Error-Analysis-Guided Post-Correction for Automated Medieval Greek HTR

Nicklas Sindlev Andersen, Aglae Pizzone, Tariq Yousef
University of Southern Denmark, Odense, Denmark
sindlev@imada.sdu.dk, pizzone@sdu.dk, yousef@imada.sdu.dk

Abstract

Handwritten text recognition (HTR) for historical documents is often performed as a staged pipeline in which layout analysis and text line detection (TLD) produce line polygons, HTR transcribes the detected text lines, and the output may then be post-corrected. We investigate post-correction for *Vaticanus graecus* 2228 (Vat. gr. 2228), a 14th-century Medieval Greek manuscript, by fine-tuning byte-level ByT5 models on synthetic noisy-to-clean training pairs generated from validation-derived HTR error profiles. We evaluate post-correction across four factors: HTR model training strategy (models trained from scratch vs. initialized from a pretrained model), line polygon quality (manually corrected vs. automatically detected polygons), source context (using the current line alone vs. using additional neighboring lines), and clean-text source mixture (relying more on manuscript ground truth vs. external Greek text). Post-correction is most effective with automatically detected polygons. In this condition, using a source mixture relying more on external Greek text, word error rate (WER) decreases from 36.82% to 30.87% for HTR initialized from a pretrained Medieval Greek base model when post-correction uses additional neighboring lines, and from 44.79% to 36.50% for HTR trained from scratch when post-correction uses the current line alone. Improvements with manually corrected polygons are smaller and less consistent, showing that post-correction can recover part of the error introduced by automated processing but does not replace accurate TLD or HTR.

1 Introduction

The digitization and transcription of historical documents increasingly depend on automated workflows that combine layout analysis, text line detection (TLD), handwritten text recognition (HTR), and downstream correction. Some systems aim to collapse more of this process into a single model, but staged pipelines remain common in practical manuscript workflows because they expose intermediate artifacts. In low-resource cultural-heritage settings, this visibility supports auditability and expert intervention, making it possible to improve different stages incrementally. Because manual intervention can be time-consuming, it is valuable to understand how errors propagate through the pipeline, whether downstream interventions can mitigate them, and where manual effort is best spent.

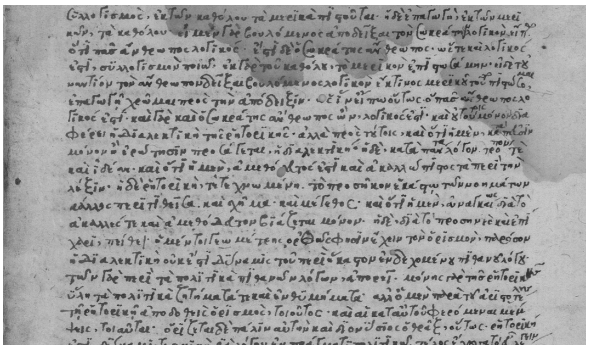

Figure 1: Example crop from Vat. gr. 2228 (f. 12r), an earlier portion whose published text supported preparation of the ground truth, showing its regular line layout and consistent writing.

For Medieval Greek bookhands, this issue is especially relevant. Rapid minuscule scripts—particularly scholarly hands—feature dense diacritics, ligatures, abbreviations, suspensions, and contractions. Together with manuscript-specific layouts, these features can affect both segmentation and recognition: TLD may produce line polygons that cut through ascenders or include marginal or between-line material, while the HTR model must distinguish visually similar characters and abbreviations.

This study focuses on the Medieval Greek manuscript Vat. gr. 2228, described in detail by

Lilla (1985, pp. 307–313). The present experiments use 44 processed pages from ff. 12r–15v, 17r–20v, and 194r–207v (Table 1). Lilla attributes the text in the sampled portions to the same hand. Together, the shared hand, regular line layout, and consistent writing reduce variation in handwriting and layout, making the sampled material well suited to a controlled within-manuscript comparison across experimental conditions. Figure 1 shows a representative page from the earlier portion, whose text was published by Walz (1835) and could therefore be used as a reference when preparing manually verified transcriptions of these pages.

The later sampled portion belongs to the primary target of the broader transcription project: ff. 194r–313r, which preserve John Doxapatres' still-unpublished 11th-century *Commentary on Hermogenes' On Invention*. Because further pages remain to be processed, the present comparison also helps determine whether future manual intervention should focus on correcting line polygons before HTR or recognized text afterward, and under which settings.

