ScribeTokens: A Fixed-Vocabulary Tokenization of Digital Ink

Douglass Wang
Independent Researcher
douglasswng@gmail.com
https://github.com/douglasswng/scribe-tokens

Abstract

Digital ink—the coordinate stream captured from stylus or touch input—lacks a unified representation. Continuous vector representations produce long sequences and suffer from training instability, while existing token representations require large vocabularies, face out-of-vocabulary issues, and underperform vectors on recognition. We propose ScribeTokens, a tokenization that decomposes pen movement into unit pixel steps. Together with two pen-state tokens, this fixed 10-token base vocabulary suffices to represent any digital ink and enables aggressive BPE compression. On handwritten text generation, ScribeTokens dramatically outperforms vectors on IAM (17.33% vs. 70.29% CER), showing tokens are far more effective for generation. On handwritten text recognition, ScribeTokens is the only token representation to outperform vectors on DeepWriting. We further introduce next-ink-token prediction as a self-supervised pretraining task, which consistently improves all token-based models in data-limited regimes and accelerates convergence by up to 21×. With pretraining, ScribeTokens achieves the best recognition results among all methods compared on both datasets (8.27% CER on IAM, 9.83% on DeepWriting).

1 Introduction

Digital ink (the coordinate stream captured from stylus or touch input) is a structured sequential modality underlying applications from handwriting recognition [1] and mathematical expression parsing [2] to sketch synthesis [3] and handwriting generation [4]. Unlike offline handwriting images, digital ink preserves temporal writing dynamics as ordered sequences of strokes, each comprising a sequence of xy-coordinates. How this two-level structure is flattened into a single sequence for modeling directly affects both sequence length and training stability—and, in turn, inference speed and task performance.

Despite growing interest in digital ink modeling, existing representations each carry significant limitations. Vector representations encode ink as sequences of continuous coordinates with binary pen-up/down flags [1, 3, 4]. While widely used, they produce long sequences and require careful design choices for input normalization and sequence handling. For generation, they rely on mixture density networks that suffer from training instability and hard-to-interpret likelihood values [5–7]. Token representations have emerged as an alternative [8, 9], enabling compression via Byte-Pair Encoding (BPE) [10] and more stable training through cross-entropy loss. However, existing tokenization methods carry their own drawbacks: out-of-vocabulary (OOV) issues, large base vocabularies that scale with canvas resolution, or fragile decoding syntax. As we show, they also underperform vector representations on recognition benchmarks.

We propose *ScribeTokens*, a token representation that eliminates OOV issues entirely using a fixed base vocabulary of just 10 tokens with robust decoding syntax. Our key insight is to decompose pen

Preprint.

Figure 1: ScribeTokens representation of a handwritten sentence. Pen strokes are decomposed into unit directional steps via Bresenham's algorithm, then compressed with BPE. Each color denotes a distinct BPE token; faint colors indicate pen-in-air movement between strokes. The zoom shows the sequence of arrows making up an example token.

movements into unit steps between adjacent pixels via Bresenham's line algorithm [11], encoding each step as one of eight directional tokens inspired by Freeman chain codes [12], with two additional pen state tokens (up/down). Each token carries an intuitive physical interpretation, and because it encodes paths rather than points, it yields a canonical representation invariant to sampling rate or point density. BPE applied over this base vocabulary yields high compression while preserving the OOV-free property, since any unseen pattern can always be decomposed into base tokens (Figure 1).

We evaluate ScribeTokens on handwritten text recognition (HTR) and handwritten text generation (HTG). On HTG, token representations substantially outperform vectors—ScribeTokens achieves 17.33% character error rate (CER) compared to 70.29% for vectors on the IAM dataset [13]—demonstrating that vector representations struggle with generation. On HTR, ScribeTokens is the only token representation to outperform vectors on DeepWriting [14]. We further show that next-ink-token prediction [15, 16] is an effective self-supervised pretraining strategy, accelerating convergence by up to 21×. It consistently improves every token representation on HTR, with vectors being the notable exception. ScribeTokens with pretraining achieves the best HTR results on both datasets and the best HTG results on IAM, where data is scarce.

Contributions. (1) We propose ScribeTokens, a canonical, OOV-free tokenization of digital ink with a fixed 10-token base vocabulary and robust decoding, enabling aggressive BPE compression.[1] ScribeTokens is the only token representation to outperform vectors on recognition, and the best-performing representation on generation where data is scarce. (2) We establish next-ink-token prediction as an effective self-supervised pretraining strategy, yielding up to 21× faster convergence and consistent gains, particularly in data-limited settings. ScribeTokens with pretraining achieves the best recognition results on both datasets and the best generation results on IAM, where data is scarce.

2 Related Work

2.1 Vector Representations

Vector representations model digital ink as a sequence of continuous vectors [3, 4, 18, 19]. For generation, the next xy-coordinate is modeled as a mixture of 2-dimensional Gaussians, with discrete events (such as pen-up) modeled by a categorical distribution [3, 4, 18].

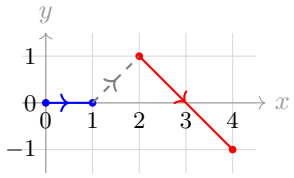
Point-3 [4] encodes each point as $(\Delta x, \Delta y, p)$, where $\Delta x, \Delta y$ are offset coordinates from the previous point and p is a binary pen-up indicator. *Point-5* [3] extends this to $(\Delta x, \Delta y, p_1, p_2, p_3)$, using a one-hot pen state across three mutually exclusive states: pen down, pen up (end of stroke), and end of sequence. Table 1 illustrates each representation for a two-stroke example.

Vector representations, while natural, suffer from several drawbacks: (1) the lack of compression often necessitates truncating long sequences [4],[2] (2) many design choices must be made for input normalization [4, 14, 18, 20] and sequence initiation/termination [4, 18], and (3) mixture density networks used for generation require additional hyperparameters, suffer from numerical instability [5] and mode collapse [6], and produce negative log-likelihood values [7] that are difficult to interpret.

[1] An accompanying Hugging Face [17] library for fast tokenization and tokenizer training is available at `https://github.com/douglasswng/tokink`.

[2] While Carbune et al. [19] present a vector representation achieving significant compression, their approach is not applicable to generation tasks.

Table 1: Digital ink representations for a two-stroke example. Colors distinguish the first stroke, second stroke, and pen-in-air movement. *Point-3* encodes offsets with a binary pen flag; *Point-5* extends this with a one-hot pen state that additionally signals end of sequence; *AbsTokens* and *RelTokens* discretize absolute and relative coordinates into tokens; *TextTokens* serialize offsets as character sequences; *ScribeTokens* (Ours) decompose strokes into unit directional steps via Bresenham's algorithm.

Representation	Value
Digital Ink	$((0,0),(1,0))$, $((2,1),(4,-1))$
Visualization	
Point-3	$(1,0,1)$, $(1,1,0)$, $(2,-2,1)$
Point-5	$(1,0,1,0,0)$, $(1,1,0,1,0)$, $(2,-2,0,0,1)$
AbsTokens	`[(0, 0)]`, `[(1, 0)]`, `[UP]`, `[(2, 1)]`, `[(4, -1)]`, `[UP]`
RelTokens	`[(1, 0)]`, `[UP]`, `[(1, 1)]`, `[(2, -2)]`, `[UP]`
TextTokens	`[1]`, `[␣]`, `[0]`, `[UP]`, `[1]`, `[␣]`, `[1]`, `[␣]`, `[2]`, `[␣]`, `[-]`, `[2]`, `[UP]`
ScribeTokens (Ours)	`[DOWN]`, `[→]`, `[UP]`, `[↗]`, `[DOWN]`, `[↘]`, `[↘]`, `[UP]`

2.2 Token Representations

Token representations treat digital ink as a sequence of discrete tokens, which addresses the challenges faced by vector representations: (1) merging algorithms such as BPE effectively reduce sequence lengths, (2) `[START]` and `[END]` tokens eliminate design choices for sequence initiation and termination, and (3) cross-entropy loss does not introduce additional hyperparameters and allows for stable training with interpretable loss values [21].

