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 satisfactory unified representation. Continuous vector representations produce long sequences and suffer from training instability, while existing token representations introduce out-of-vocabulary issues and underperform vectors on recognition tasks. We propose ScribeTokens, a tokenization that decomposes pen strokes into unit pixel steps via Bresenham's line algorithm and encodes each step as one of eight directional tokens inspired by Freeman chain codes. Together with two pen-state tokens, this fixed 10-token base vocabulary can represent any digital ink regardless of language or writing style, and BPE compression over it yields high compression rates. On handwritten text recognition, ScribeTokens is the only token representation to outperform vectors on DeepWriting (10.75% vs. 10.94% CER). On handwritten text generation, ScribeTokens achieves 17.33% CER compared to 70.29% for vectors on IAM, demonstrating that token representations are far more effective for generation. We further introduce next-token prediction on ink sequences as a self-supervised pretraining objective, which consistently improves all token-based models in data-limited regimes and accelerates convergence by over 20×. With pretraining, ScribeTokens achieves the best recognition results across both datasets evaluated (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 [Graves et al., 2008] and mathematical expression parsing [Chan and Yeung, 2000] to sketch synthesis [Ha and Eck, 2017] and handwriting generation [Graves, 2013]. 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 sequence length—which impacts inference and training speed—and training stability, which influences both training 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 [Graves et al., 2008, Graves, 2013, Ha and Eck, 2017]. While widely used, they produce long sequences and suffer from training instability, particularly for generation tasks where models must output continuous coordinate values [Cui et al., 2019, Makansi et al., 2019]. Token representations have emerged as an alternative [Fadeeva et al., 2024, Ribeiro et al., 2020], enabling compression via Byte-Pair Encoding (BPE) [Sennrich et al., 2016] and more stable training through cross-entropy loss. However, existing tokenization methods introduce out-of-vocabulary (OOV) issues, and as we show, they underperform vector representations on recognition benchmarks.

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.

We propose *ScribeTokens*, a token representation that eliminates OOV issues entirely using a fixed 10-token base vocabulary. Our key insight is to decompose pen movements into unit steps between adjacent pixels via Bresenham's line algorithm [Bresenham, 1965]. Each step is encoded as one of eight directional tokens inspired by Freeman chain codes [Freeman, 1961]. These eight tokens and two pen state tokens (up/down) suffice to represent any digital ink, regardless of language, script, or writing style. 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 [Liwicki and Bunke, 2005]—demonstrating that vector representations struggle with generation. On HTR, ScribeTokens is the only token representation to outperform vectors on DeepWriting [Aksan et al., 2018] (10.75% vs. 10.94% CER) and narrows the gap on IAM.

We further investigate next-token prediction (NTP) on ink sequences [Radford et al., 2018, Brown et al., 2020] as a self-supervised pretraining task. Across all representations and tasks, NTP accelerates convergence by over 20× even when pretraining on the same training set, suggesting that scaling to larger unlabeled ink corpora could yield further gains. For HTR, NTP consistently improves every token representation; vectors are the only representation that worsens. ScribeTokens with NTP achieves the best HTR performance on both metrics and both datasets. For HTG, NTP does not consistently improve performance but is particularly effective in data-limited regimes, where ScribeTokens with NTP achieves the best results on the IAM dataset.

Contributions. (1) We propose ScribeTokens, a principled, OOV-free tokenization of digital ink based on a fixed 10-token vocabulary that enables aggressive compression through BPE and stable cross-entropy training. ScribeTokens outperforms both vector representations and prior tokenization methods on handwriting recognition and generation.[1] (2) We establish next-token prediction on ink sequences as a self-supervised pretraining paradigm, yielding over 20× faster convergence and consistent performance gains for token-based models, particularly in data-limited settings.

2 Related Work

2.1 Vector Representations

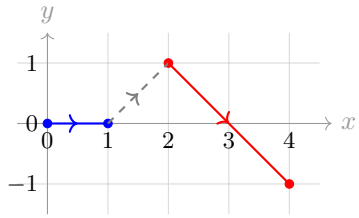
Vector representations model digital ink as a sequence of continuous vectors [Graves, 2013, Dai et al., 2023, Ha and Eck, 2017, Carbune et al., 2020]. For generation, the next xy-coordinate is modeled as a mixture of 2-dimensional Gaussians, with special pen events (such as pen-up) represented by a discrete distribution [Graves, 2013, Dai et al., 2023, Ha and Eck, 2017].