We examine this trade-off by fine-tuning ByT5 post-correction models across conditions of HTR accuracy, line polygon quality, source context, and clean-text source mixture, using synthetic training data informed by validation-derived HTR error profiles. The resulting case study clarifies the potential and limitations of post-correction in a practical Medieval Greek manuscript workflow and provides a point of comparison for other historical languages and scripts.

2 Related Work

Post-correction is commonly formulated as translation from noisy HTR output to corrected text. For historical documents, prior work shows both the promise and the risk of this formulation: correction can reduce character error rate (CER) and word error rate (WER), but may also introduce errors when applied to already accurate recognition output or to text whose error profile differs from the correction model's training data (Boros et al., 2024; Pavlopoulos et al., 2023). Medieval Greek is a particularly relevant point of comparison. Pavlopoulos et al. (2024) presented a seven-century shared task using actual and synthetic evaluation data for recognized Medieval Greek manuscripts and papyri, comparing ByT5-large with an engineered rule-based system. In contrast, the present single-manuscript study isolates polygon quality, HTR training strategy, context, and source mixture, using validation-derived error profiles to generate synthetic ByT5-small training pairs. Pavlopoulos et al. (2023) further showed that detecting erroneous HTR output before correction can reduce harmful corrections on lines that should have been left unchanged.

Synthetic data are often used when real parallel error data are limited. In grammatical error correction, byte-level models trained first on synthetic errors and then on curated real errors can outperform models based on fixed word or subword vocabularies, especially for noisy or morphologically rich text (Ingólfsdóttir et al., 2023). In OCR/HTR correction, synthetic data can increase training diversity, but prior Medieval Greek results also warn that artificial perturbations may not match real HTR errors (Pavlopoulos et al., 2024). LLM-based correction studies for historical text report mixed results: some handwriting-recognition experiments show prompt-dependent gains, while broader historical OCR/HTR studies find substantial variability and frequent degradation (Principe et al., 2025; Boros et al., 2024; Kanerva et al., 2025; Evangelatos et al., 2026). This motivates the approach used here: instead of relying on prompt-only correction, we derive HTR error profiles from validation data and use them to parameterize a controlled synthetic data generator.

Byte-level, or token-free, models process raw bytes rather than word or subword tokens. ByT5 removes fixed-vocabulary preprocessing, handles arbitrary scripts, and is robust to spelling and input noise, but at the cost of longer sequences (Xue et al., 2022). We use ByT5 because its public encoder-decoder architecture is well suited to noisy-line-to-clean-line correction and has already been used in comparable OCR/HTR correction studies (Löfgren and Dannélls, 2024; Momtaz et al., 2025; de Araújo et al., 2025; Pavlopoulos et al., 2024; Chincholikar et al., 2025).

3 Pipeline Context

3.1 TLD & HTR Setup

Data curation, TLD/HTR training and inference were performed in Transkribus (READ-COOP SCE, 2025). In particular, HTR models of varying accuracy were developed using a fixed page-level training and validation split whose pages had manually corrected line polygons. We used two training

Split	Ground truth			Line polygons
	Pages	Lines	Chars	
Train	36	1507	115,012	Manual
Validation	4	171	13,318	Manual + Automatic
Test	4	165	12,849	Manual + Automatic
Total	44	1843	141,179	–

Table 1: Ground-truth (GT) page split for the Vat. gr. 2228 post-correction experiments.

HTR strategy	Line polygons	CER (%)	WER (%)
Base Model	Manual	2.69 ± 0.20	14.36 ± 0.85
	Automatic	8.90 ± 0.34	36.82 ± 1.20
From Scratch	Manual	3.01 ± 0.17	16.15 ± 0.79
	Automatic	12.54 ± 1.62	44.79 ± 4.38

Table 2: Raw HTR performance on held-out test pages before post-correction. Values are mean $\pm$ standard deviation across three HTR runs.

strategies: *From Scratch* HTR, trained directly on the manuscript ground truth (GT), and *Base Model* HTR, initialized from a Medieval Greek model.[1] In parallel with HTR model training, TLD models were trained on the same dataset splits.

This study builds directly on the Transkribus workflow and models reported by Andersen et al. (2026). Since that study, the dataset has expanded and the models have improved. We use the expanded dataset and improved models here while otherwise following the same workflow.