AbsTokens [9] treats each pixel coordinate as a token, with `[UP]` for pen-up. *RelTokens* [9] is similar but uses relative offsets $(\Delta x, \Delta y)$ as tokens. *TextTokens* [8] serializes each offset as its decimal string (digits, minus signs, and spaces).

Existing token representations, however, have notable drawbacks. AbsTokens and RelTokens suffer from out-of-vocabulary (OOV) issues, mapping unseen coordinates to `[UNKNOWN]`, and require large base vocabularies that scale with canvas resolution. TextTokens avoids both problems but produces longer sequences with a fragile syntax: an autoregressive model can emit malformed sequences that do not parse into valid coordinate pairs.

2.3 Pretraining

Next-token prediction (NTP) on large unlabelled corpora followed by supervised fine-tuning has become the dominant paradigm in natural language processing [15, 16, 22, 23] and has been extended to speech [24] and music [25], but NTP pretraining directly on digital ink remains largely unexplored. Fadeeva et al. [8] leverage pretrained vision-language models to achieve state-of-the-art ink recognition on the DeepWriting dataset, but the underlying language pretraining is performed on generic text corpora rather than on ink data. In this work, we investigate next-ink-token prediction as a self-supervised pretraining task.

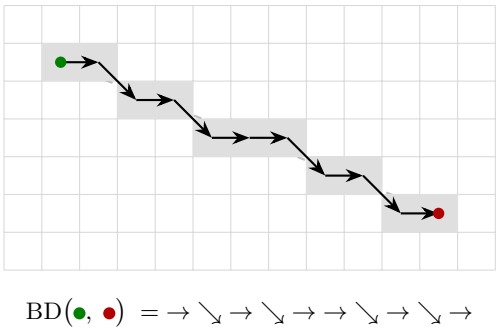


Figure 2: Bresenham Decomposition of a line segment between two grid points (start ●, end ●). The segment is rasterized into adjacent grid cells via Bresenham's algorithm, then encoded as a sequence of Freeman chain code directions.

3 Methods

3.1 Preprocessing

We define digital ink $\mathcal{I} = (S_j)_{j=1}^{M}$ as a sequence of M strokes, where each stroke $S_j = (p_i^{(j)})_{i=1}^{n_j}$ is an ordered sequence of continuous xy-coordinates. Since token representations are intrinsically discrete, we quantize the coordinates to a grid. More aggressive quantization decreases sparsity of the training signal and improves BPE compression rates, at the cost of increased staircase artifacts in the reconstructed ink.

Following Ribeiro et al. [9], we round each coordinate to the nearest point on a uniform grid with spacing $\delta > 0$:

$$(x_i, y_i) \mapsto \left(\text{round}\left(\frac{x_i}{\delta}\right),\ \text{round}\left(\frac{y_i}{\delta}\right)\right).$$

This produces an *integer ink* in which all coordinates are integers. All subsequent methods operate on integer inks.

3.2 Bresenham Decomposition

The core idea behind ScribeTokens is to represent pen strokes as sequences of unit directional steps on a discrete grid. This is achieved by combining two classical algorithms: Freeman chain codes [12] for directional encoding and Bresenham's line algorithm [11] for integer rasterization.

A Freeman chain code encodes a path on a discrete grid as a sequence of unit steps in eight directions—the four cardinal ($\rightarrow, \uparrow, \leftarrow, \downarrow$) and four diagonal ($\nearrow, \nwarrow, \swarrow, \searrow$)—making it a lossless encoding for any path between adjacent pixels. However, consecutive points in an integer ink are generally not adjacent. To bridge non-adjacent points, we rasterize the straight-line segment between them using Bresenham's line algorithm, which computes the optimal sequence of adjacent grid cells.

Given two integer points—possibly non-adjacent—we first rasterize the segment between them using Bresenham's algorithm, then encode transitions between consecutive rasterized pixels as Freeman chain code directions. We call this composition *Bresenham Decomposition* (BD). The result is a deterministic sequence of directional tokens uniquely determined by the two endpoints. Figure 2 illustrates this process.

3.3 ScribeTokens

To tokenize an integer ink $\mathcal{I} = (S_j)_{j=1}^{M}$, we combine the eight direction tokens with two pen-state tokens: `[DOWN]` (pen touches surface) and `[UP]` (pen lifts off), forming a fixed vocabulary of 10 base tokens. Each stroke is delimited by `[DOWN]` and `[UP]`, and consecutive points—both within strokes and during pen-in-air transitions—are encoded via Bresenham Decomposition. Algorithm 1 details the full procedure.

Algorithm 1 ScribeTokens tokenization

Require: Integer ink $\mathcal{I} = (S_j)_{j=1}^{M}$, each $S_j = (p_i^{(j)})_{i=1}^{n_j}$ a sequence of integer coordinates.
Ensure: Token sequence $T = (t_i)_{i=1}^{n}$

$T \leftarrow ()$
for $j = 1$ to M **do**
  // Begin stroke j (pen touches surface)
  Append `[DOWN]` to T
  for $i = 1$ to $n_j - 1$ **do**
    Append tokens from $\mathrm{BD}(p_i^{(j)},\ p_{i+1}^{(j)})$ to T {within-stroke movement}
  end for
  // End stroke j (pen lifts off surface)
  Append `[UP]` to T
  // Movement to next stroke (pen in air)
  if $j < M$ **then**
    Append tokens from $\mathrm{BD}(p_{n_j}^{(j)},\ p_1^{(j+1)})$ to T {between-stroke movement}
  end if
end for
return T

Sampling invariance. An important property of ScribeTokens is sampling invariance: digital inks that rasterize identically on a discrete grid produce identical token sequences, regardless of differences in sampling rate or point density that would yield distinct sequences under other representations.

Compression. While the base vocabulary is compact, raw ScribeTokens sequences can be long due to the pixel-level granularity of the decomposition. We apply BPE to compress sequences, with merges restricted to direction tokens only—pen-state tokens `[UP]` and `[DOWN]` are never merged—ensuring that stroke boundaries remain explicit. Since any merged token can always be decomposed back into its constituent base direction tokens, the representation remains OOV-free by construction.

Detokenization. For generation tasks, the predicted token sequence must be detokenized back to ink coordinates. Starting from an origin, each directional token specifies a unit step, and pen-state tokens delimit stroke boundaries. The recovered integer coordinates are scaled by δ to return to the original coordinate space, and Savitzky–Golay smoothing [26] is applied to mitigate staircase artifacts from the grid discretization (see Appendix A.1).

3.4 Task Formulation

We formulate all tasks under a unified prompt-completion framework. Given a prompt sequence $\mathbf{x}$ and a completion sequence $\mathbf{y} = (y_1, \ldots, y_T)$, a causal model is trained to maximize the conditional log-likelihood:

$$\log p(\mathbf{y} \mid \mathbf{x}) = \sum_{t=1}^{T} \log p(y_t \mid \mathbf{x},\ y_{<t}).$$

At inference, $\mathbf{y}$ is decoded autoregressively from a beginning-of-sequence state until an end-of-sequence condition is reached.