Vector representations, while natural, suffer from several drawbacks: (1) the lack of compression often necessitates truncating long sequences [Graves, 2013],[2] (2) many design choices must be made for input normalization [Graves, 2013, Aksan et al., 2018, Dai et al., 2023, Zhang et al., 2017] and sequence initiation/termination [Graves, 2013, Dai et al., 2023], and (3) mixture density networks used for generation require additional hyperparameters, suffer from numerical instability [Cui et al.,

[1] An accompanying Hugging Face [Wolf et al., 2020] library for fast tokenization and tokenizer training is available at `https://github.com/douglasswng/tokink`.

[2] While Carbune et al. [2020] 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]`

2019] and mode collapse [Makansi et al., 2019], and produce negative log-likelihood values [Bishop, 1994] that are difficult to interpret.

Point-3 [Graves, 2013] 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* [Ha and Eck, 2017] 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.

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) special `[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 [Bengio et al., 2017].

Existing token representations, however, face common problems: (1) discretization of continuous coordinates introduces approximation, (2) out-of-vocabulary (OOV) issues when unseen coordinate combinations are mapped to `[UNKNOWN]`, and (3) large base vocabularies that scale with canvas resolution.

AbsTokens [Ribeiro et al., 2020] treats each pixel coordinate as a token, with `[UP]` for pen-up and `[UNKNOWN]` for unseen coordinates. *RelTokens* [Ribeiro et al., 2020] is similar but uses relative offsets $(\Delta x, \Delta y)$ as tokens. *TextTokens* [Fadeeva et al., 2024] serializes each offset as its decimal string (digits, minus signs, and spaces), avoiding OOV issues but producing longer sequences.

2.3 Pretraining

Next-token prediction (NTP) on large unlabelled corpora followed by supervised fine-tuning has become the dominant paradigm in NLP [Radford et al., 2018, Devlin et al., 2018, Brown et al., 2020, Raffel et al., 2020] and has been extended to speech [Du et al., 2024] and music [Dhariwal et al., 2020], but NTP pretraining directly on digital ink remains unexplored. Fadeeva et al. [2024] leverage

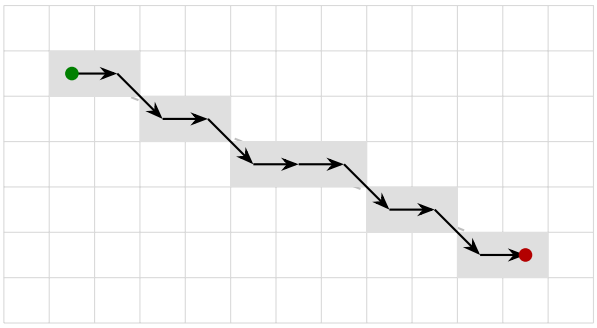


Figure 2: Bresenham Decomposition of a line segment between two grid points (start ●, end ●). The segment is rasterized into adjacent grid cells via Bresenham's algorithm, then encoded as a sequence of Freeman chain code directions.

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 NTP as a self-supervised pretraining objective applied directly to ink token sequences.

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. [2020], 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 [Freeman, 1961] for directional encoding and Bresenham's line algorithm [Bresenham, 1965] 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.

Algorithm 1 ScribeTokens encoding

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

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.

Rendering invariance. An important property of ScribeTokens is rendering invariance: digital inks that render 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.

Decoding. For generation tasks, the predicted token sequence must be decoded 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 [Savitzky and Golay, 1964] 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 token 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 [Graves, 2013].

4 Experiments

4.1 Datasets

IAM. The IAM On-Line Handwriting Database (IAM-OnDB) [Liwicki and Bunke, 2005] 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 [Aksan et al., 2018] 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. 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 each δ, decode back to the original coordinate space, and apply Savitzky–Golay post-processing (Appendix A.1). Figure 5 (Appendix) shows representative examples: 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 $\delta \geq 128$.