The developed HTR models were applied to validation and held-out test pages under two line polygon conditions: *Manual* (manually corrected) line polygons and *Automatic* line polygons produced by the best-validation-loss TLD model trained on the same manually corrected training data as the HTR models.[2] The Automatic condition is therefore the direct downstream application of this selected TLD model, whose Transkribus-reported pixel-wise misclassification was 6.43% on train and 5.94% on validation. Recorded settings for TLD/HTR training and inference are given in Appendix D.

The fixed page-level split used for TLD/HTR training is summarized in Table 1. The same split is used for post-correction training and evaluation. In particular, training pages provide clean manuscript GT for synthetic data generation. Validation pages are used for error profiling, calibration of the synthetic data generator, and post-correction model checkpoint selection, while test pages are otherwise reserved for final raw HTR and post-correction output evaluation.

The preceding HTR-development workflow produced three separately trained models for each HTR training strategy. We refer to them as HTR runs and evaluate all of them. Table 2 reports their raw HTR means and standard deviations. Overall, HTR evaluation shows that using automatically detected line polygons substantially increases raw error rates. For Base Model HTR, Automatic increases average test CER from 2.69% to 8.90% and WER from 14.36% to 36.82%. For From Scratch HTR, it increases CER from 3.01% to 12.54% and WER from 16.15% to 44.79%.

3.2 Text Normalization

After text normalization, manuscript GT and HTR outputs are represented as line-level pairs. In the Manual condition, corresponding GT and HTR lines are paired one-to-one. In the Automatic condition, the TLD output and regular manuscript layout yielded equal line counts, permitting one-to-one pairing in reading order.

All manuscript GT, raw HTR outputs, and post-correction predictions undergo the same text-normalization protocol before scoring. The protocol applies Unicode Normalization Form C (NFC), removes zero-width characters, harmonizes punctuation/sign variants, contextually maps Latin to Greek characters, and standardizes whitespace. Table 5 in Appendix A lists the ordered transformations. The same protocol is applied to manuscript GT lines and external pseudo-lines derived from external Greek text before they are used as clean targets for generating synthetic post-correction training pairs.

The character counts in Table 1 and CER/WER values in Table 2 are based on the normalized text.

4 Error Analysis

To understand the validation HTR error profiles and design the synthetic data generator, we first analyze validation HTR outputs from the developed HTR models. These outputs come from the two HTR training strategies, the three HTR runs per strategy, and their application to validation pages under the Manual and Automatic line polygon conditions (Section 3). This section separates character-level

[1] Chrysostomicus I: https://www.transkribus.org/models/chrysostomicus-i

[2] Ground truth is available at https://doi.org/10.5281/zenodo.20718084, while experimental data are available at https://doi.org/10.5281/zenodo.21757657.

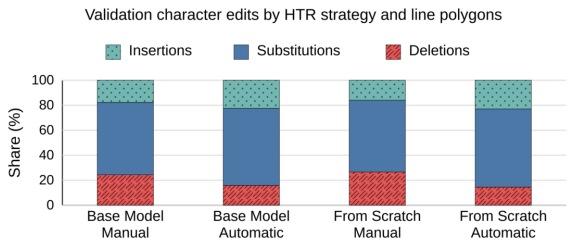

Figure 2: Validation character-level operation proportions, grouped by HTR training strategy and line polygon condition.

diagnostics from line-level diagnostics. Section 5 then uses them to parameterize the synthetic data generator.

4.1 Character-Level Diagnostics

We compute character-level edit operations from the same Levenshtein alignments used for CER. For a GT character c and an HTR character $\hat{c}$, errors are represented as:

$$\begin{aligned} c \to \hat{c}, \quad c \neq \hat{c} \qquad & \text{(substitution)}, \\ c \to \varnothing \qquad & \text{(deletion)}, \\ \varnothing \to \hat{c} \qquad & \text{(insertion)}. \end{aligned}$$

Figure 2 summarizes the proportions of insertions, deletions, and substitutions for validation HTR outputs. Substitutions dominate overall, but the Automatic condition exhibits more severe and structurally complex errors than the Manual condition.

Figure 3 summarizes the most frequent validation edit operations under the Manual and Automatic conditions. The Manual profiles add useful contrast: their edit operations are more concentrated, and the most frequent operation is deletion of a space. Automatic errors are less concentrated, and the most frequent operation is insertion of a space. This supports the interpretation that automatic layout analysis introduces a broader and more segmentation-sensitive error profile across both HTR training strategies.