Let $\mathbf{s}$ denote the sequence encoding of a digital ink $\mathcal{I}$ under the chosen representation, and let $\mathbf{c}$ denote its text character sequence. The three tasks considered in this work are special cases of this framework, differing only in what constitutes $\mathbf{x}$ and $\mathbf{y}$:

- NTP: $\mathbf{x} = \varnothing$, $\mathbf{y} = \mathbf{s}$ — unconditional ink generation as a self-supervised pretraining objective.
- HTR: $\mathbf{x} = \mathbf{s}$, $\mathbf{y} = \mathbf{c}$ — the model reads ink and produces a text transcription $\mathbf{c}$.
- HTG: $\mathbf{x} = \mathbf{c}$, $\mathbf{y} = \mathbf{s}$ — the model generates ink conditioned on a text prompt.

Training loss. The choice of loss depends on the completion modality. When $\mathbf{y}$ consists of discrete tokens—text for HTR, or a token representation of ink for NTP and HTG—we use standard cross-entropy loss. When $\mathbf{y}$ uses a vector representation of ink, coordinates are modeled via a mixture

density network (MDN) and pen states via a categorical distribution, with the total loss being the sum of the negative log-likelihood for coordinates and cross-entropy for pen states [4].

4 Experiments

4.1 Datasets

IAM. The IAM On-Line Handwriting Database (IAM-OnDB) [13] is a widely adopted benchmark for digital ink analysis, containing labeled text lines from 221 writers. We use the standard writer-disjoint split, which—after removing corrupt samples—yields 5,042 train, 2,264 validation, and 3,541 test text lines. Because samples are full text lines, sequences are long, and the dataset is relatively small—making IAM a challenging setting for evaluating long-range modeling and data efficiency.

DeepWriting. The DeepWriting dataset [14] is derived from IAM-OnDB by filtering low-quality writers and segmenting text lines into individual words. We use only the IAM-derived portion, excluding supplementary collections that differ substantially in ink scale. As no standardized split exists, we use a random 80/10/10 train/validation/test partition without writer-disjoint constraints, yielding 36,912, 4,614, and 4,614 samples respectively; results on DeepWriting therefore reflect interpolation within observed writing styles. Compared to IAM, DeepWriting offers shorter sequences and more training data, providing a complementary regime for evaluation.

4.2 Tokenization Analysis

We train BPE tokenizers for AbsTokens, RelTokens, TextTokens, and ScribeTokens on the IAM training set across different quantization parameters δ. We use IAM rather than DeepWriting because its line-level samples contain long-range pen movements—such as spaces between words—that are absent from word-level data, yielding tokenizers that generalize better to diverse digital ink.

Compression and OOV rates are evaluated on the IAM validation set across target vocabulary sizes. Some combinations of δ and target vocabulary size are absent for AbsTokens and RelTokens, as the number of unique base tokens already exceeds the vocabulary budget, leaving no capacity for BPE merges.

Compression. Figure 3 shows average compression rates across target vocabulary sizes and quantization parameters. ScribeTokens consistently achieves the highest compression across all settings, except at very fine quantization ($\delta \in \{1, 2\}$).

OOV. Figure 4 shows average OOV rates across the same settings. ScribeTokens and TextTokens are OOV-free by construction. AbsTokens exhibits OOV rates of 0.15–0.3%, while RelTokens ranges from approximately 0.1% up to over 1.2%. These rates are non-trivial given that typical ink samples contain on the order of 1,000 points.

Quantization artifacts. Beyond tokenization properties, δ also governs the fidelity of the reconstructed ink. To evaluate this qualitatively, we quantize IAM validation samples at different values of δ and apply Savitzky–Golay post-processing (Appendix A.1). Figure 5 (Appendix) shows an example: at $\delta \leq 8$, original and reconstructed inks are visually indistinguishable; artifacts become noticeable around $\delta = 16$ and progressively degrade, with ink becoming unrecognizable at $\delta \geq 128$.

4.3 Downstream Setup

Based on considerations of compression rate, OOV rate, and reconstruction quality, we fix $\delta = 8$ and a target vocabulary size of 32,000 for all downstream experiments. Four representations are compared: Point-5, RelTokens, TextTokens, and ScribeTokens. Point-3 is excluded because it lacks an end-of-sequence pen state and therefore cannot terminate generation, and AbsTokens because its base vocabulary already exceeds the vocabulary budget at $\delta = 8$, leaving no capacity for BPE merges. The same model architecture and training is used for all representations and tasks, ensuring that performance differences are driven by the choice of representation.

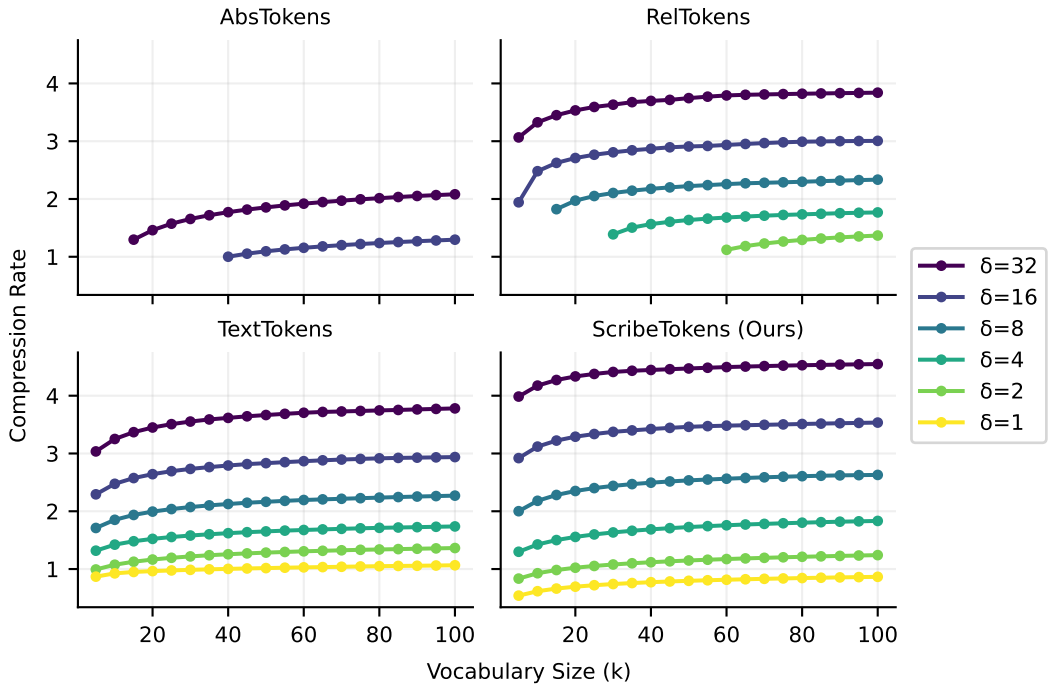


Figure 3: Average compression rates (↑) of BPE-based digital ink representations on the IAM validation set, across target vocabulary sizes and quantization parameters δ. ScribeTokens consistently achieves the highest compression across nearly all settings.