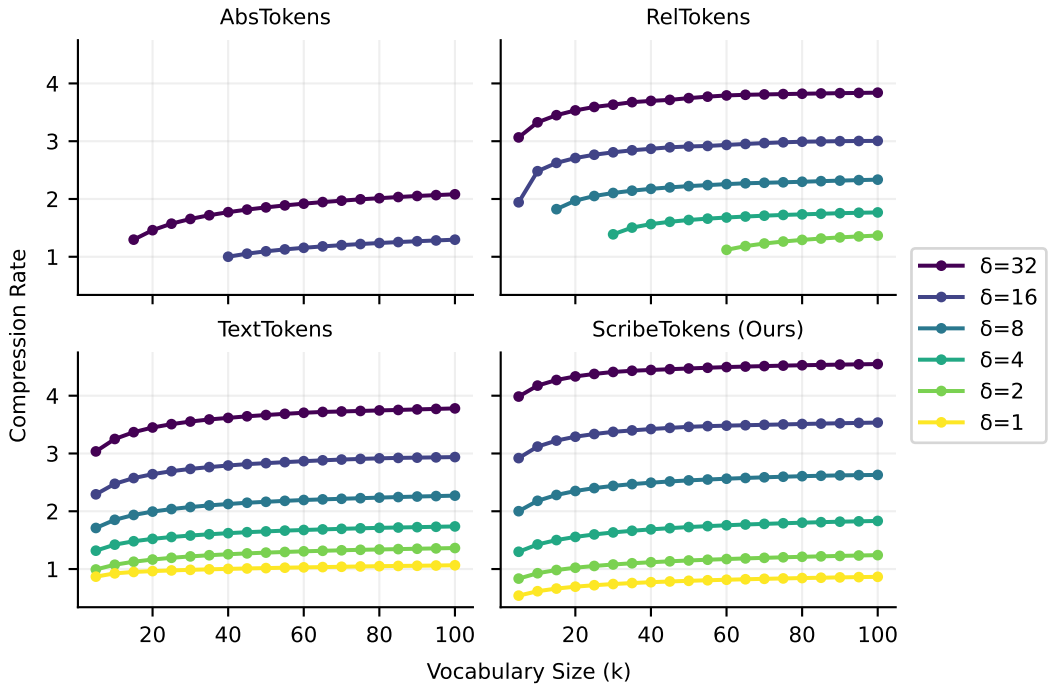


Figure 3: Average compression rates (↑) of BPE-based digital ink representations on the IAM validation set, across target vocabulary sizes and quantization parameters δ. ScribeTokens consistently achieves the highest compression across nearly all settings.

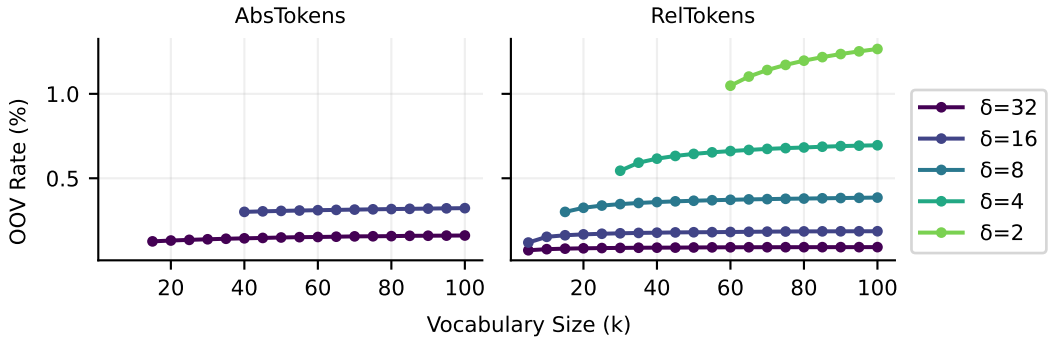


Figure 4: Average out-of-vocabulary (OOV) rates (↓) of BPE-based digital ink representations on the IAM validation set, across target vocabulary sizes and quantization parameters δ. ScribeTokens and TextTokens are OOV-free by construction, while AbsTokens and RelTokens exhibit non-zero OOV rates as their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

4.3 Experimental 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 is used for all representations and tasks, ensuring that performance differences are driven by the choice of representation rather than architecture.

Architecture and training. Our model is a 12-layer decoder-only Transformer [Vaswani et al., 2017] following the LLaMA design [Touvron et al., 2023], with approximately 34M parameters

Table 2: Handwritten text recognition on IAM and DeepWriting, measured by CER (↓) and accuracy (↑) under autoregressive decoding (T=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)

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 [Loshchilov and Hutter, 2019] 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 recognition. Table 2 reports character error rate (CER) and exact-match accuracy under autoregressive decoding (T=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. NTP 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).