4.2 Line-Level Diagnostics

Character-level operations alone do not capture the full error structure. In the Automatic condition, recognition errors often reflect interactions between line detection and transcription: spaces may be inserted or deleted, word boundaries may shift, a short continuation from a neighboring visual line may be included, or several adjacent characters may be affected together. We therefore compute whitespace/token-boundary diagnostics and bounded multi-character span edits in addition to single-character edit counts.

Span diagnostics are derived from the Levenshtein backtrace by grouping adjacent non-matching operations until the next matching character. A bounded multi-character span edit is a grouped region with more than one character on either the GT or HTR side and no more than six characters on either side. This excludes very long line-level mismatches while retaining compact multi-character disturbances.

Table 3 reports these diagnostics for validation HTR outputs pooled across the three HTR runs. Automatic has substantially more edits per line, whitespace edits, and bounded multi-character span edits than Manual. The From Scratch Automatic condition is the most severe case: 11.20 character edits per line, a 20.35% whitespace-edit share, and 2.17 bounded multi-character span edits per line. These results show that errors under Automatic are not merely more frequent versions of those under Manual. They are structurally different and more often involve token-boundary and short span-level disturbances. Prior Byzantine Greek studies likewise describe substantial variation in line-level HTR error severity and boundary errors such as mistaken word division and merging, contrasting these cases with isolated character errors (Pavlopoulos et al., 2023, 2024).

5 Post-Correction Method

5.1 Task & Training Data

We formulate post-correction as sequence-to-sequence translation from normalized HTR output to normalized GT. For normalized GT line g_i, let $h_i^{m,p,r}$ denote the corresponding normalized HTR output for HTR training strategy m, line polygon condition p, and HTR run r. A model with parameters θ receives a source string x_i built from the HTR output and predicts g_i by maximizing $p_\theta(g_i \mid x_i)$. The training objective is the standard encoder-decoder negative log-likelihood over target bytes:

$$\mathcal{L}(\theta) = -\sum_i \log p_\theta(g_i \mid x_i).$$

For post-correction model development, we use `google/byt5-small`[3], a byte-level T5 model (Xue et al., 2022).

