mathbf{c}_t^{\text{raw}}$ from a vector $\mathbf{c}_t$.
g_θ^{LSTM}			Outputs a vector $\mathbf{c}_{c_t}^{\text{raw}}$ from a list $[\mathbf{c}_1^{\text{raw}}, ..., \mathbf{c}_t^{\text{raw}}]$.
g_θ^{FC2}			Outputs a Character-DSD $\mathbf{C}_{c_t}$ from a vector $\mathbf{c}_{c_t}^{\text{raw}}$.
h_θ	Temporal encoder		LSTM to restore dependencies between Writer-Character DSDs $\mathbf{w}_{c_t}$.
k_θ	Segmentation function		Segments a handwriting sample $\mathbf{x}$ into characters.

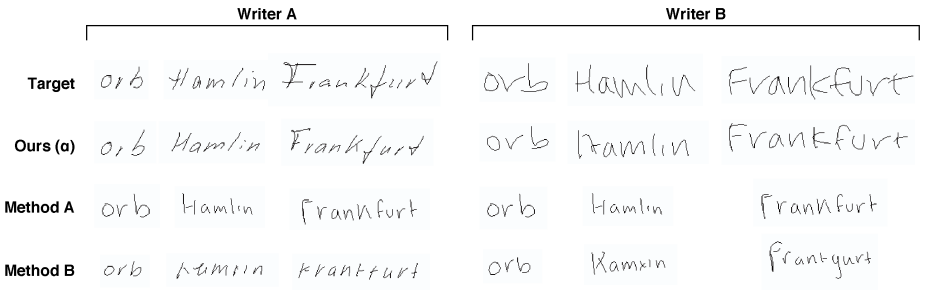


Fig. 1: Qualitative evaluation of two common style-transfer techniques.

B Comparison with Style Transfer Baselines

We evaluated our proposed model against two style transfer baselines. We define a style vector as $\mathbf{s} = f_\theta^{\text{enc}}(\mathbf{x})$ and a character-content vector as $\mathbf{c} = g_\theta(c_t)$. To interweave $\mathbf{s}$ and $\mathbf{c}$, we consider a new operator F where $F(\mathbf{s}, \mathbf{c}) = z$. Then, we feed z into our decoder function f_θ^{dec} to synthesize a drawing. We examined two operators for F: A) three stacked FC+ReLU layers, and B) AdaIN layer [5].

In our method, $f_\theta^{\text{enc}}(\mathbf{x})$ produces $\mathbf{w}_{c_t}$, which is then decoupled from the character content via our $\mathbf{C}$ matrix operation. In Method A and B, $f_\theta^{\text{enc}}(\mathbf{x})$ produces $\mathbf{s}$, and via F the network must implicitly represent content and writer style parts. For fairness, we keep the architectures of f_θ^{enc}, f_θ^{dec}, g_θ the same as in our approach, and train each method from scratch with the same data and loss function as in our approach.

Neither Method A or B is competitive with our method or with DeepWriting [1]. While A and B can generate readable letters, A) has only one style, and B) fails to capture important character shape details leaving some illegible, and has only basic style variation like slant and size (Figure 1). This is because f_θ^{enc} must represent a content-independent style for a reference sample without its content information. The DeepWriting model decouples style and content by making f_θ^{enc} additionally predict the reference content via a character classification loss. Our approach does not try to decouple style and content within f_θ^{enc}; instead our model extracts style from the output of f_θ^{enc} by multiplication with a content-conditioned matrix $\mathbf{C}$.

This simple experiment demonstrates that style-content decoupling is a difficult task. Instead of making one network (i.e., f_θ^{enc}) responsible for filtering out content information from the style reference sample implicitly, we show empirically that our method to structurally decouple content information via $\mathbf{C}$ matrix multiplication is more effective in the online handwriting domain.

C Investigating the C-matrix

The $\mathbf{C}$ matrix for a character string c_t—$\mathbf{C}_{c_t}$—is designed to contain information about how people generally write c_t: its role is to extract character(s)-specific

information from $\mathbf{w}_{c_t}$. Intuitively, the relationship between $\mathbf{C}_{c_t}$ and $\mathbf{w}_{c_t}$ can be seen as one of a key and a key-hole. Our model tries to create a perfect fit between a key ($\mathbf{w}_{c_t}$) and a key-hole ($\mathbf{C}_{c_t}$), where both shapes are learned simultaneously. But what if we fix the key-hole shape ahead of time, and simply learn to fit the key? That is, what if we assign pre-defined values to substring character matrices $\mathbf{C}$ ahead of time? This would reduce the number of model parameters, speed up training and inference, and allow us to store $\mathbf{C}$ in memory as a look-up table rather than predict its values.

One issue with fixing the $\mathbf{C}$ matrix is the exponential growth in the number of possible strings c_t as we allow longer words. Thus, for this analysis, we will initialize $\mathbf{C}$ for single- and two-character substrings only, which have a tractable number of variations in our Latin alphabet (Sec. K). For example, instead of $\mathbf{C}_{hello}$ for a word 'hello', we consider its five constituent single- and two-character substrings $\mathbf{C}_h$, $\mathbf{C}_{he}$, $\mathbf{C}_{el}$, $\mathbf{C}_{ll}$, $\mathbf{C}_{lo}$. Consequently, we modified the training data format by segmenting every sentence into two-character pairs.

We consider three scenarios (Figure 2):

Fixed random single- and two-character C In principle, each substring that $\mathbf{C}$ represents only needs to be *different* from other substrings, and so we assign a random matrix to each two-character substring.

Fixed well-spaced single- and two-character C Two matrices $\mathbf{C}_{sh}$ and $\mathbf{C}_{he}$ could contain mutual information about how to write the character h, and so we try to assign fixed matrices in a way that places similar substrings close to each other in high-dimensional space.

