Deep Learning Transcription and Normalization of Early Modern English Texts

Nick Antoine, Arun Munagala, Anuj Prakash, Evgenios Oikonomou
Deep Learning Systems

***Abstract*—Early Modern English (EME) documents present a challenging testbed for deep learning, combining visual noise from historical scans with orthographic variation and obsolete spelling. In this project we develop and compare three neural OCR models—a baseline PyTorch model trained from scratch, a TrOCR model, and a Kraken model—on a corpus of approximately 11,000 line images. Due to resource constraints, the baseline and Kraken models were trained on the full dataset, whereas TrOCR was trained on a 3,000-line subset. On top of the best-performing OCR (Kraken), we stack a large language model to perform spelling normalization and OCR error correction. For this second stage we experimentally compare Flan-T5 with Mistral-7B and ultimately adopt Mistral-7B as our main normalizer. The resulting pipeline demonstrates that combining domain-specific OCR with a strong sequence model can substantially improve readability and searchability of historical text, while also highlighting the limitations posed by task complexity, GPU budgets, and irrecoverable OCR errors.**

I. Introduction

Digitized historical collections such as EEBO, ECCO, and related Text Creation Partnership (TCP) corpora offer millions of pages of Early Modern English print, but these scans are often accompanied by noisy OCR output that is unsuitable for linguistic research or downstream NLP applications. Conventional OCR engines struggle with EME-specific challenges: long-s and blackletter glyphs, ligatures, damaged type, and highly non-standard spelling. Even when high-quality human transcriptions exist, they are rarely aligned with the images at line level in a way that directly supports deep-learning OCR training.

Our project targets this gap by designing a two-stage deep-learning system. The first stage is OCR, which takes a line image and produces a character sequence; the second stage is normalization, which takes noisy OCR text and outputs a corrected and modernized version. Methodologically, we frame OCR as a sequence prediction problem over images and normalization as a conditional generation problem over text.

Practically, we compare multiple architectures for both stages: a baseline CNN–BiLSTM–CTC model implemented in PyTorch, a transformer-based TrOCR model, and the historical-ocr-focused Kraken engine for OCR; and Flan-T5 versus Mistral-7B for normalization. We also explore lightweight alternatives including SymSpell spell-checking and a hybrid pipeline combining phonetic matching with fuzzy logic, evaluating the trade-offs between transformer-based and traditional NLP approaches for historical text correction.

The project sits at the intersection of computer vision, sequence modeling, and computational humanities. It aims not only to achieve strong empirical results but also to document the trade-offs between architectures, dataset size, GPU cost, and error propagation across a multi-stage pipeline.

II. Related Work

This section reviews prior work relevant to both stages of our pipeline: optical character recognition for historical documents and post-OCR normalization using rule-based, neural, and large language model approaches. Rather than discussing each work exhaustively in prose, we provide concise thematic summaries below and refer the reader to the accompanying tables for a complete comparison.

A. OCR Models

Prior work on OCR for historical and handwritten text spans both classical sequence models and modern transformer-based architectures. Early neural approaches such as CNN–RNN–CTC models established a strong foundation for image-based sequence recognition and remain widely used in historical text recognition systems. More recent transformer-based OCR models leverage large-scale pretraining and autoregressive decoding, achieving strong performance on clean and printed text, though often struggling with degraded or highly stylized historical material.

Comparative studies consistently show that domain-specific OCR engines tailored for historical scripts outperform generic OCR models when applied to early printed or handwritten documents. In particular, systems such as Kraken demonstrate robustness to long-s characters, ligatures, and variable typography that commonly appear in Early Modern English sources.

B. NLP Models

Post-OCR correction and normalization has been studied using a wide range of techniques, from classical rule-based methods to modern neural and large language models. Early approaches relied on edit distance, dictionary lookup, and handcrafted confusion rules to correct OCR errors at scale. While effective for systematic noise, these methods lack contextual awareness and struggle with ambiguous corrections.

More recent work incorporates contextual language models such as BERT-style masked language models and sequence-to-sequence transformers, enabling context-sensitive correction and historical spelling normalization. Large language models further improve fluency and semantic coherence but remain

sensitive to severe OCR degradation and domain mismatch. These findings motivate our exploration of both lightweight hybrid methods and transformer-based normalization models.