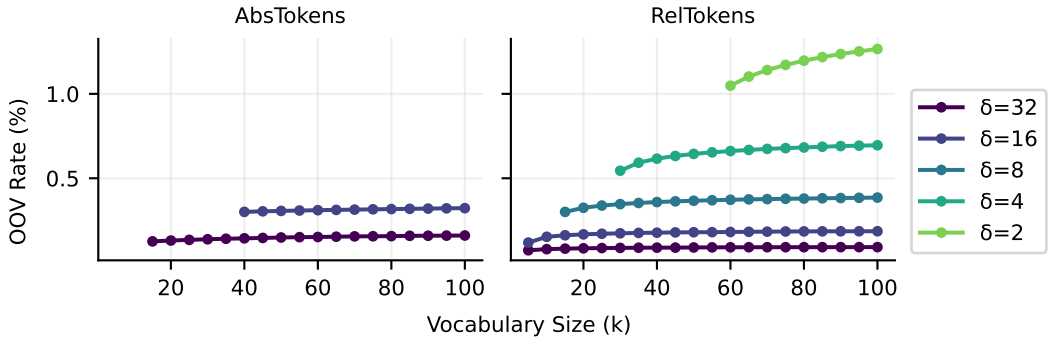


Figure 4: Average out-of-vocabulary (OOV) rates (↓) of BPE-based digital ink representations on the IAM validation set, across target vocabulary sizes and quantization parameters δ. ScribeTokens and TextTokens are OOV-free by construction, while AbsTokens and RelTokens exhibit non-zero OOV rates as their coordinate-based vocabularies inevitably encounter unseen values at test time.

Architecture and training. Our model is a 12-layer decoder-only Transformer [27] following the LLaMA design [28], with approximately 34M parameters for token-based models. Point-5, which replaces the large embedding table with lightweight linear projections, has approximately 21M parameters (see Appendix A.2 for details). Models are trained with AdamW [29] for up to 200 epochs with early stopping, using stochastic geometric augmentations on ink inputs. Full architecture and training details are provided in Appendix A.3.

4.4 Handwritten Text Recognition

The task is to predict the text transcript from an input ink sequence. Models denoted +PT are first pretrained with next-ink-token prediction on the training set using only ink data, then fine-tuned for

Table 2: Handwritten text recognition on IAM and DeepWriting, measured by CER (↓) and accuracy (↑) under autoregressive decoding (temperature 0). +PT denotes initialization from next-ink-token prediction pretraining prior to task fine-tuning. Deltas show the effect of pretraining: teal indicates improvement, red degradation. Best results are **bolded**.

		IAM		DeepWriting	
Method		CER (%) ↓	Acc (%) ↑	CER (%) ↓	Acc (%) ↑
Point-5		9.43	31.38	10.94	81.97
	+PT	13.63 (+4.19)	18.92 (-12.45)	10.25 (-0.69)	83.16 (+1.19)
RelTokens		12.69	25.11	11.22	81.60
	+PT	9.16 (-3.53)	30.75 (+5.65)	10.49 (-0.73)	82.73 (+1.13)
TextTokens		82.00	0.00	11.65	81.17
	+PT	9.54 (-72.46)	29.57 (+29.57)	10.07 (-1.58)	83.33 (+2.17)
ScribeTokens (Ours)		13.15	22.79	10.75	82.29
	+PT	**8.27** (-4.87)	**32.93** (+10.14)	**9.83** (-0.92)	**83.40** (+1.11)

recognition. Table 2 reports character error rate (CER) and exact-match accuracy under autoregressive decoding (temperature 0).

Without pretraining. On DeepWriting, ScribeTokens (10.75% CER, 82.29% accuracy) outperforms Point-5 (10.94%, 81.97%), making it the only token-based model to do so. On IAM, Point-5 achieves the best results (9.43% CER, 31.38% accuracy) among all methods trained from scratch, outperforming every token-based representation. TextTokens fails entirely on IAM without pretraining (82.00% CER, 0% accuracy); a detailed analysis is provided in Appendix A.5.

With pretraining. Pretraining (+PT) consistently improves all token-based models on both datasets, with especially large gains on IAM where training data is scarce: ScribeTokens improves by 4.87 points in CER and 10.14 in accuracy, and TextTokens recovers from complete failure (82.00% to 9.54% CER). Point-5 is the notable exception: pretraining *degrades* its IAM performance (CER rises from 9.43% to 13.63%, accuracy drops from 31.38% to 18.92%); we hypothesize why in Appendix A.6. Overall, ScribeTokens + PT achieves the best performance across both datasets (8.27% CER on IAM, 9.83% on DeepWriting). For reference, Graves et al. [1] report 11.5% CER on IAM with a BLSTM, CTC, and no language model; ScribeTokens + PT surpasses this with a decoder-only Transformer alone. State-of-the-art systems reach below 5% CER [19] using bidirectional encoders, CTC decoding, and language models—choices orthogonal to both ink representation and pretraining strategy.

4.5 Handwritten Text Generation

The task is to produce an ink sequence conditioned on a text prompt. Table 3 reports CER and exact-match accuracy under autoregressive decoding (temperature 1), where generated inks are scored against the input text by our best-performing recognizer—ScribeTokens + PT—trained exclusively on real handwriting. All outputs are first detokenized to raw ink coordinates and post-processed (Appendix A.1). Sampling invariance (Section 3.3) then ensures recognition depends only on the rasterized path. Using a ScribeTokens-trained recognizer therefore does not bias evaluation toward ScribeTokens-generated ink: indeed, TextTokens matches ScribeTokens' CER on IAM with pretraining (Table 3). Qualitative examples of generated inks are provided in Appendix A.7.

Without pretraining. On IAM, where sequences are long, most methods struggle. Point-5 nearly fails entirely (70.29% CER, 0.03% accuracy), likely because the lack of compression yields excessively long sequences that are difficult to model autoregressively. ScribeTokens (17.33% CER) and RelTokens (25.89%) also underperform, while TextTokens achieves the best CER among all methods (13.65%). On DeepWriting, all methods perform reasonably, with RelTokens achieving the lowest CER (12.75%) and highest accuracy (73.04%). RelTokens' weak performance on sentence-level IAM but strong showing on word-level DeepWriting is consistent with its OOV susceptibility: sentence-

Table 3: Handwritten text generation on IAM and DeepWriting, measured by CER (↓) and accuracy (↑) under autoregressive decoding (temperature 1). +PT denotes initialization from next-ink-token prediction pretraining prior to task fine-tuning. Generated inks are post-processed (Appendix A.1) and evaluated by the best HTR model (ScribeTokens + PT; Table 2). Deltas show the effect of pretraining: teal indicates improvement, red degradation. Best results are **bolded**.

		IAM		DeepWriting	
Method		CER (%) ↓	Acc (%) ↑	CER (%) ↓	Acc (%) ↑
Point-5		70.29	0.03	14.36	71.09
	+PT	16.83 (-53.46)	15.96 (+15.93)	26.84 (+12.48)	51.24 (-19.85)
RelTokens		25.89	4.43	**12.75**	**73.04**
	+PT	11.93 (-13.96)	20.33 (+15.90)	12.98 (+0.23)	72.48 (-0.56)
TextTokens		13.65	16.58	15.32	70.85
	+PT	10.45 (-3.20)	22.71 (+6.13)	14.12 (-1.20)	72.61 (+1.76)
ScribeTokens (Ours)		17.33	12.93	15.47	72.78
	+PT	**10.45** (-6.88)	**23.84** (+10.90)	16.02 (+0.55)	68.23 (-4.55)

level samples contain large inter-word jumps that produce rare displacements unlikely to appear in the vocabulary.