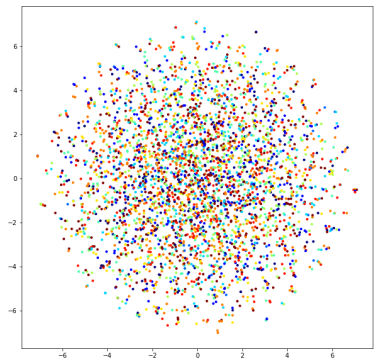
Learned single- and two-character C Our model trained only on single characters and two-character pairs. This trains g_θ to predict the values of $\mathbf{C}$.

Well-spaced C. If we randomly initialize $\mathbf{C}$ (i.e., $I(\mathbf{C}_{sh}; \mathbf{C}_{he}) \approx 0$), the values of two writer-character-DSDs, $\mathbf{w}_{he}$ and $\mathbf{w}_{she}$, must be significantly different from each other to output consistent $\mathbf{w}$, and this makes the learning task harder for the f_θ^{enc} LSTM. Instead, to determine how to manually initialize $\mathbf{C}$ such that they are more well spaced out, we look at the character information within $\mathbf{w}_{c_t}$. As we use an LSTM to encode the input drawing to obtain $\mathbf{w}_{c_t}$, it models long temporal dependencies. In other words, by the nature of LSTMs, $\mathbf{w}_{c_t}$ tends to 'remember' more recent characters than older characters, and so we assume $\mathbf{w}_{c_t}$ remembers the second character more than the first character. Thus, we initialize the character matrix for a two-character substring c_t as follows:

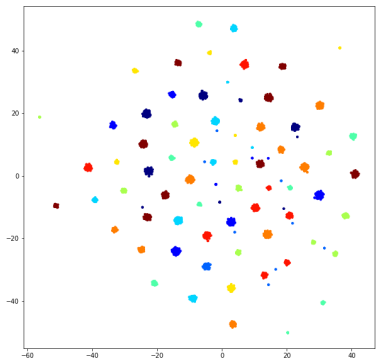
$$\mathbf{C}_{c_t} = r\mathbf{C}_{c_1} + (1.0 - r)\mathbf{C}_{c_2} \tag{1}$$

where $\mathbf{C}_{c_1}$ and $\mathbf{C}_{c_2}$ are randomly initialized single-character-DSDs, and we set $r = 0.1$. This leads to $\mathbf{C}_{c_t}$ that have the same ending character (i.e., c_2) having similar representations, as shown in Figure 2b.

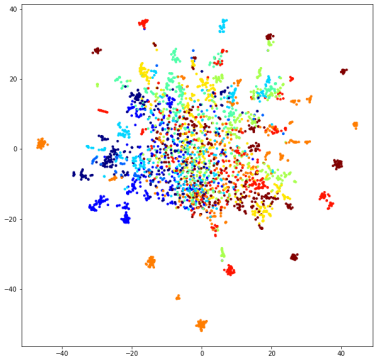
Results. Under t-SNE projections, the learned $\mathbf{C}$ models create more meaningful $\mathbf{C}$ representation layouts (Figure 2). Unlike the well-spaced $\mathbf{C}$, when we project $\mathbf{C}$ for two-character substrings (Figure 2c), we see a few outer clusters, with a larger



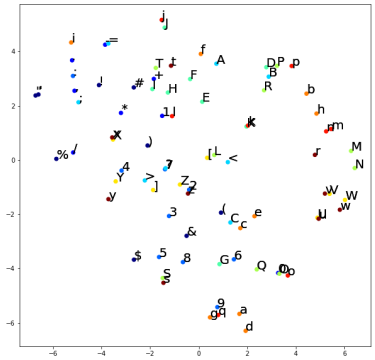
(a) Random **C** (2-character-string)



(b) Spaced **C** (2-character-string)



(c) Learned **C** (2-character-string)



(d) Learned **C** (1-character-string)

Fig. 2: t-SNE visualization of different **C**. Each dot indicates different substring. The substrings with the same last character (e.g., 'a**b**', 'b**b**', 'c**b**') are colored same. The learned **C** in Figure 2c are mostly concentrated in the middle. As each **C** contains information about how to draw *two* characters, even the two **C** with the same last character (e.g., 'a**b**' and 'b**b**') are often distant from each other, because of the different first character. By contrast, isolating the single characters within the learned **C** (Figure 2d) shows them to be well mapped in the space: similar characters such as '**(**', '**c**' and '**C**' are closely positioned.

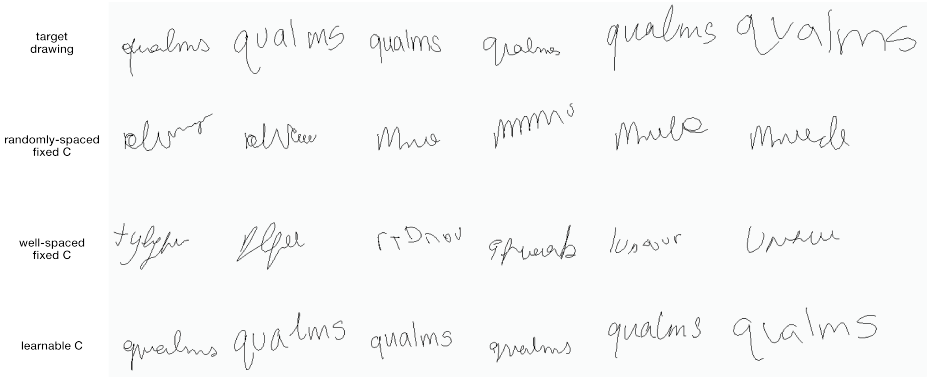


Fig. 3: Qualitative comparisons of results from different two-character **C**. When **C** are fixed through training, the models failed to synthesize recognizable letters.