4.5 Handwritten Text Generation

The task is to produce an ink sequence conditioned on a text prompt. As above, +PT models are first pretrained with next-ink-token prediction, then fine-tuned for generation. Table 3 reports CER and exact-match accuracy under autoregressive decoding (T=1); to evaluate legibility, generated inks are recognized by the best HTR model (ScribeTokens + PT) and scored against the input text.

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

Table 3: Handwritten text generation on IAM and DeepWriting, measured by CER (↓) and accuracy (↑) under autoregressive decoding ($T=1$). Generated inks are evaluated by the best HTR model (ScribeTokens + PT from Table 2). +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		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)

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-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 trajectory, eliminates out-of-vocabulary issues by construction, and enables aggressive BPE compression. Across handwriting recognition and generation benchmarks, ScribeTokens outperforms both vector representations and prior tokenization methods, demonstrating that a principled discrete representation can match the natural spatial structure of vectors while retaining the training stability and compression advantages of tokens.

We further established next-token prediction on ink sequences as a self-supervised pretraining paradigm. NTP pretraining consistently improves every token-based model on recognition, accelerates convergence by over 20× (Appendix A.4), and is particularly effective in data-limited regimes. Attention analysis reveals that pretraining shifts the model toward relying on the ink signal rather than memorized text patterns (Appendix A.5). We provide intuition for why pretraining benefits token-based models but not continuous vectors in Appendix A.6.

Limitations and future work. Our experiments are limited to English handwriting on two datasets with a single model architecture. ScribeTokens' fixed vocabulary naturally extends to any language or script, but this remains to be validated empirically. NTP pretraining was performed on the same labeled training sets (using only the ink); scaling to larger unlabeled ink corpora is a natural next step that could yield further gains. Finally, while we focused on recognition and generation, other digital ink tasks—such as writer identification, sketch recognition, and mathematical expression parsing—may also benefit from ScribeTokens and NTP pretraining.

Acknowledgments and Disclosure of Funding

The author thanks Huidong Liang and Zeli Wang for reading earlier drafts and providing valuable feedback. The author also thanks his family for their unwavering support and encouragement throughout this work. This work was self-funded and not supported by any external grants or organizations.

References

Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers. *arXiv preprint arXiv:2005.00928*, 2020.

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.

Yoshua Bengio, Ian Goodfellow, Aaron Courville, et al. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

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

J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1): 25–30, 1965. ISSN 0018-8670. doi: 10.1147/sj.41.0025.

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.

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 (IJDAR)*, 23(2):89–102, 2020.

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

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 *2019 international conference on robotics and automation (icra)*, pages 2090–2096. IEEE, 2019.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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.

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.

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.

Herbert Freeman. On the encoding of arbitrary geometric configurations. *IEEE Transactions on Electronic Computers*, EC-10(2):260–268, June 1961. ISSN 0367-7508. doi: 10.1109/tec.1961.5219197.

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

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, 2008.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

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 (ICDAR'05)*, pages 956–961. IEEE, 2005.

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

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.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI*, 2018.

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.

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.

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

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)*, page 1715–1725. Association for Computational Linguistics, 2016. doi: 10.18653/v1/p16-1162.

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

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

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.

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.

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. Association for Computational Linguistics, 2020.

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

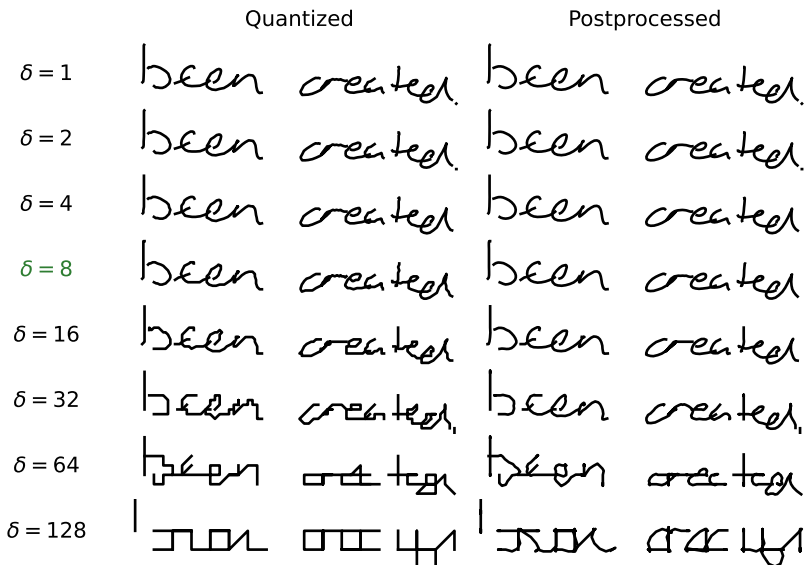


Figure 5: Effect of quantization parameter δ on reconstruction quality. Each row shows the same IAM sample quantized at a different δ, displayed both as raw quantized ink (left) and after Savitzky–Golay post-processing (right). Post-processed inks are visually indistinguishable from the original for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