[3] https://huggingface.co/google/byt5-small/

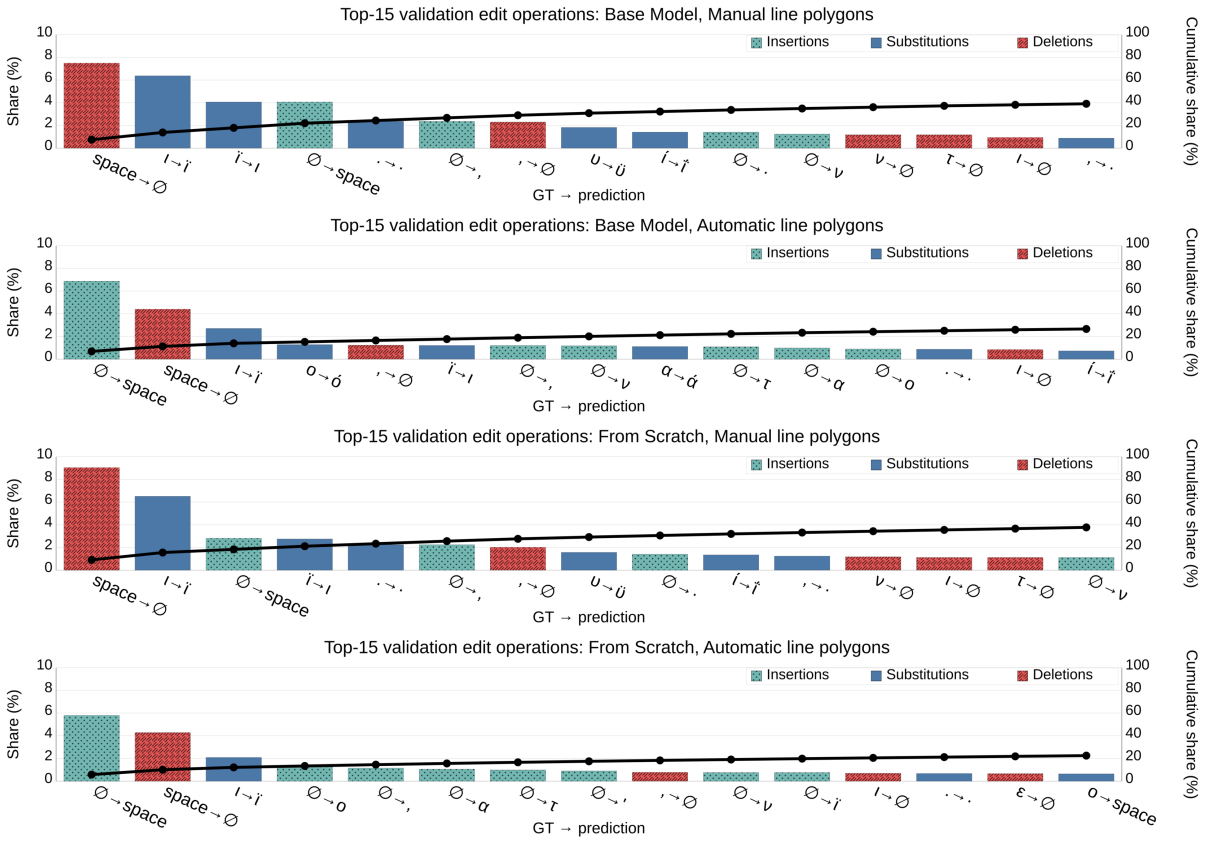

Figure 3: The 15 most frequent validation edit operations by HTR training strategy and line polygon condition.

HTR strategy	Line polygons	Edit diagnostics			Whitespace edits		
		Avg. edits/line	Whitespace edits (%)	Avg. spans/line	Insertions	Substitutions	Deletions
Base Model	Manual	3.30	14.99	0.44	69	58	127
	Automatic	8.06	18.32	1.41	284	291	182
From Scratch	Manual	3.47	14.76	0.42	50	52	161
	Automatic	11.20	20.35	2.17	333	590	246

Table 3: Validation structural diagnostics pooled across three HTR runs. Avg. edits/line is the mean character-edit count, Whitespace edits (%) is the share of edit operations involving whitespace, and Avg. spans/line is the mean bounded multi-character span-edit count. Counted spans contain at most six GT and six HTR characters.

We compare two source context settings, *Current* (x_i^{cur}) and *Neighboring* (x_i^{ctx}):

$$
\begin{aligned}
x_i^{\text{cur}} &= \langle \text{CUR} \rangle h_i^{m,p,r} \langle / \text{CUR} \rangle, \\
x_i^{\text{ctx}} &= \langle \text{PREV} \rangle h_{i-1}^{m,p,r} \langle / \text{PREV} \rangle \\
&\quad \langle \text{CUR} \rangle h_i^{m,p,r} \langle / \text{CUR} \rangle \\
&\quad \langle \text{NEXT} \rangle h_{i+1}^{m,p,r} \langle / \text{NEXT} \rangle.
\end{aligned}
$$

The angle-bracketed labels are literal source tags. Thus, x_i^{cur} and x_i^{ctx} are tagged source strings containing the current HTR line alone or together with its neighboring HTR lines. The target in both settings is the current normalized GT line g_i. In the Neighboring setting, the neighboring lines serve only as input context. At validation and test time, the tags enclose actual HTR lines, with neighboring lines taken from the same page and HTR run. During training, they enclose the synthetic corruptions described in Section 5.2. At page boundaries, missing neighboring lines are represented by the empty string.

Using train-split HTR output directly would be misleading because train HTR is nearly perfect. In the pooled train split, Base Model HTR has only 70 character edits across 4,521 run-level line instances, and even the From Scratch strategy is far cleaner on train pages than on validation or test pages. A post-correction model trained on those pairs would mostly learn to copy its input and would give little signal about the validation errors documented in Figure 3 and Table 3. At

the same time, the validation data show systematic mistakes, especially under the Automatic condition. We therefore use validation HTR to estimate error characteristics, but use clean manuscript training GT and external Greek text as the source material from which noisy training pairs are generated.

We consider two source mixtures: *Manuscript-Heavy* and *External-Heavy*. Both combine manuscript training GT with pseudo-lines derived from a pool of external Greek text, but in reversed proportions. The external passages were provided by the authors of Yousef et al. (2022). We divide the passages into pseudo-lines, filter them to resemble manuscript training GT in length, script composition, and formatting, and sample from the resulting pool. We exclude pseudo-lines that exactly match any complete normalized manuscript GT line. Appendix B documents the source repositories, processing, sampling, and lexical-overlap analysis.