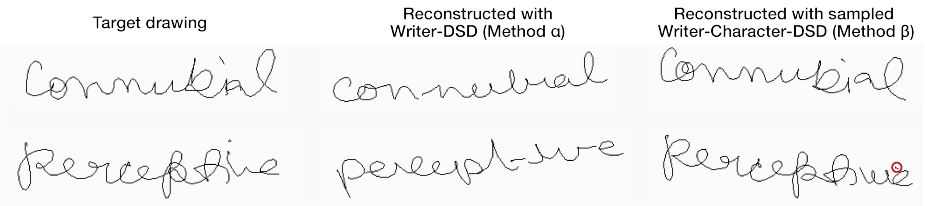


Fig. 4: Instances of missed delayed strokes by our proposed model.

'more chaotic' central concentration. As each dot in the projections represents **C** for two characters, these **C** cannot be easily clustered by the ending characters (e.g., considering general shapes, '**cb**' is likely to have a representation closer to the one of 'C**6**' than 'f**b**', despite the common 2nd character **b**). When looking at just the single-characters within our learned **C** representations (Figure 2d), characters with similar shapes (e.g., '9','q','g') are closely positioned, and this indicates a successful representation learning for **C**.

Figure 3 shows writing generation results from these different approaches. Both fixed **C** approaches fail to generate good samples.

Limitations of two-character substrings. One might think that using single- and two-character substrings could represent most variation in writing—how much do letters two behind the currently-written letter really affect the output? Cursive writing especially contains delayed strokes: for example, adding the dot for 'i' in 'himself' after writing 'f'. Changing to two-character substrings removes the ability of our model to learn delayed strokes in writings. In practice, our original **C** model can struggle to correctly place delayed strokes (Fig. 4): the model must predict a negative x-axis stroke to finish a previous character, which rarely occurs in our training set. This is one area for future work to improve.

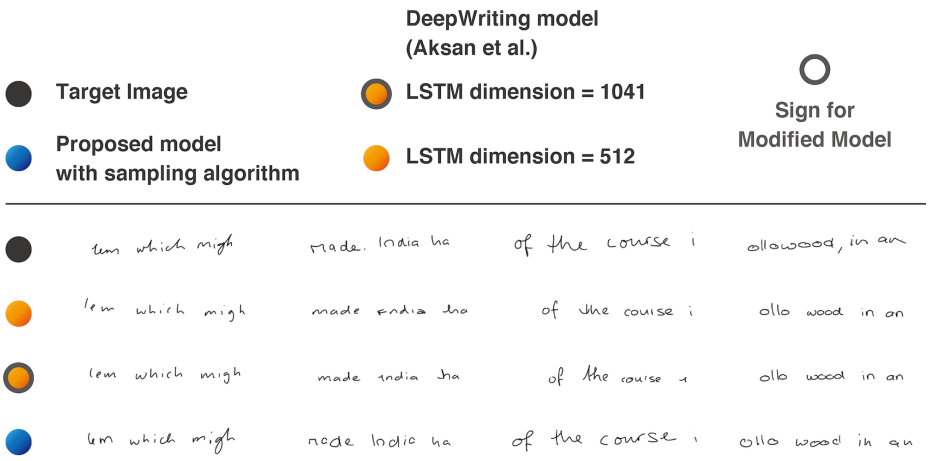


Fig. 5: To match the total number of parameters between DeepWriting and our model, we increased the LSTM dimension in DeepWriting from 512 to 1,041. There is little improvement in quality from the initial DeepWriting model of 512 LSTM dimension to 1,041. These drawings are generated with 10 sentence-level reference samples of the same writer.

D Network Capacity

To validate our network capacity, we conducted two comparison studies with the DeepWriting model by Aksan et al. [1]. The first is to decrease the number of parameters in our DSD-based model, and the second is to increase the number of parameters in the DeepWriting model.

D.1 Increasing DeepWriting Model Parameters

We modified the hidden state dimension for the DeepWriting model from 512 to 1,071, and the total number of parameters subsequently increased from $7.2M$ to $31.3M$. We show a side-by-side comparison of generated samples with DSD-256 model in Figure 5. For our 8GB VRAM GPU to accommodate this large LSTM, we decreased the batch size by half from 64 to 32, and doubled the number of learning rate decay steps from 1,000 to 2,000. However, increasing the capacity of the DeepWriting model did not improve the generated results (Figure 5).

D.2 Decreasing DSD Model Parameters

Is our decoupled model more efficient than the DeepWriting model [1], or simply more capacitive? With 256-dimensional latent vectors, our model has $31.33M$ parameters, whereas DeepWriting has $7.27M$. This difference is largely in the g_θ fully connected layer which expands $\mathbf{c}_{c_t}^{\text{raw}}$ into $\mathbf{C}_{c_t}$ via $16.84M$ parameters. As such, we reduced our latent vector dimension from 256 to 141, which leads