With pretraining. On IAM, pretraining (+PT) dramatically improves all methods. The largest gain is for Point-5 (-53.46 points CER), though it still lags behind the token-based models. ScribeTokens and TextTokens both reach 10.45% CER, with ScribeTokens achieving the highest accuracy (23.84% vs. 22.71%). On DeepWriting, pretraining yields mixed results: it modestly improves TextTokens but degrades Point-5 severely (CER rises from 14.36% to 26.84%) and notably hurts ScribeTokens (-4.55 points accuracy), while RelTokens is largely unaffected. Overall, ScribeTokens + PT achieves the best results on IAM (10.45% CER, 23.84% accuracy); on DeepWriting, RelTokens without pretraining leads (12.75% CER, 73.04% accuracy).

5 Conclusion

We introduced ScribeTokens, a fixed-vocabulary tokenization of digital ink that decomposes pen strokes into unit directional steps via Bresenham's line algorithm. The resulting 10-token base vocabulary is sufficient to encode any ink, eliminates out-of-vocabulary issues by construction, admits robust decoding, and enables aggressive BPE compression. ScribeTokens is the only token representation to outperform vectors on recognition and the best-performing representation on generation where data is scarce, demonstrating that a canonical discrete representation can retain the training stability and compression advantages of tokens without sacrificing task performance.

We further established next-ink-token prediction as an effective self-supervised pretraining strategy. Pretraining consistently improved every token-based model on recognition, accelerated convergence by up to 21× (Appendix A.4), and was particularly effective in data-limited regimes. ScribeTokens with pretraining achieved the best recognition results on both datasets and the best generation results on IAM, where data is scarce. Attention analysis reveals that pretraining shifts the model toward relying on the ink signal rather than memorized text patterns (Appendix A.5). We attribute this to the cold-start problem: token embeddings are initially random, and pretraining forces the model to discover their spatial structured—a burden that continuous vectors do not face (Appendix A.6).

Limitations and future work. We evaluated only on English handwriting recognition and generation with a single architecture; generalization to other scripts, tasks, and model families remains to be verified. Additionally, next-ink-token prediction pretraining was performed on the same training sets rather than large unlabeled corpora. The techniques that advanced language modeling—scaling pretraining data, efficient attention for long sequences, and reinforcement learning from human feedback for generation quality—are directly applicable to digital ink. ScribeTokens provides the vocabulary; the language modeling playbook provides the roadmap.

Acknowledgments and Disclosure of Funding

The author thanks Frédéric Fillieux, Huidong Liang, and Zeli Wang for feedback on earlier drafts. The author also thanks his family for their support. This work was not supported by any external grants or organizations.

References

[1] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[2] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3:3–15, 2000.

[3] David Ha and Douglas Eck. A neural representation of sketch drawings. In *International Conference on Learning Representations*, 2018.

[4] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[5] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *International Conference on Robotics and Automation*, pages 2090–2096, 2019.

[6] Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7144–7153, 2019.

[7] Christopher M. Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, Birmingham, UK, 1994.

[8] Anastasiia Fadeeva, Philippe Schlattner, Andrii Maksai, Mark Collier, Efi Kokiopoulou, Jesse Berent, and Claudiu Musat. Representing online handwriting for recognition in large vision-language models. *arXiv preprint arXiv:2402.15307*, 2024.

[9] Leo Sampaio Ferraz Ribeiro, Tu Bui, John Collomosse, and Moacir Ponti. Sketchformer: Transformer-based representation for sketched structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14153–14162, 2020.

[10] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.

[11] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[12] Herbert Freeman. On the encoding of arbitrary geometric configurations. *IEEE Transactions on Electronic Computers*, EC-10(2):260–268, 1961.

[13] Marcus Liwicki and Horst Bunke. Iam-ondb-an on-line english sentence database acquired from handwritten text on a whiteboard. In *Eighth International Conference on Document Analysis and Recognition*, pages 956–961, 2005.

[14] Emre Aksan, Fabrizio Pece, and Otmar Hilliges. Deepwriting: Making digital ink editable via deep generative modeling. In *Proceedings of the 2018 CHI conference on human factors in computing systems*, pages 1–14, 2018.

[15] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

[16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[17] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

[18] Gang Dai, Yifan Zhang, Qingfeng Wang, Qing Du, Zhuliang Yu, Zhuoman Liu, and Shuangping Huang. Disentangling writer and character styles for handwriting generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5977–5986, 2023.

[19] Victor Carbune, Pedro Gonnet, Thomas Deselaers, Henry A Rowley, Alexander Daryin, Marcos Calvo, Li-Lun Wang, Daniel Keysers, Sandro Feuz, and Philippe Gervais. Fast multi-language lstm-based online handwriting recognition. *International Journal on Document Analysis and Recognition*, 23(2):89–102, 2020.

[20] Xu-Yao Zhang, Fei Yin, Yan-Ming Zhang, Cheng-Lin Liu, and Yoshua Bengio. Drawing and recognizing chinese characters with recurrent neural network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):849–862, 2017.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

[24] Zhihao Du, Qian Chen, Shiliang Zhang, Kai Hu, Heng Lu, Yexin Yang, Hangrui Hu, Siqi Zheng, Yue Gu, Ziyang Ma, et al. Cosyvoice: A scalable multilingual zero-shot text-to-speech synthesizer based on supervised semantic tokens. *arXiv preprint arXiv:2407.05407*, 2024.

[25] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

[26] Abraham Savitzky and Marcel JE Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[29] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[30] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[31] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[32] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[33] Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 4190–4197, 2020.

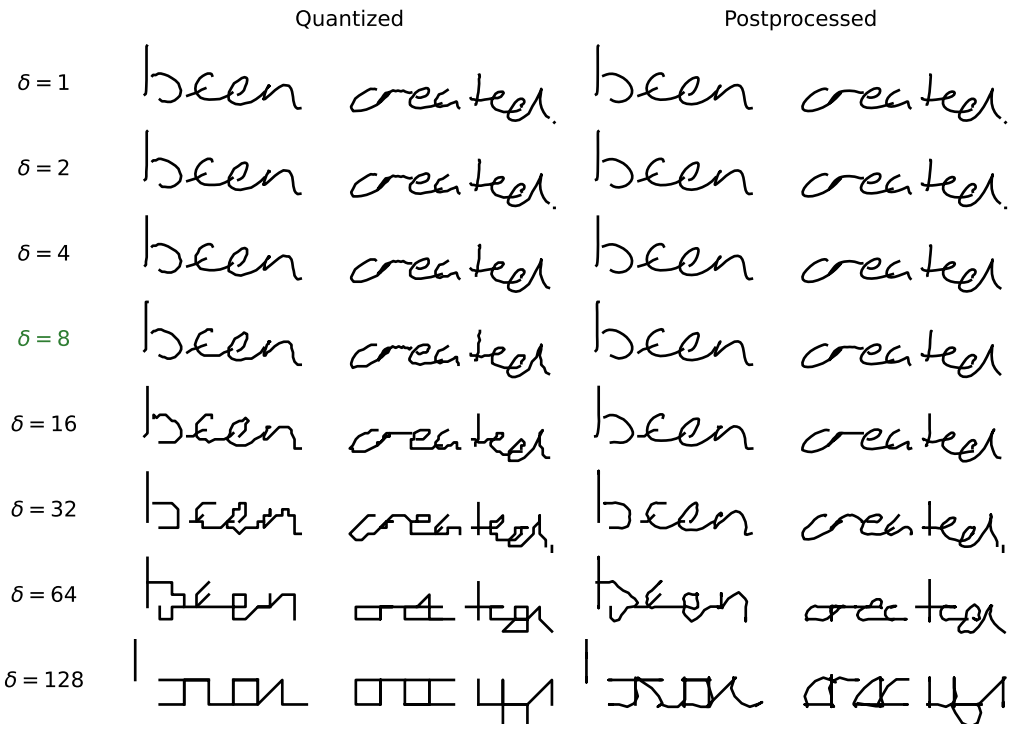


Figure 5: Effect of quantization parameter δ on reconstruction quality. Each row shows the same IAM sample quantized at a different δ, displayed both as raw quantized ink (left) and after Savitzky–Golay post-processing (right). Post-processed inks are visually indistinguishable from the original for $\delta \leq 8$; the row at $\delta = 8$ (highlighted in green) maximizes compression without sacrificing fidelity and is used in all downstream experiments.