For each HTR strategy and line polygon condition, validation HTR errors pooled across the three HTR runs define one error profile $(\pi_{m,p}, \phi_{m,p})$. The four profiles parameterize the synthetic data generator and are crossed with source context and source mixture to define the post-correction configurations.

5.2 Synthetic Error Calibration & Generation

The goal of the synthetic data generator is to create training pairs that reflect the validation HTR error characteristics while still using clean manuscript and external text as targets. We therefore control not only which edit types are sampled, but also how severe the generated errors are. Line-level CER provides an interpretable severity axis. For each HTR strategy m and line polygon condition p, we define the pooled validation set as

$$D_{\text{val}}^{m,p} = \{(g_i, h_i^{m,p,r})\}_{i,r}.$$

For each normalized validation pair $(g, h) \in D_{\text{val}}^{m,p}$—shorthand for a GT line g_i and its corresponding HTR output $h_i^{m,p,r}$—we define the line-level severity as

$$s(g, h) = \frac{d_{\text{Lev}}(g, h)}{|g|}, \tag{1}$$

where d_{Lev} denotes Levenshtein edit distance and $|g|$ is the number of GT characters. Equation (1) is used only within the synthetic calibration and generation procedure. Writing $s = s(g, h)$, the empirical severity distribution $\pi_{m,p}$ is computed over five bins. The *identity* bin has $s = 0$. Among validation pairs with $s > 0$, t_{50}, t_{90}, and t_{95} are the 50th, 90th, and 95th percentiles of line-level CER. They delimit *low* as $0 < s \leq t_{50}$, *medium* as $t_{50} < s \leq t_{90}$, *high* as $t_{90} < s \leq t_{95}$, and *extreme* as $s > t_{95}$. For the same validation pairs, the Levenshtein backtraces identify substitutions, insertions, and deletions; their pooled frequencies and structural diagnostics define an edit profile $\phi_{m,p}$. Thus, $\pi_{m,p}$ determines how often each severity bin is requested, whereas $\phi_{m,p}$ determines which edit patterns are used to construct candidates. Figure 3 and Table 3 summarize the main components of $\phi_{m,p}$, while Appendix C gives the exact bin thresholds and $\pi_{m,p}$ values.

The calibration and generation procedure is as follows:

1. **Pool validation HTR–GT pairs.** Pool the pairs across the three HTR runs for one HTR strategy m and line polygon condition p.
2. **Estimate the error profile.** Estimate the five-bin severity distribution $\pi_{m,p}$ and edit profile $\phi_{m,p}$.
3. **Select the local-edit multiplier.** For each error profile and source mixture, select a multiplier λ that scales the probabilities of character, whitespace, and short-span edits in $\phi_{m,p}$. Larger values make such edits more likely and tend to increase corruption severity. Calibration selects λ to minimize the weighted difference between a realized synthetic severity distribution $\hat{\pi}_\lambda$ and the corresponding validation distribution $\pi_{m,p}$.
4. **Draw requested severity bins.** For each clean line selected from a source mixture, independently draw a bin from $\pi_{m,p}$ for that line and each available non-empty neighboring line.
5. **Generate corruptions within the requested bins.** Leave an *identity* request unchanged. Otherwise, use the validation-derived local edits in $\phi_{m,p}$, with their probabilities scaled by the selected λ, to generate candidates. For *high*/*extreme* requests, the candidate set may also include candidates guided by the numerical error properties of one validation HTR–GT pair. Retain the first generated candidate that falls within the requested bin or, if none does, the candidate with the smallest CER distance to that bin.

6. **Form Current and Neighboring training pairs.** Form x_i^{cur} from the corrupted line alone or x_i^{ctx} from the corrupted line and available neighboring lines. In both cases, use the clean line g_i as the target.

Appendix C follows this sequence in more detail and assesses how closely the generated training corruptions reproduce the validation-derived distribution.

6 Experimental Setup

Post-correction setup & compute. The experimental grid crosses HTR training strategy (Base Model vs. From Scratch), line polygon condition (Manual vs. Automatic), source context (Current vs. Neighboring), and source mixture. For each validation-derived error profile, the Manuscript-Heavy 75%/25% mixture uses 37,500 examples sampled with replacement from manuscript training GT and 12,500 pseudo-lines sampled without replacement from the filtered external pool; the External-Heavy 25%/75% mixture reverses these counts. Each configuration is trained on 50,000 synthetic pairs; within each error profile and source mixture, the context variants reuse the same corruptions and targets, and their sources differ only in the tagged lines included. The full $2 \times 2 \times 2 \times 2$ grid therefore contains 16 post-correction configurations, each fine-tuned once.