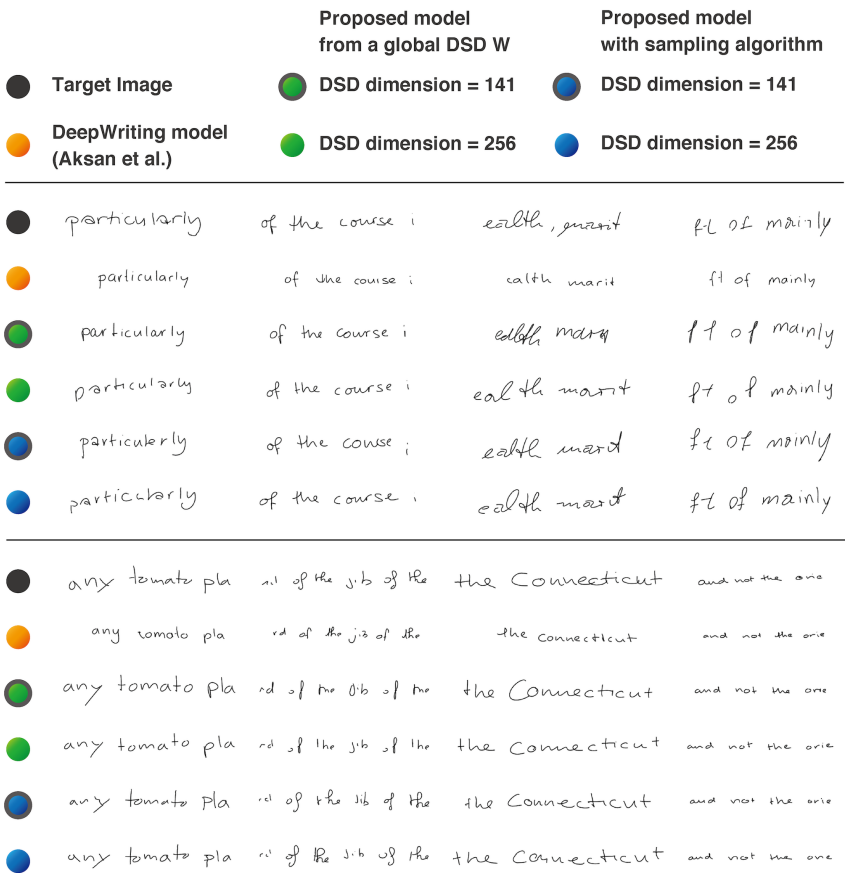


Fig. 6: We decreased the DSD dimension in our model from 256 to 141 to match the total number of parameters to DeepWriting. As we decrease DSD dimensions, there is a slight fall in quality, particularly the examples with green dots that are generated from a single global writer-DSD $\overline{\mathbf{w}}$ in Method α.

to a model with $7.25M$ parameters. While we observe minor deterioration in generation quality (Figure 6), the model still creates higher-quality samples than DeepWriting. This suggests that our architecture is more efficient.

E Further Generated Comparisons

Figure 7 and 8 show all 40 samples of drawing used for our qualitative/quantitative study on Amazon Mechanical Turk.

Fig. 7: The first 20 out of 40 samples used for quantitative evaluation.

Fig. 8: The second 20 out of 40 samples used for quantitative evaluation.

Fig. 9: Generated image by our sampling algorithm. The black letters in the synthesis indicate that they are predicted from $\overline{\mathbf{w}}$, while the colored characters in reference samples are encoded and saved in the database in the form of writer-character-DSDs $\mathbf{w}_{c_t}$ and retrieved during synthesis.

F Sampling Algorithm for Writer-Character-DSD $\mathbf{w}_{c_t}$

When handwriting samples $\mathbf{x}$ with corresponding character strings s are provided for inference, we can extract writer-character-DSDs $\mathbf{w}_{c_t}$ from $\mathbf{x}$ for substrings of s. For example, for character string *his*, we can first extract the following 3 arrays of writer-character-DSDs using f_θ^{enc}: $[\mathbf{w}_h]$, $[\mathbf{w}_h, \mathbf{w}_{hi}]$, and $[\mathbf{w}_h, \mathbf{w}_{hi}, \mathbf{w}_{his}]$. In addition, if the handwriting is non-cursive and each character is properly segmented, then we can also obtain 3 more ($[\mathbf{w}_i]$, $[\mathbf{w}_i, \mathbf{w}_{is}]$, and $[\mathbf{w}_s]$). However, we must ensure that the handwriting is cursive, as *h*, *i*, and *s* could be connected by a single stroke. In such cases, we only extract the first 3 arrays.

We create a database D of these arrays of writer-character-DSDs with substrings as their keys, and query substrings in the target sentence s^* for generation to obtain relevant writer-character-DSDs. We also compute the mean global writer-DSD $\overline{\mathbf{w}}$ as $\overline{\mathbf{w}} = \frac{1}{N}\sum_{c_t} \mathbf{C}_{c_t}^{-1}\mathbf{w}_{c_t}$ where N is the number of obtained $\mathbf{w}_{c_t}$.

To synthesize a sample *thin* from *his*, we query the substring *hi* and receive an array of DSDs: $[\mathbf{w}_h, \mathbf{w}_{hi}]$. As $\mathbf{w}_t$ and $\mathbf{w}_n$ are computed from $\overline{\mathbf{w}}$:

$$\mathbf{w}_t^{\text{rec}} = h_\theta([\mathbf{w}_t]) \tag{2a}$$
$$\mathbf{w}_{th}^{\text{rec}} = h_\theta([\mathbf{w}_t, \mathbf{w}_h]) \tag{2b}$$
$$\mathbf{w}_{thi}^{\text{rec}} = h_\theta([\mathbf{w}_t, \mathbf{w}_{hi}]) \tag{2c}$$
$$\mathbf{w}_{thin}^{\text{rec}} = h_\theta([\mathbf{w}_t, \mathbf{w}_{hi}, \mathbf{w}_n]) \tag{2d}$$

We use $[\mathbf{w}_t, \mathbf{w}_{hi}]$ instead of $[\mathbf{w}_t, \mathbf{w}_h, \mathbf{w}_{hi}]$ in Equations 2c and 2d because, as one might recall from generation Method β in the main paper (Sec. 3), the function approximator h_θ is designed to *restore* temporal dependencies between writer-character-DSDs. As 'h' and 'i' are already temporally *dependent* within $\mathbf{w}_{hi}$, we need only connect characters 't' and 'h' through LSTM h_θ. The pseudocode for this sampling procedure is shown in Algorithm 1, with example generations in Figure 9.

Algorithm 1: Pseudocode for our sampling algorithm to reconstruct writer-character-DSDs for the target sentence to synthesize.