III. Data and Preprocessing

Our original intention was to build a training corpus by scraping line-level data from TCP XML and Google Books scans, following the project proposal. However, the extracted images were noisy, inconsistently cropped, and frequently misaligned with their transcriptions. A second attempt using the Bodleian First Folio yielded too few line samples for meaningful training.

Consequently, we adopted the Bentham Papers dataset (Transkribus / Zenodo), an open-source, well-segmented corpus containing 11,473 labeled line images. Its consistency and size made it far more suitable for training deep OCR. Images were converted to grayscale, normalized in height, lightly augmented, and batched for model training. Kraken outputs on this dataset served as input to our normalization models.

Despite being handwritten rather than printed, Bentham offered the only viable path to stable model training within our computational budget.

IV. Methods

A. OCR Models

We explore three distinct OCR architectures.

1) Baseline PyTorch Model (From Scratch): The baseline is a conventional CNN–BiLSTM–CTC model implemented directly in PyTorch. A convolutional stack encodes the input greyscale line image into a sequence of feature vectors along the horizontal axis. This sequence is then passed through a bidirectional LSTM, and a linear layer maps each timestep to a distribution over characters. Training uses CTC loss, allowing us to align variable-length character sequences with variable-width images without explicit segmentation.

We train this model on the full approximately 11,000-line dataset. The model is intentionally simple, serving as a didactic reference point for the course rather than an optimized production OCR.

2) TrOCR Model: TrOCR combines a vision transformer (ViT) encoder with an autoregressive text decoder. In our experiment, we fine-tune a pretrained TrOCR variant on a 3,000-line subset of the dataset due to resource constraints: full-dataset fine-tuning would exceed Colab Pro limits in both memory and wall-clock time.

This model leverages large-scale pretraining on generic OCR-like data, but its effective adaptation to Early Modern typography is limited by the smaller fine-tuning set.

3) Kraken Model: Kraken is an OCR/HTR system specifically designed for historical documents. Under the hood, it uses a CNN–RNN–CTC architecture but with training recipes and text encoding tailored to complex scripts. We train Kraken on the same full 11,000-line dataset, using a batch size of four and approximately 20 or more hours of training on an L4 GPU.

Training proceeds for multiple epochs, on the order of tens of passes over the data, and checkpoints are saved as `.mlmodel` files for later evaluation. In practice, Kraken consistently delivered the most stable and legible line transcriptions among the three approaches.

B. Normalization Models

1) SymSpell: SymSpell is a fast symmetric delete spell-checking algorithm optimized for edit-distance lookups. We experiment with SymSpell as a lightweight alternative to transformer-based normalizers. The algorithm generates all possible edits (deletions) for each dictionary word up to a maximum edit distance (set to 2 in our experiments), then uses these precomputed variations to find candidate corrections for OCR errors. We build a custom SymSpell dictionary from the ground-truth Bentham transcriptions, enriched with common Early Modern English words. This adaptation is critical: generic spell-checkers trained on modern English fail on historical vocabulary. SymSpell correction is deterministic and executes in under one second for the entire 48-document test set, requiring no GPU. However, SymSpell operates word-by-word without context, limiting its ability to resolve ambiguous errors where multiple plausible corrections exist.

2) Hybrid Method (Five-Stage Pipeline): The hybrid method combines multiple complementary techniques in sequence to maximize coverage while maintaining speed. First, regex-based cleanup handles systematic OCR patterns: replacing "——" with "ll", isolated "—" with "l", "rn" with "m", and digit-like characters (0→O, 1→I) in non-numeric contexts. Second, SymSpell performs fast dictionary lookup with edit distance 2. Third, for words not found by SymSpell, we apply phonetic matching using the Double Metaphone algorithm: this encodes both the OCR output and dictionary words into phonetic codes, then retrieves candidates with matching codes. Fourth, fuzzy string matching (via Levenshtein ratio) ranks phonetic candidates and rejects low-similarity corrections. Fifth, capitalization is preserved from the original OCR to maintain proper names and sentence structure.

The hybrid method outperforms standalone SymSpell by leveraging phonetic and fuzzy logic to recover from errors that pure edit-distance methods miss, particularly OCR mistakes that preserve phonetic similarity (e.g., "consigned" misread as "aomsegmed").