A Appendix

A.1 Quantization Artifacts

Quantizing ink coordinates to a discrete grid introduces staircase artifacts in reconstructed strokes. Figure 5 visualizes the quantization artifacts across eight values of δ, as well as the effect of the proposed post-processing to mitigate them.

To recover smooth trajectories, we apply Savitzky–Golay filtering [26] independently to the x- and y-coordinate sequences of each stroke. We prefer Savitzky–Golay over Gaussian smoothing because its local polynomial fitting better preserves peaks and sharp transitions in the stroke geometry. We use a window size of $w = 7$ and polynomial order $k = 3$ for all experiments.

For ScribeTokens, Bresenham's line algorithm introduces many intermediate pixel-level points, producing denser strokes than the original ink. To counteract this, we downsample each generated stroke (retaining every d-th point) before applying the filter. The downsampling rate d depends on both the dataset and the choice of δ. Since generated coordinates are rescaled by δ, larger values of δ produce sparser strokes and require smaller d. With $\delta = 8$ on IAM [13] and DeepWriting [14], $d = 2$ works well.

A.2 Point-5 Model

The Point-5 representation [3] encodes each time step as a 5-dimensional vector $(\Delta x, \Delta y, p_1, p_2, p_3)$, which is projected to the model's hidden dimension via a learned linear layer. Generation is initiated with a fixed start vector $(0, 0, 0, 1, 0)$, corresponding to zero offset in the pen-up state.

The output head models continuous coordinates and discrete pen states separately. For coordinates, we use a mixture density network (MDN) with $K = 20$ mixture components. The hidden state

Table 4: Architecture and training hyperparameters shared across all representations and tasks.

Architecture	
Model type	Decoder-only Transformer (LLaMA-style)
Parameters	$\sim$34M (token-based) / $\sim$21M (Point-5)
Layers	12
Attention heads	6
Hidden dimension	384
Positional encoding	RoPE [30]
Activation	SwiGLU [31] (expansion factor $8/3$)
Normalization	Pre-norm RMSNorm [32]
Embedding tying	Yes (token-based models)
Dropout	0.2
Optimization	
Optimizer	AdamW
Learning rate	3×10^{-4} (constant)
Weight decay	0.1
Max gradient norm	1.0
Batch size	64 (DeepWriting) / 32 (IAM)
Mixed precision	bfloat16
Early stopping	200 epochs, patience 50
Data augmentation (each applied independently with $p = 0.5$)	
Random scaling	$\pm 30\%$
Shearing	± 0.5
Rotation	$\pm 5^\circ$
Gaussian jitter	$\sigma = 5$

is mapped to mixture weights $\pi_k \in (0, 1)$ (linear + softmax), means $\boldsymbol{\mu}_k \in \mathbb{R}^2$ (linear), standard deviations $\boldsymbol{\sigma}_k \in \mathbb{R}^2$ (linear + softplus), and correlation coefficients $\rho_k \in (-1, 1)$ (linear + tanh) for each component. Separately, the pen state (p_1, p_2, p_3) is mapped from the hidden state to a categorical distribution (linear + softmax).

The MDN output head requires several safeguards to train reliably. All ink coordinates are scaled down by a factor of 10 to keep offset magnitudes in a stable range. Standard deviations are clamped to a minimum of $\sigma_{\min} = 0.1$ to prevent the mixture components from collapsing to near-zero variance, and correlation coefficients are bounded to $|\rho_k| \leq 0.99$ to avoid degenerate covariance matrices. Without these measures, training diverges early. These constraints are unnecessary for token-based models, which use standard cross-entropy loss.

A.3 Architecture and Training

Table 4 lists all architecture and training hyperparameters used across experiments. Since RelTokens models are susceptible to out-of-vocabulary tokens at inference, we inject `[UNKNOWN]` tokens into its training data with probability 0.4%, matched to the empirical OOV rate on the validation set, so the model learns to handle unseen tokens gracefully. All models are implemented in PyTorch. All experiments are run on a single NVIDIA GH200 GPU; total compute across all runs is approximately 85 GPU-hours.

A.4 Pretraining Speedup

Beyond final task performance, next-ink-token prediction pretraining also accelerates downstream convergence. Tables 5 and 6 report, for each representation, the number of pretraining epochs run, the number of epochs to converge without pretraining, and how many fine-tuning epochs the pretrained model needs to reach the same converged validation loss.

Table 5: Convergence speedup from pretraining for HTR. *PT Ep.*: pretraining epochs run. *No PT*: epochs to converge without pretraining. *+PT*: fine-tuning epochs to reach the same converged loss. *Spd.*: speedup ratio (No PT / +PT). "–" indicates the pretrained model never reached the baseline loss. Best speedups are **bolded**.

	IAM				DeepWriting			
Method	PT Ep.	No PT	+PT	Spd.	PT Ep.	No PT	+PT	Spd.
Point-5	36	171	–	–	55	13	–	–
RelTokens	72	193	13	14.8×	99	23	9	2.6×
TextTokens	99	18	6	3.0×	100	30	6	**5.0**×
ScribeTokens (Ours)	90	193	9	**21.4**×	99	26	8	3.2×

Table 6: Convergence speedup from pretraining for HTG. *PT Ep.*: pretraining epochs run. *No PT*: epochs to converge without pretraining. *+PT*: fine-tuning epochs to reach the same converged loss. *Spd.*: speedup ratio (No PT / +PT). "–" indicates the pretrained model never reached the baseline loss. Best speedups are **bolded**.

	IAM				DeepWriting			
Method	PT Ep.	No PT	+PT	Spd.	PT Ep.	No PT	+PT	Spd.
Point-5	36	41	20	2.0×	55	61	–	–
RelTokens	72	80	1	80.0×	99	98	18	**5.4**×
TextTokens	99	87	4	21.8×	100	99	19	5.2×
ScribeTokens (Ours)	90	83	1	**83.0**×	99	96	27	3.6×

HTR (Table 5). On IAM, ScribeTokens achieves the largest speedup at $21.4\times$: the baseline converges in 193 epochs, while the pretrained model reaches the same loss in just 9 fine-tuning epochs. Notably, the total number of epochs for ScribeTokens with pretraining (90 pretraining + 9 fine-tuning = 99 epochs) is fewer than training without pretraining (193 epochs), while also achieving substantially better final performance (8.27% vs. 13.15% CER).