Input: D: database of writer-character-DSD, s^*: target sentence to generate, $\overline{\mathbf{w}}$: mean global writer-DSD

Function `PerformSamplingAlgorithm(`D, s^*, $\overline{\mathbf{w}}$`):`
- Initialize empty sets L, R and *result*
- $s^* \leftarrow$ MarkAllCharactersAsUncovered(s^*)
- $\mathbf{ss}^* \leftarrow$ ExtractSubStringsAndOrderByLength(s^*)
- **for** each substring ss in $\mathbf{ss}^*$ **do**
  - **if** ss is in D and every characters in ss are not-covered **then**
    - $[\mathbf{w}_{c_1}, ..., \mathbf{w}_{c_t}] =$ QueryDatabaseWithKey(ss)
    - Add $[\mathbf{w}_{c_1}, ..., \mathbf{w}_{c_t}]$ to L
    - $\mathbf{s}^* \leftarrow$ MarkCharactersInSubstringAsCovered($\mathbf{s}^*$, ss)
- **for** each uncovered character c_t in s^* **do**
  - $\mathbf{w}_{c_t} \leftarrow \mathbf{C}_{c_t}\overline{\mathbf{w}}$
  - Add $[\mathbf{w}_{c_t}]$ to L
- $L^* \leftarrow$ OrderSetBySubstringAppearanceIn(s^*)
- **for** each array A in L^* **do**
  - **for** each $\mathbf{w}_{c_i}$ in A **do**
    - $\mathbf{w}_{c_i}^{\text{rec}} \leftarrow h_\theta([R_1, R_2, ..., \mathbf{w}_{c_i}])$
    - Add $\mathbf{w}_{c_i}^{\text{rec}}$ to the *result* list
    - **if** $\mathbf{w}_{c_t}$ is the last element in A **then**
      - Add $\mathbf{w}_{c_i}$ to the reference set R
- **return** *result*

G Sequence Decoder f_θ^{dec}

To synthesize a new sample from a list of writer-character-DSD $\mathbf{w}_{c_t}$, we train a sequence decoder function f_θ^{dec}. The inputs to this decoder are: 1) initial point $p_0 = (0, 0, 0)$, and 2) the first writer-character-DSD $\mathbf{w}_{c_1}$. Continuing with the *thin* example, we predict the first point p_1 from p_0 and $\mathbf{w}_t$. At runtime, the predicted point p_1^* will be fed into the LSTM at the next timestep to predict p_2. When the decoder model outputs an $eoc > 0.5$ (end-of-character probability), the model stops drawing the current character and start referencing the next writer-character-DSD so that it starts drawing the next character. This procedure is illustrated as the red lines in Figure 10. Similarly, to determine the touch/untouch status of the pen to the canvas, we use the *eos* (end-of-stroke probability) which is enclosed in point prediction p_t^*. If $eos_t > 0.5$, our model lifts up the pen; if $eos_t \leq 0.5$, our model continues the stroke.

Note that when we use the predicted p_t^* as an input to the LSTM at runtime, we binarize the *eos* value. This is because all *eos* values in training data are binarized. Further, we do not use the predicted points to predict the next point

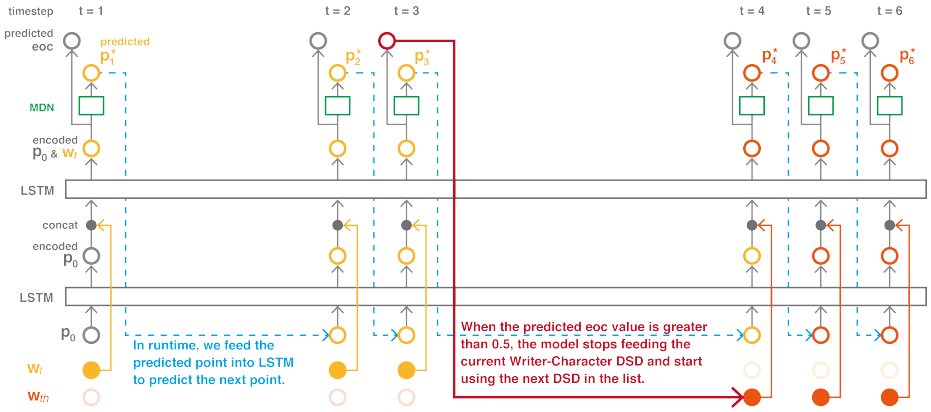


Fig. 10: Overview of our decoder architecture. During training, we feed true point sequences to the LSTM and do not use the predicted output p_t^* as the next input (the procedure shown as dotted blue lines).

Fig. 11: Variations in generated results from a single writer-character-DSD $\mathbf{w}_{c_t}$, achieved by sampling points from predicted MDN distributions.

during training, because we have the true point sequence $\mathbf{x}$. In other words:

$$p_{t+1}^* = f_\theta^{\text{dec}}(p_0, p_1, ..., p_t | \mathbf{w}_{c_t}) \qquad (training) \tag{3a}$$

$$p_{t+1}^* = f_\theta^{\text{dec}}(p_0, p_1^*, ..., p_t^* | \mathbf{w}_{c_t}) \qquad (runtime) \tag{3b}$$

where $*$ indicates *predicted* outputs by the decoder network.

Finally, the mixture density networks [2] (MDN) layer in our decoder makes it possible for our model to generate varying samples even from the same writer-character-DSD $\mathbf{w}_{c_t}$. Examples are shown in Figure 11.

H Character Encoder Function g_θ

Next, we discuss in detail how the character matrix $\mathbf{C}$ is computed. First, we convert each one-hot character vector $\mathbf{c}_t$ in the sentence $\mathbf{s}$ into a 256 dimensional

vector $\mathbf{c}_t^{\text{raw}}$ via a fully-connected layer g_θ^{FC1}. Then, we feed this vector into LSTM g_θ^{LSTM} and receive outputs $\mathbf{c}_{c_t}^{\text{raw}}$ of the same size. g_θ^{LSTM} is designed to encode temporal dependencies among characters. Then, we use a mapping function g_θ^{FC2} to transform the 256×1 vector into a $65,536$ dimensional vector, and finally reshape the output vector to a 256×256 matrix $\mathbf{C}_{c_t}$. This process is as follows:

$$\mathbf{c}_t^{\text{raw}} = g_\theta^{\text{FC1}}(c_t) \tag{4a}$$

$$\mathbf{c}_{c_t}^{\text{raw}} = g_\theta^{\text{LSTM}}([\mathbf{c}_1^{\text{raw}}, ..., \mathbf{c}_t^{\text{raw}}]) \tag{4b}$$

$$\mathbf{C}_{c_t} = Reshape(g_\theta^{\text{FC2}}(\mathbf{c}_{c_t}^{\text{raw}})) \tag{4c}$$