All 16 configurations used the same recorded training settings. Validation and checkpoint selection used 513 real HTR–GT pairs: 171 GT lines under each of three HTR runs. Because ByT5 uses byte-level tokenization, sequence limits are measured in tokenizer tokens rather than characters. During fine-tuning, sources and targets were capped at 1,024 and 256 tokens, respectively. Content beyond these limits was truncated. Training and validation batches contained up to 8 examples. Validation batches were used only for scoring.

Optimization used fused AdamW at 5×10^{-5} with a linear schedule and no warmup. BF16 mixed precision and TF32 were enabled for more efficient GPU execution. Seed 1337 controlled randomness throughout the post-correction pipeline. Models were trained for three epochs, evaluated and saved after each epoch, and the checkpoint with the lowest validation loss was retained.

At inference, sources were again capped at 1,024 tokens and processed in batches of up to 16. Generation stopped when the model produced an end-of-sequence token or after 256 newly generated tokens, bounding output length and decoding cost. Training and inference used PyTorch and Hugging Face Transformers.

For each configuration, the selected checkpoint was evaluated separately on the outputs of the three HTR runs. Reported test values average these three evaluations. They therefore reflect different HTR-run outputs, not repeated ByT5 fine-tuning. Post-correction training and evaluation used one NVIDIA GeForce RTX 5090 GPU (32 GB VRAM), an AMD EPYC 9224 CPU (24 cores), and 64 GB RAM. Across all 16 configurations, logged post-correction pipeline-stage durations summed to 10.21 hours on this machine, excluding manual curation and Transkribus-side TLD and HTR processing.

Evaluation metrics. For a collection of evaluated normalized GT/prediction line pairs $\mathcal{S} = \{(g_i, \hat{g}_i)\}$ within one HTR run, CER sums character edits over all lines in the split and divides by the total number of GT characters; WER does the same with whitespace-delimited word edits and GT words. Here $\hat{g}_i$ denotes the evaluated output, either raw HTR or post-corrected text:

$$\mathrm{CER}(\mathcal{S}) = \frac{\sum_{(g_i,\hat{g}_i)\in\mathcal{S}} d_{\mathrm{Lev}}(g_i, \hat{g}_i)}{\sum_{(g_i,\hat{g}_i)\in\mathcal{S}} |g_i|},$$

$$\mathrm{WER}(\mathcal{S}) = \frac{\sum_{(g_i,\hat{g}_i)\in\mathcal{S}} d_{\mathrm{Lev}}(\tau(g_i), \tau(\hat{g}_i))}{\sum_{(g_i,\hat{g}_i)\in\mathcal{S}} |\tau(g_i)|}.$$

Here, $\tau(\cdot)$ denotes whitespace tokenization. The line-level CER used for synthetic severity calibration and candidate selection is the single-pair computation in Equation (1). Thus, lines contribute to split-level scores in proportion to their GT length rather than receiving equal weight; reported configuration values then average the three run-level scores. For a metric $E \in \{\mathrm{CER}, \mathrm{WER}\}$, relative error reduction is reported as:

$$\mathrm{reduction}(E) = \frac{E_{\mathrm{raw}} - E_{\mathrm{post}}}{E_{\mathrm{raw}}}.$$

Positive values indicate improvement over raw HTR, while negative values indicate degradation.

7 Results

7.1 Overall Post-Correction Effects

Table 4 reports test-set performance for all configurations, comparing raw HTR with post-corrected