HTG (Table 6). The speedups for HTG are even more pronounced. On IAM, ScribeTokens reaches the baseline loss in a single fine-tuning epoch, yielding an $83.0\times$ speedup. Interestingly, although pretraining worsens HTG task metrics for some models on DeepWriting (Table 3), every pretrained token-based model still achieves lower cross-entropy loss than its non-pretrained counterpart. This demonstrates a disconnect between cross-entropy loss and generation quality as measured by CER and accuracy.

A.5 Double Descent in HTR Training

Without pretraining, TextTokens achieves 82.00% CER and 0% accuracy on IAM (Table 2), effectively failing to learn the recognition task. Figure 6 plots the validation cross-entropy loss over training for the three token-based models. All three initially decrease before rising around epoch 20–40, but only RelTokens and ScribeTokens recover and descend to convergence. TextTokens continues to diverge, never escaping the second ascent. We hypothesize that when the ink representation is difficult to leverage, the model initially collapses into a language model that memorizes training transcripts, grossly overfitting to the text side. RelTokens and ScribeTokens eventually learn to leverage the ink signal and escape this collapse; TextTokens never does.

To investigate this, we visualize how the most recently decoded text token attends to the ink input at different stages of decoding. We use attention rollout [33] to collapse the attention weights from all layers into a single heatmap. We also report the *attention split*: the fraction of total rolled-out attention mass allocated to ink tokens versus the previously decoded text prefix.

Figure 7 shows the attention rollout for the TextTokens HTR model (82.00% CER, 0% accuracy on IAM). The attention over ink appears diffuse and unstructured, with no discernible correspondence

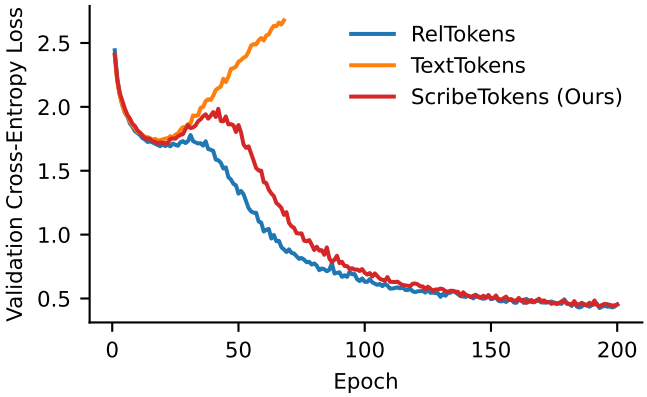


Figure 6: Validation cross-entropy loss during HTR training on IAM without pretraining. RelTokens and ScribeTokens exhibit a double descent pattern, recovering after an initial loss increase, while TextTokens diverges and fails to learn.

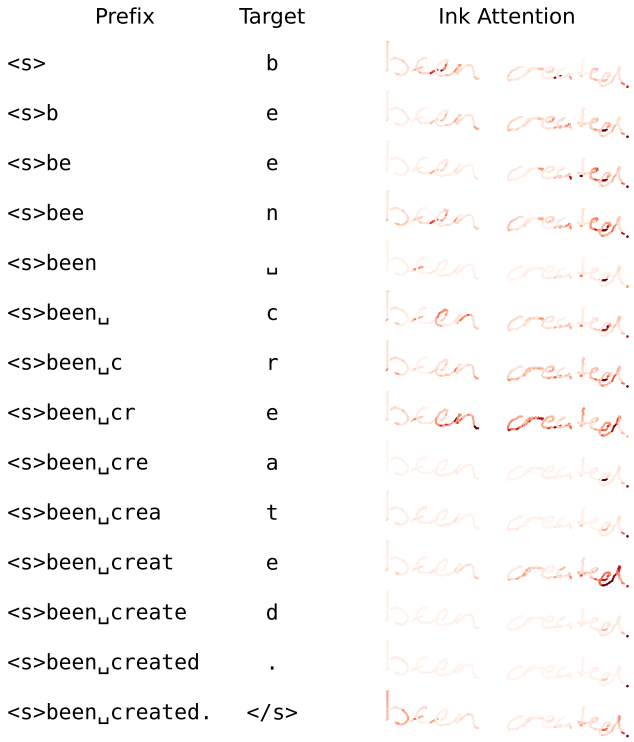


Figure 7: Attention rollout for the TextTokens HTR model (82.00% CER, 0% accuracy on IAM). Each subplot shows the rolled-out attention from the most recently decoded text token onto the ink input at a different decoding step. The attention is diffuse with no clear correspondence to the character being decoded. Attention split: 50.7% ink, 49.3% text.

between the attended ink region and the character being decoded. The attention split is 50.7% ink and 49.3% text, suggesting the model relies nearly as much on the text prefix as on the ink input.

Compare this with Figure 8, which shows the same visualization for the ScribeTokens HTR model without pretraining (13.15% CER, 22.79% accuracy on IAM). Although some spurious attention remains, there is a clear pattern: the bulk of the attention falls slightly ahead of the ink region corresponding to the character being recognized, consistent with a left-to-right reading strategy. The attention split shifts to 66.9% ink and 33.1% text.

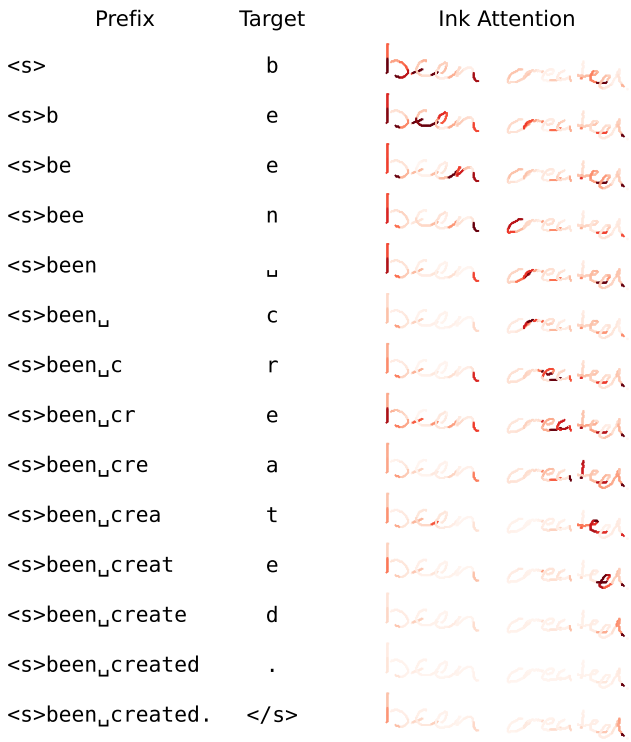


Figure 8: Attention rollout for the ScribeTokens HTR model (13.15% CER, 22.79% accuracy on IAM). Compared to TextTokens (Figure 7), the attention exhibits a clear left-to-right pattern, with the bulk of attention falling slightly ahead of the ink corresponding to the character being decoded. Attention split: 66.9% ink, 33.1% text.

We observe the same trend when pretraining is added: the best-performing model, ScribeTokens with pretraining (8.27% CER, 32.93% accuracy), pushes the attention split further to 91.2% ink and 8.8% text. Across all three models, better recognition performance correlates with a greater fraction of attention allocated to ink. While this analysis is not conclusive, it is consistent with the hypothesis that TextTokens' failure stems from an inability to effectively use the ink representation.

A.6 Understanding Pretraining

Token-based models require learned embeddings that map discrete tokens to continuous vectors. At initialization, these embeddings are random and carry no information about the spatial structure of ink. By contrast, Point-5's continuous coordinates inherently encode spatial proximity: nearby points in ink space map to nearby values in input space, giving the model a useful inductive bias from the start. For token representations, the model must first discover which tokens correspond to similar pen movements before it can reason about stroke geometry—a burden that grows with vocabulary size and token complexity. Next-ink-token pretraining addresses this cold-start problem by forcing the model to predict the next ink token from its prefix.