The parameters in g_θ^{FC2} take up about one third of total number of parameters in our proposed model; this is expensive. However, using a fully-connected layer allows each value in the output $\mathbf{C}_{c_t}$ to be computed from all values in the 256-dimensional vector $\mathbf{c}_{c_t}^{\text{raw}}$. If each value in $\mathbf{c}_{c_t}^{\text{raw}}$ represents some different information about the character, then we intended to weight them $65,536$ times via distinct mapping functions to create a matrix $\mathbf{C}_{c_t}$. We leave the study of other possible g_θ architectures for future work.

I Segmentation Network k_θ

We introduce an unsupervised training technique to segment sequential handwriting samples into characters without any human intervention. For comparison, the existing state-of-the-art DeepWriting handwriting synthesis model [1] relies on commercial software for character segmentation.

Our data samples for training arrive as stroke sequences and character strings, with no explicit labeling on where one character ends and another begins within the stroke sequence. As such, we train a segmentation network k_θ to segment sequential input data $\mathbf{x}$ into characters, and to predict *end of character (eoc)* labels for each point in $\mathbf{x}$. Relying on these predicted *eoc* labels, we can extract $\mathbf{w}_{c_t}$ from encoded $\mathbf{x}^*$ and synthesize new samples with f_θ^{dec}.

To prepare the input data for training, we extract 23 features per point p_t in $\mathbf{x}$, as is commonly used in previous work [4,6,7]. We feed these into a bidirectional LSTM to output a probability distribution over all character classes. From this output O of size (N, Q), where N is the input sequence length and Q is the total number of characters, we compute a loss that is similar to a connectionist temporal classification (CTC) loss [3]. As seen in Figure 12, we make a slight modification in connections among nodes to adjust the change in two domains: character recognition and segmentation. In the recognition task, the blank character - was introduced to fill the gap between two character predictions (e.g., *a*–*b*–*b*), but because our goal is to label each point in the input sequence with a specific character in the corresponding sentence, we must avoid unnecessary use of the blank character and instead predict actual characters (e.g., *aaabbbb*). The only case where the blank character is needed in segmentation is when a character is repeated in a sentence. To highlight the switch from the first *b* case to the second *b* case, we use the - (e.g., *aaabb-b*). This slight modification in

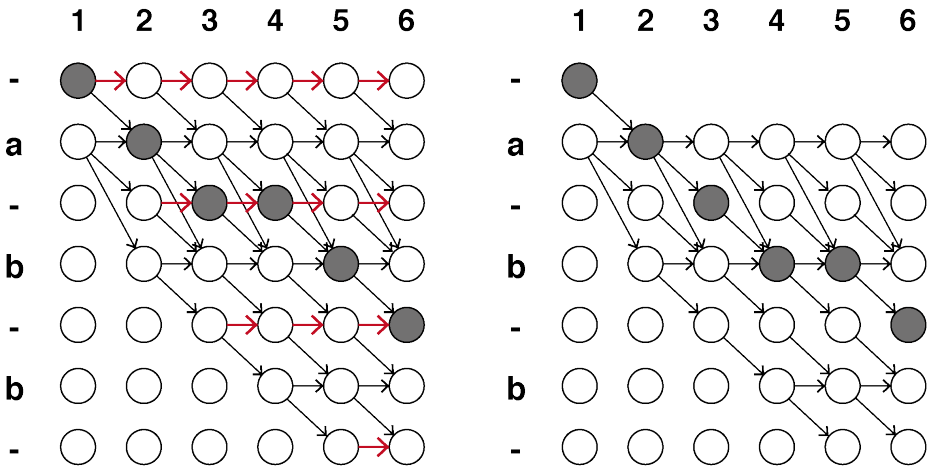


Fig. 12: Illustration of connections between temporal nodes. (Left) Original CTC connections. (Right) Our design of CTC connections. The connections between non-character nodes '-' are prohibited (red arrows in the original). The shaded nodes shows an example route for prediction.

CTC connections enables us to train our segmentation network in an unsupervised manner, automatically label sequential handwriting data with characters and identify *eoc* indices. Examples of segmentation are shown in Figure 13.

J Detailed Training Procedure

J.1 Ablation Study

In the main paper, we discussed three different ways to obtain $\mathbf{w}_{c_t}$: f_θ^{enc}, Method α, and Method β. As we compute losses for each type, we conducted a simple ablation study. First, removing $\mathscr{L}_\alpha$ from the total loss will take away the ability to generate handwriting samples from the mean writer-DSD $\overline{\mathbf{w}}$ from the model by construction. Similarly, excluding $\mathscr{L}_\beta$ will disallow our model to synthesize a new sample from saved writer-character-DSDs in the database D and only allow it to generate from the mean $\overline{\mathbf{w}}$. It is clear that we need both types of losses, $\mathscr{L}_\alpha$ and $\mathscr{L}_\beta$, to have the current model capabilities.

However, eliminating $\mathscr{L}_{f_\theta^{\text{enc}}}$, the loss term for a method that uses the original writer-character-DSD extracted from f_θ^{enc}, does not change model dynamics. Hence, we trained ablated models with modified loss functions that do not include: 1) any terms related to f_θ^{enc} (i.e., $\mathscr{L}_{f_\theta^{\text{enc}}}$, $\mathscr{L}_\alpha^{\mathbf{w}_{c_t}}$ and $\mathscr{L}_\beta^{\mathbf{w}_{c_t}}$), and 2) just $\mathscr{L}_{f_\theta^{\text{enc}}}$. Figure 14 shows the training curve for word-level location losses $\mathscr{L}^{loc}$. Removing $\mathscr{L}_{f_\theta^{enc}}$ had significant influence on Method β locational loss, $\mathscr{L}_\beta^{loc}$ (standard: -3.213 vs. ablated: -1.811 after 250K training steps).