3) DistilBERT (Masked Language Model): DistilBERT is a distilled version of BERT with 66 million parameters, trained for masked language modeling. We experiment with using DistilBERT to identify and correct suspicious words based on contextual plausibility. The approach works in three steps: first, calculate perplexity for each word in context by masking it and measuring the model's uncertainty; second, flag high-perplexity words as likely OCR errors; third, use DistilBERT's fill-mask pipeline to generate correction candidates, filtering by both model confidence ($>15\%$) and string similarity ($>60\%$) to avoid spurious replacements. Corrections are capped at three per document to prevent cascading errors.

This method applies the hybrid correction first, then refines it using contextual information. However, DistilBERT catastrophically failed in initial trials due to over-aggressive corrections. The conservative thresholds and similarity filters described above were implemented to mitigate this, but even with these safeguards, DistilBERT struggled with domain mismatch: it is pretrained on modern web text and lacks the historical vocabulary necessary for Early Modern English. This highlights a key limitation of general-purpose language models for specialized historical domains.

4) Flan-T5: Flan-T5 is an instruction-tuned family of encoder–decoder transformers. We experiment with using a Flan-T5 checkpoint as a conditional generator: given a raw OCR line, the model is trained to output the corrected and normalized line. While Flan-T5 can handle moderate noise, its capacity is limited by model size and the fact that its tokenizer and pretraining are not specialized for historical English.

5) Mistral-7B: Mistral-7B is a 7-billion-parameter decoder-only language model. In our setup, we use it in a text-to-text format by prompting it with the OCR line and training (or instruction-tuning) it to emit the normalized counterpart. Despite being more resource-intensive than Flan-T5, Mistral-7B demonstrates stronger robustness to noisy inputs. Empirically, and as documented in the notebook, Mistral-7B outperforms Flan-T5 on our normalization task, especially when dealing with moderate OCR corruption and archaic spellings. For this reason, Mistral-7B is adopted as the main normalizer in our final pipeline, while Flan-T5 is retained as a baseline.

V. Experiments and Results

The three OCR models are compared on held-out lines not seen during training. We compute standard metrics such as character error rate (CER) and qualitatively inspect the output. While exact numeric scores are reported in the notebook, the trends are clear.

First, the baseline PyTorch model suffers from relatively high CER, especially on lines with ornate fonts or significant noise. Given that it is trained from scratch on only approximately 11,000 lines, this is unsurprising: the model lacks both architectural sophistication and large-scale pretraining.

Second, TrOCR, despite its strong architecture and pretraining, is limited by being fine-tuned on just 3,000 lines. It performs better than the baseline on many simpler examples, but underfits the full diversity of the historical material. Some systematic confusions remain unresolved.

Third, Kraken, trained on the full 11,000-line dataset, provides the best overall OCR performance. It handles long-s, ligatures, and variable kerning more gracefully, producing line transcriptions that are often close enough to the ground truth for downstream normalization to succeed.

For normalization, we evaluate Flan-T5 and Mistral-7B on the task of mapping Kraken output to normalized text. Both models can correct minor OCR mistakes and modernize straightforward archaic spellings. However, Mistral-7B consistently generates more plausible and grammatical sentences, particularly when the input contains multiple corrupted tokens. The difference is most evident in cases where Kraken's output is noisy but recognizable: Flan-T5 may partially correct the line, while Mistral-7B often reconstructs a fully fluent modern rendering.

A crucial failure mode arises when Kraken catastrophically misrecognizes a word, producing an output that bears little resemblance to any English token. In those cases, neither normalizer can reliably infer the original word. This supports the conclusion that improvements in the OCR stage are strictly necessary to push end-to-end performance beyond a certain threshold.

Beyond these transformer approaches, we also tested a lightweight hybrid method combining SymSpell, phonetic matching, and fuzzy logic, achieving an 8.86% CER improvement in under one second on CPU. We further experimented with DistilBERT for context-aware correction, but it catastrophically failed, degrading CER by 31% due to domain mismatch with historical text.

VI. Discussion and Limitations

Methodologically, our experiments show that combining a domain-adapted OCR model with a strong language model can significantly improve the usability of historical text. Kraken's specialization for historical scripts and its training on the full 11,000 lines give it a clear advantage over both the from-scratch baseline and the partially fine-tuned TrOCR model. On the text side, Mistral-7B's capacity and training regime make it more tolerant of noisy inputs than Flan-T5, justifying its selection as the main normalizer in the final system.