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.

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 [Savitzky and Golay, 1964] 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 [Liwicki and Bunke, 2005] and DeepWriting [Aksan et al., 2018], $d = 2$ works well.

A.2 Point-5 Model

The Point-5 representation [Ha and Eck, 2017] 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.

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 [Su et al., 2024]
Activation	SwiGLU [Shazeer, 2020] (expansion factor $8/3$)
Normalization	Pre-norm RMSNorm [Zhang and Sennrich, 2019]
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	$\sigma = 5$

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 is projected to mixture weights π_k (via softmax), means $\boldsymbol{\mu}_k \in \mathbb{R}^2$, standard deviations $\boldsymbol{\sigma}_k \in \mathbb{R}^2$ (via softplus), and correlation coefficients ρ_k (via tanh) for each component. The pen state (p_1, p_2, p_3) is modeled as a 3-class categorical distribution via a separate linear projection.

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, NTP pretraining also accelerates downstream convergence. Tables 5 and 6 report, for each representation, how many fine-tuning epochs a pretrained model needs to match the converged validation loss of its non-pretrained counterpart.

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 [Abnar and Zuidema, 2020] 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

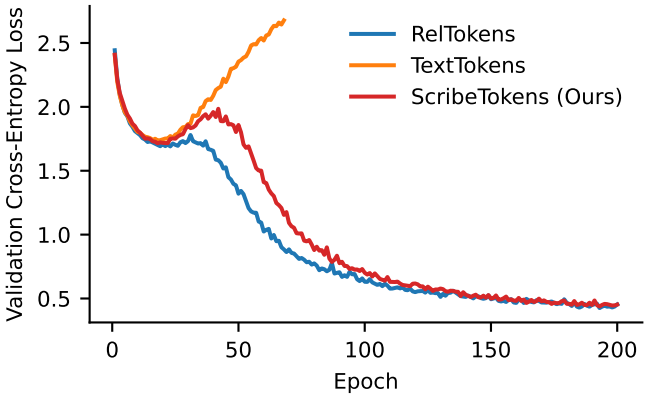


Figure 6: Validation cross-entropy loss during HTR training on IAM without pretraining. RelTokens and ScribeTokens exhibit a double descent pattern, recovering after an initial loss increase, while TextTokens diverges and fails to learn.

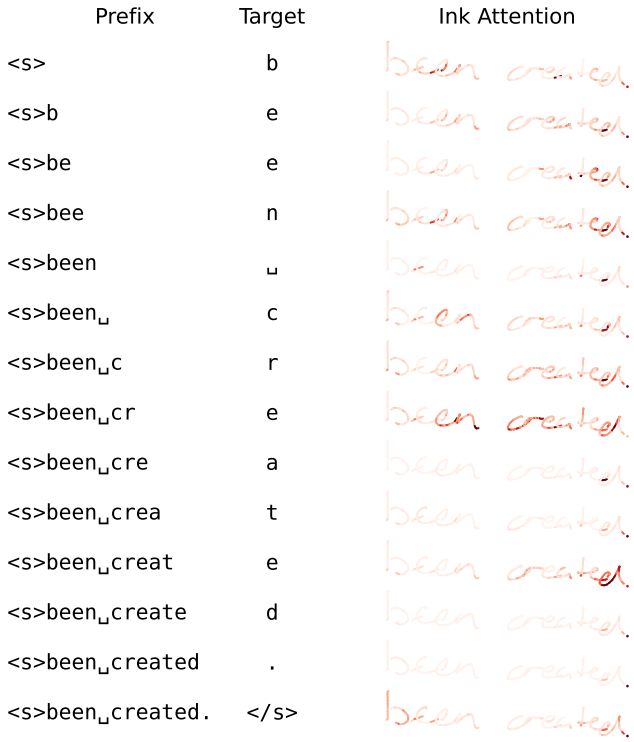


Figure 7: Attention rollout for the TextTokens HTR model (82.00% CER, 0% accuracy on IAM). Each subplot shows the rolled-out attention from the most recently decoded text token onto the ink input at a different decoding step. The attention is diffuse with no clear correspondence to the character being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