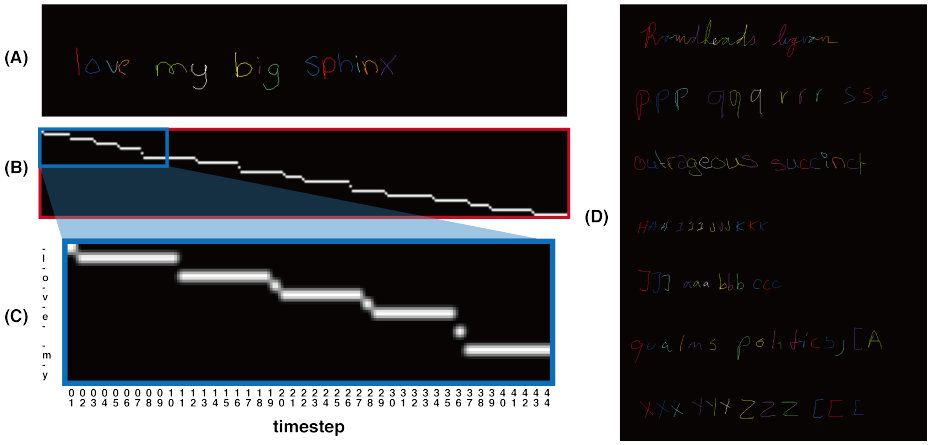


Fig. 13: Segmentation results. A) in handwriting image format. Different colors indicate different character segments. B) in CTC best route format. C) enlarged figure of the route path. D) More results.

From this result, we assume that $\mathscr{L}_{f_\theta^{\text{enc}}}$ works as a learning guideline for our model, and speeds up the training. We analyze that this is because having $\mathscr{L}_{f_\theta^{\text{enc}}}$ in our loss function encourages accurate learning for our decoder function f_θ^{dec}. In this setting, the function f_θ is indeed an autoencoder, and the decoder is trained to restore $\mathbf{x}$ from its encoded representation, writer-character-DSDs. This will increase the decoder performance, and as the decoder accuracy is maintained, the model can focus on learning the encoder problem, which is reconstruction of writer-character-DSDs by Method α and β.

The reconstruction losses, $\mathscr{L}_\alpha^{\mathbf{w}_{ct}}$ and $\mathscr{L}_\beta^{\mathbf{w}_{ct}}$, by contrast, did not affect the learning speed. We assume this can also be addressed by the same reason. Even if we constrained the reconstructed DSDs by Method α and β to minimize their differences with the original DSDs from f_θ^{enc}, those constraints will penalize the encoder more than they do for the decoder. To effectively train the decoder function, our model thus requires the loss term $\mathscr{L}_{f_\theta^{\text{enc}}}$.

J.2 Hyperparameters

To train our synthesis model, we use Adam [8] as our optimizer and set the learning rate to 0.001. We also clip the gradients in the range $[-10.0, 10.0]$ to enhance learning stability. We use 5 sentence-level samples (relevant word-level and character-level samples are included as well) for each batch in training. We use multi-stacked (3-layers) LSTMs for our recurrent layers.

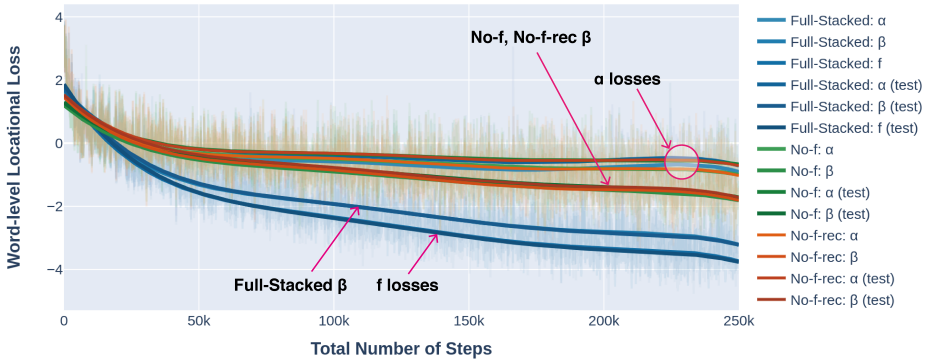


Fig. 14: Effects of $\mathscr{L}_{f_\theta^{\text{enc}}}$ on training word-level location loss $\mathscr{L}_{loc}^{word}$. Transparent lines show the actual data points, and solid lines show smoothed training curves. *Full-Stacked model* is trained with the full loss, while *No-f* model's loss function does not include $\mathscr{L}_{f_\theta^{\text{enc}}}$. Further, *No-f-rec* model does not have $\mathscr{L}_{f_\theta^{\text{enc}}}$, $\mathscr{L}_{\alpha}^{\mathbf{w}_{c_t}}$, $\mathscr{L}_{\beta}^{\mathbf{w}_{c_t}}$ terms in its loss function. *Test* data is 20 held-out writers.

K Dataset Specification and Collection Methodology

Our dataset considers the 86 characters: a space character ' ', and the following 85 characters:

```
0123456789
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
!?"'*+-=:;,.<>\/[]()#$%&
```

We collected handwriting samples from 170 writers using Amazon Mechanical Turk. An example screen of our data collection website is shown in Figure 15. Writing arbitrary words is laborious, and so we set a data-collection time limit of 60 minutes. Given this, it was necessary to select a subset of English words for our data collection.