At the same time, three key limitations constrain what our pipeline can achieve. First, the complexity of the task is substantially higher than in typical educational OCR projects. We work with full sentences drawn from Early Modern English print, not isolated digits or modern words. The models must handle non-standard spelling, historical punctuation, long-s, ligatures, and scan artifacts simultaneously.

Second, we face limited computational resources. Training Kraken on 11,000 lines with a batch size of four already requires more than 20 hours on a Colab Pro L4 GPU, leaving little slack for extensive hyperparameter tuning or multiple restarts. TrOCR is constrained to a 3,000-line subset precisely because full-dataset fine-tuning would exceed our budget.

Third, some errors are irrecoverable at the normalization stage. When Kraken's transcription of a word is heavily distorted, involving multiple insertions, deletions, and substitutions, the resulting token is effectively out of vocabulary and context for the normalizer. Even Mistral-7B cannot reconstruct the original word if the visual signal has been destroyed.

A further limitation concerns evaluation reproducibility across toolchains. Although Kraken reports high character and word accuracy when evaluated using the `ketos test` command, reproducing exactly the same results in a custom Python-based evaluation proved difficult. This discrepancy arises because `ketos test` uses Kraken's internal data pipeline, including assumptions about line segmentation, preprocessing, and ground-truth alignment that are tightly coupled

to training. Python-based evaluations necessarily reimplement parts of this pipeline, and small mismatches can yield substantially different metrics. As a result, Python-side evaluations are best interpreted qualitatively, while `ketos test` should be treated as authoritative for Kraken performance.

VII. Conclusion

This project demonstrates a realistic end-to-end deep learning pipeline that combines computer vision and large language models in a challenging historical setting. We systematically compare three OCR approaches—baseline PyTorch, TrOCR, and Kraken—on approximately 11,000 historical line images, and we evaluate two normalization models, Flan-T5 and Mistral-7B, on top of the best-performing OCR.

Our findings underscore the importance of domain-specific OCR training and the benefits of using a strong sequence model for error correction and modernization. Future work could explore larger OCR architectures, broader fine-tuning of transformer-based OCR models, and scaling normalization to full TCP corpora. Integrating confidence estimation, active learning, and lightweight domain-adapted language models may further improve performance while maintaining feasibility for large-scale digitization projects.

Acknowledgment

This work was completed as part of the Deep Learning Systems graduate course.

References

[1] B. Shi, X. Bai, and C. Yao, "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 11, pp. 2298–2304, 2017.

[2] M. Maarand, A. Lervik, and L. J. Kjelstadli, "A Comprehensive Comparison of Open-Source Libraries for Handwritten Text Recognition in Norwegian," in *Proc. DAS*, 2022, pp. 1–6.

[3] M. Li, T. Lv, Y. Cui, Y. Lu, D. Florencio, C. Zhang, Z. Li, and J. Wei, "TrOCR: Transformer-Based Optical Character Recognition with Pre-Trained Models," in *Proc. AAAI*, 2022, pp. 13094–13102.

[4] J. Westerdijk, A. van der Sijs, and E. Kanoulas, "Improving OCR for Historical Texts of Multiple Languages," *Digital Scholarship in the Humanities*, 2025.

[5] G. Crosilla, L. Klic, G. Colavizza, "Benchmarking Large Language Models for Handwritten Text Recognition," arXiv preprint arXiv:2503.15195, 2024.

[6] M. Reynaert, "Non-Interactive OCR Post-Correction for Giga-Scale Digitization Projects," in *Proc. CICLing*, 2008, pp. 617–630.

[7] M. Hajiali, J. F. Cacho, and K. Taghva, "Generating Correction Candidates for OCR Errors Using BERT Language Model and FastText Subword Embeddings," in *Lecture Notes in Networks and Systems*, vol. 294, Springer, 2022, pp. 241–252.

[8] Y. Lyu, T. Demeester, and C. Develder, "Neural OCR Post-Hoc Correction of Historical Corpora," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 479–493, 2021.

[9] L. Debaene, M. Reynaert, and V. Hoste, "Evaluating Transformers for OCR Post-Correction in Early Modern Dutch Theatre," in *Proc. COLING*, 2025.

[10] J. Kanerva, C. Ledins, S. Käpyaho, F. Ginter, "OCR Error Post-Correction with Large Language Models in Historical Documents," arXiv preprint arXiv:2502.01205, 2025.