Why does pretraining help HTR? Learning to generate characters well requires an implicit ability to recognize what has already been written: the model must track context to decide what comes next. For example, completing the word "the" after writing "th" requires recognizing the previously generated characters. Pretraining thus builds internal representations that capture character identity from ink patterns—precisely the capability HTR requires. Pretraining also bootstraps the model into relying on the ink signal rather than the text prefix. As shown in Appendix A.5, the non-pretrained ScribeTokens HTR model allocates only 66.9% of its attention to ink; with pretraining, this rises to 91.2% (Figure 9), indicating the model has learned to decode almost entirely from the ink representation.

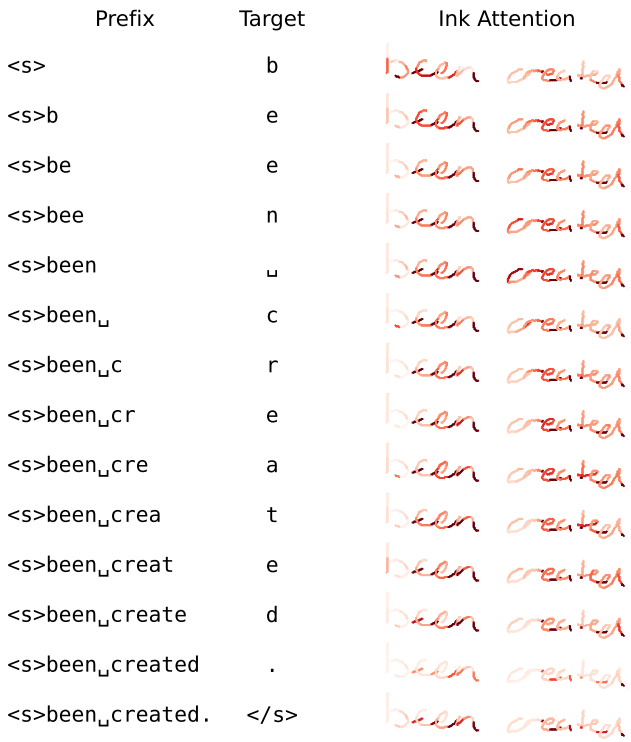


Figure 9: Attention rollout for the ScribeTokens HTR model with pretraining (8.27% CER, 32.93% accuracy on IAM). Compared to the non-pretrained model (Figure 8; 66.9% ink, 33.1% text), pretraining shifts attention almost entirely to ink. Attention split: 91.2% ink, 8.8% text.

Why does pretraining help HTG? Next-ink-token prediction is itself a generation task: the model must compose strokes into characters and characters into words to predict accurately. Indeed, pretrained models produce recognizable characters—and short words—well before pretraining converges, confirming that the objective teaches stroke composition directly. Fine-tuning on HTG then only needs to condition this already-learned generation ability on a text prompt, rather than learning both composition and conditioning from scratch. This is consistent with the large convergence speedups observed in Appendix A.4: on IAM, ScribeTokens reaches the baseline loss in a single fine-tuning epoch ($83.0\times$ speedup).

Why does pretraining not help Point-5? Point-5's continuous coordinates already encode spatial structure explicitly, so its embeddings do not suffer from the cold-start problem. Moreover, because each Point-5 prediction step covers only a single coordinate offset, next-ink-vector prediction can be mostly solved by learning local momentum and stroke dynamics alone, without capturing higher-level character structure—unlike token representations, where each merged token spans a longer ink segment and demands genuine compositional understanding. Pretraining can even *hurt* Point-5: CER rises from 9.43% to 13.63% on IAM HTR (Table 2) and nearly doubles from 14.36% to 26.84% on DeepWriting HTG (Table 3).

A.7 Generated Ink Samples

Figure 10 shows generated ink samples from all four representations, with and without next-ink-token prediction pretraining. All outputs are post-processed (Appendix A.1) before visualization. Pretraining improves legibility across the board, but representation-specific failure modes remain visible. Point-5's uncompressed sequences are considerably longer, slowing inference and degrading quality toward the end. RelTokens exhibits collapsed spacing between characters—an artifact of generated `[UNKNOWN]` tokens that encode no valid displacement. TextTokens generates malformed digit sequences that do not parse into valid coordinate pairs, yielding missing strokes. This is

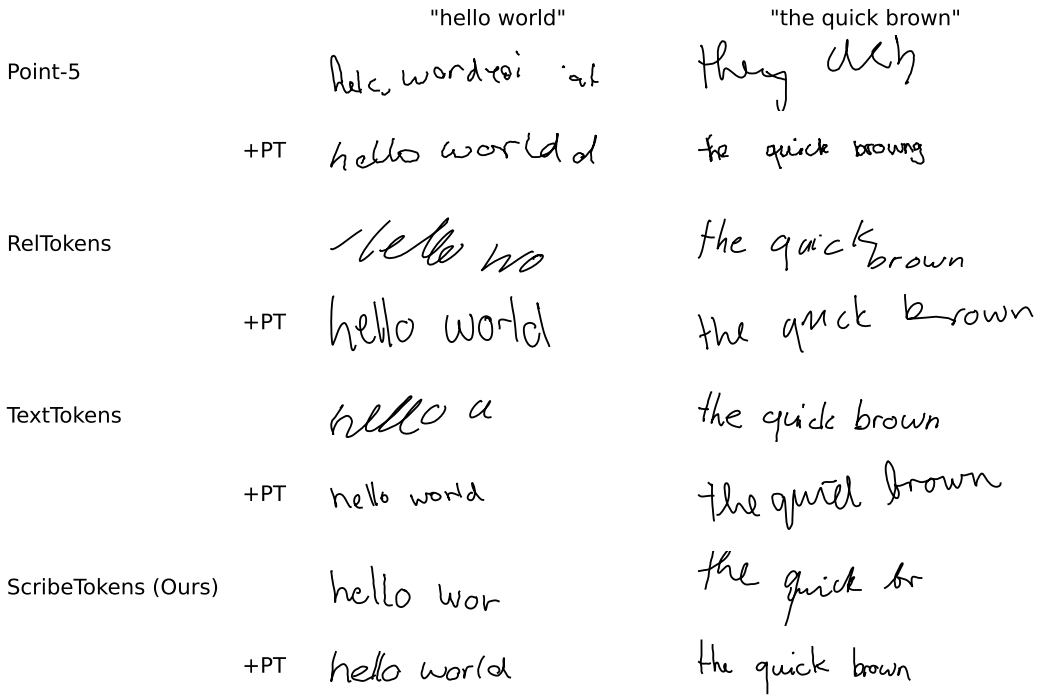


Figure 10: Generated ink samples from all four representations trained on IAM, under autoregressive decoding at temperature 1. All outputs are post-processed (Appendix A.1). Columns correspond to different text prompts; rows alternate between models trained from scratch and models initialized with next-ink-token prediction pretraining (+PT). Without pretraining, outputs are frequently incomplete or illegible; pretraining substantially improves legibility across all methods, consistent with the quantitative gains in Table 3.

especially common for long strokes such as those in cursive, where the model emits an odd number of coordinate values that cannot be grouped into (x, y) pairs. ScribeTokens + PT is the only method to correctly generate both prompts.