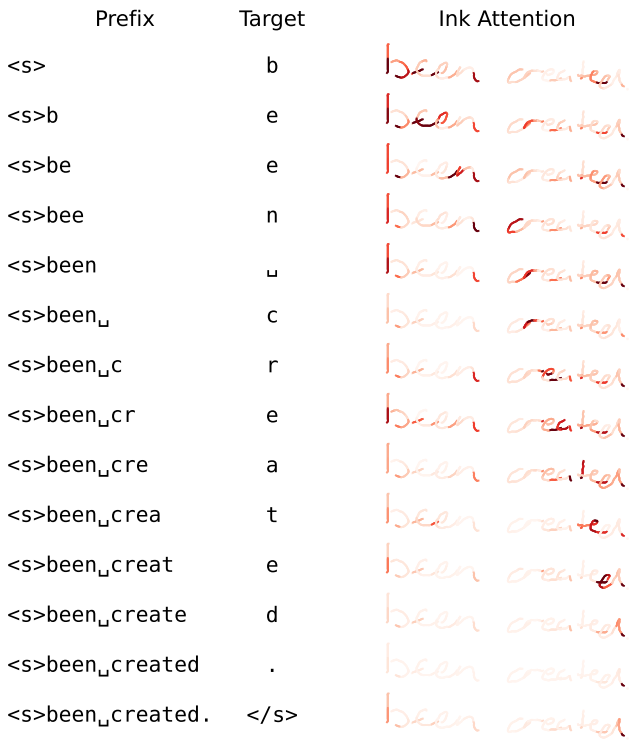


Figure 8: Attention rollout for the ScribeTokens HTR model (13.15% CER, 22.79% accuracy on IAM). Compared to TextTokens (Figure 7), the attention exhibits a clear left-to-right pattern, with the bulk of attention falling slightly ahead of the ink corresponding to the character being decoded. Attention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

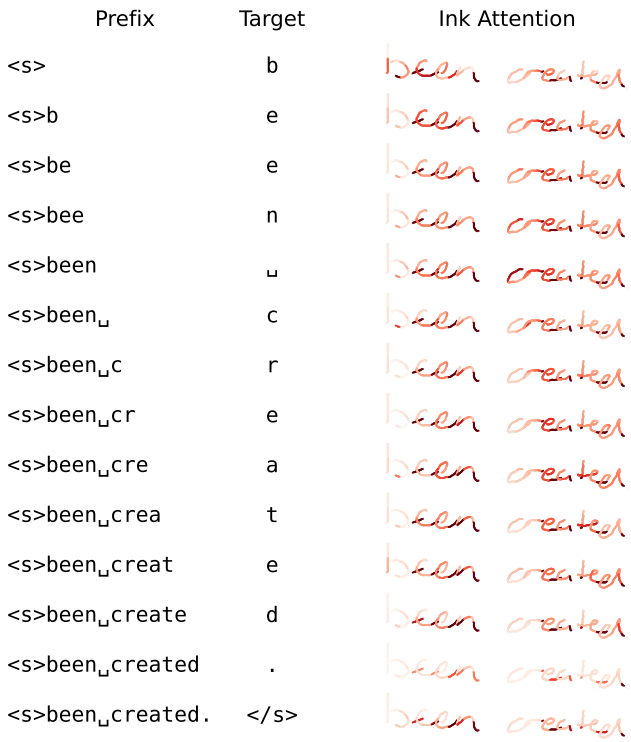


Figure 9: Attention rollout for the ScribeTokens HTR model with pretraining (8.27% CER, 32.93% accuracy on IAM). Compared to the non-pretrained model (Figure 8; 66.9% ink, 33.1% text), pretraining shifts attention almost entirely to ink. Attention split: 91.2% ink, 8.8% text.

Why does pretraining help HTG? NTP is itself a generation task: the model must compose strokes into characters and characters into words to predict accurately. Indeed, NTP 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× 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, NTP can be 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), suggesting that the low-level dynamics learned during NTP conflict with downstream task demands.