K.1 Defining Target Words and Sentences

We analyzed the Gutenberg Dataset [9], which is a large corpus of 3,036 English books. These documents use 99 characters in total, including alphabetical, numerical, and special characters. In total, 5,831 character pairs appear in the dataset, while theoretically there are 9,702 possible character pairs ($99 \times 99 - 99$). By counting the number of occurrences of each character pair, we constructed an ordered list of character pairs that is then used to score 2,158,445 distinctive words within the corpus.

The first word to be selected from the corpus was *therefore*, which includes the two most frequently used character pairs *th* and *he*. In fact, the character

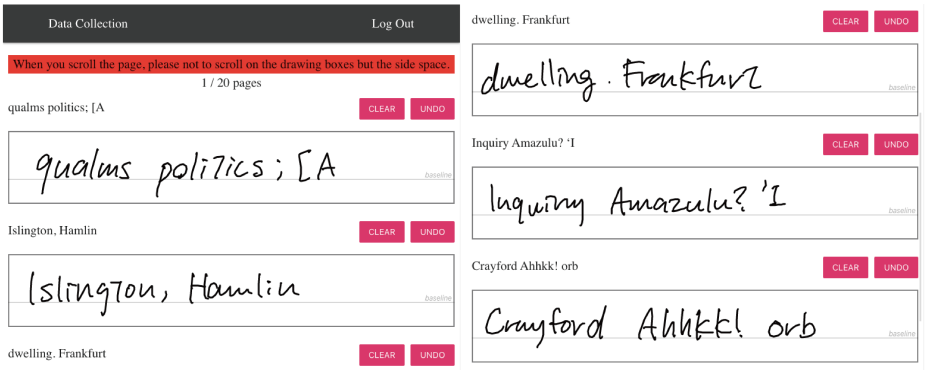


Fig. 15: Example screen of our data collection website. Each drawing box is 750 pixels × 120 pixels, and we provide a baseline at 80 pixels from the top.

pairs within *therefore* appear so frequently that they altogether cover 13.5% of all character pair occurrences.

After adding *therefore* to the list of words for experiments, we then add additional words iteratively: we re-calculate scores for all other words with updated scores of character pairs (i.e., after adding *therefore*, the pairs *th* and *he* will not have high scores in future iterations). This process was repeated until the words in the list exceeded 99% coverage of all character pair occurrences.

Then, we constructed sentences from these high-scored words. Each sentence was less than 24 characters length to meet a space constraint due to our experiment setup. We asked tablet owners to write the prompted sentences using their stylus within the bounding box (750 pixels × 120 pixels), and 24 characters was the maximum number of characters that could reasonably fit into the region.

We also added several pangram sentences as well as repeated characters sentences (e.g., *aaa bbb ccc ddd*), and that led to our basic list of 192 sentences. These sentences use 86 unique characters, instead of 99 available characters, due to our decision to ignore rarely used special characters. They also use $1,182$ distinct character pairs which cover 99.5% of all character pair occurrences ($1,158,051,103/1,164,429,941$). The remaining pairs could have been ignored, yet because that 0.5% was still large—$6,378,838$ occurrences by $4,649$ character pairs—we decided to create a list of extra words with less frequently used character pairs, distribute them to 170 writers. Thus, each writer creates some rarer data that varies for each writer, in addition to their basic 192 sentences. As a result, we achieve 99.9% coverage with $3,894$ character pairs.

K.2 Writer Behavior

Handwriting dataset collection is complex for various reasons, and in general creating a clean dataset without heuristic or manual cleaning is difficult. In our collection process, sometimes a writer would realizes that s/he missed certain

characters in the sentence after finishing the line, and so would go back to the location to add new strokes. These 'late' character additions are accidental rather than intentional. In contrast, conventional online handwriting recognition literature defines *delayed strokes*, where in cursive writing the horizontal bar of *t* and *f*, or the dot of *i* and *j*, are often added after a writer finished the current word. To distinguish between these two cases of late characters and delayed strokes, we disregard the temporal order of each stroke in a sample and reorder them from left to right *if* the leftmost point in a stroke is to the right of the rightmost point in another stroke. In this way, accidental omissions are removed.

Further, although we strongly advised participants to erase previous lines if they made mistakes, most participants either ignored this and left mistakes in, or scribbled over those regions to block them out. Writers also missed characters from the prompted sentences, and not a single participant (out of 170 writers) succeeded in near-perfect writing of 192 sentences. As our segmentation network (Sec. I) assumes that each drawing sample is labeled with the accurate character sequence, missed characters can directly affect the performance of segmentation. Hence, we manually corrected these instances throughout our dataset.

References

1. Aksan, E., Pece, F., Hilliges, O.: DeepWriting: Making Digital Ink Editable via Deep Generative Modeling. In: SIGCHI Conference on Human Factors in Computing Systems. CHI '18, ACM, New York, NY, USA (2018)
2. Bishop, C.M.: Mixture density networks (1994)
3. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd International Conference on Machine Learning. p. 369–376. ICML '06, Association for Computing Machinery, New York, NY, USA (2006). https://doi.org/10.1145/1143844.1143891, `https://doi.org/10.1145/1143844.1143891`
4. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A novel connectionist system for unconstrained handwriting recognition. IEEE transactions on pattern analysis and machine intelligence **31**(5), 855–868 (2008)
5. Huang, X., Belongie, S.: Arbitrary style transfer in real-time with adaptive instance normalization. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (Oct 2017)
6. Jaeger, S., Manke, S., Reichert, J., Waibel, A.: Online handwriting recognition: the npen++ recognizer. International Journal on Document Analysis and Recognition **3**(3), 169–180 (2001)
7. Keysers, D., Deselaers, T., Rowley, H.A., Wang, L.L., Carbune, V.: Multi-language online handwriting recognition. IEEE transactions on pattern analysis and machine intelligence **39**(6), 1180–1194 (2016)
8. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2014)
9. Lahiri, S.: Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In: Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics. pp. 96–105. Association for Computational Linguistics, Gothenburg, Sweden (April 2014), `http://www.aclweb.org/anthology/E14-3011`