Source mixture	HTR strategy	Line polygons	Context	CER (%)			WER (%)		
				Raw	Post	Red.	Raw	Post	Red.
Manuscript-Heavy	Base Model	Manual	Current	2.69	2.71	-0.73	14.36	13.82	3.76
			Neighboring	2.69	2.65	1.54	14.36	13.79	4.00
		Automatic	Current	8.90	8.52	4.25	36.82	31.65	14.09
			Neighboring	8.90	8.48	4.74	36.82	31.08	15.61
	From Scratch	Manual	Current	3.01	3.03	-0.72	16.15	14.94	7.51
			Neighboring	3.01	2.98	0.99	16.15	14.75	8.67
		Automatic	Current	12.54	11.85	5.23	44.79	36.99	17.22
			Neighboring	12.54	11.70	6.49	44.79	36.25	18.86
External-Heavy	Base Model	Manual	Current	2.69	2.74	-2.12	14.36	14.02	2.36
			Neighboring	2.69	2.69	0.02	14.36	13.68	4.76
		Automatic	Current	8.90	8.51	4.38	36.82	31.49	14.53
			Neighboring	8.90	8.44	5.07	36.82	30.87	16.18
	From Scratch	Manual	Current	3.01	2.95	1.82	16.15	14.58	9.73
			Neighboring	3.01	2.93	2.62	16.15	14.34	11.19
		Automatic	Current	12.54	11.55	7.67	44.79	36.50	18.28
			Neighboring	12.54	11.55	7.73	44.79	36.60	18.06

Table 4: Test-set post-correction performance by source mixture, HTR strategy, line polygon condition, and source context. Values are means across three HTR runs; Red. is the mean run-level relative reduction.

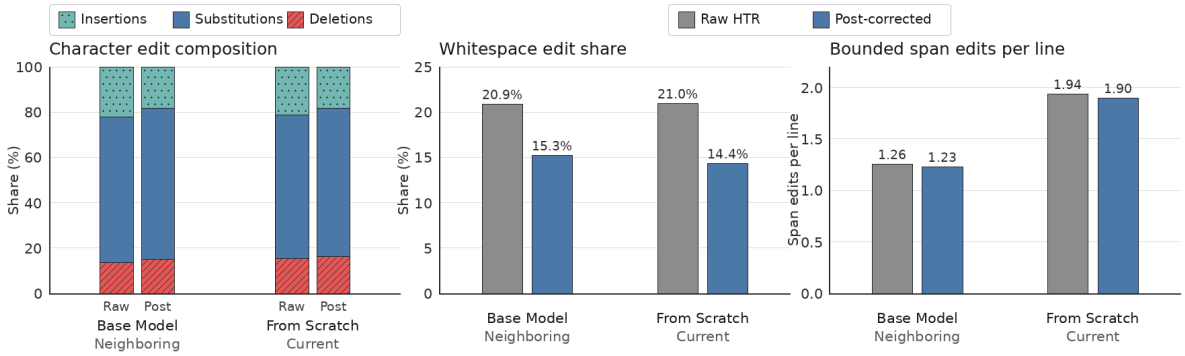

Figure 4: Test diagnostics under the External-Heavy Automatic condition, using the lowest-WER context setting for each HTR strategy (Base Model/Neighboring and From Scratch/Current), pooled across three HTR runs. Left to right: character-operation shares, whitespace-edit share, and bounded multi-character span edits per line.

output under the same inputs. Across conditions, post-correction reduces WER more consistently than CER, consistent with the token-boundary and segmentation-sensitive errors observed for Automatic output.

The External-Heavy mixture is strongest in most Automatic conditions and gives the larger reduction in six of eight comparisons for each metric. For Base Model HTR with Neighboring context, WER decreases from 36.82% to 30.87%, corresponding to a 16.18% mean run-level relative WER reduction. For From Scratch HTR with Current context, WER decreases from 44.79% to 36.50%, while CER decreases from 12.54% to 11.55%. Manual line polygons show smaller and less consistent effects across configurations, including CER degradation in some settings, because raw HTR is already much better under manually corrected segmentation. Thus, in this experiment external text diversity is most useful for noisy automated-pipeline output. It does not make the Automatic condition competitive with manually corrected line polygons in absolute error, but it helps recover part of the error introduced by automation. Figure 4 compares raw and post-corrected error profiles under the External-Heavy Automatic condition, using the context setting with the lowest WER for each HTR strategy: Neighboring for Base Model HTR and Current for From Scratch HTR. Insertion shares fall from 21.96% to 18.19% and from 21.09% to 18.14%, respectively, while substitutions account for a correspondingly larger share of edit operations. Whitespace-edit shares likewise fall from 20.94% to 15.27% and from 21.01% to 14.37%; bounded multi-character span edits change only from 1.26 to 1.23 and from 1.94 to 1.90 per line. Thus, the clearest structural change is reduced whitespace-related error, not a comparable reduction in bounded multi-character span edits.

7.2 Source-Context Ablation

The Neighboring context is most beneficial in the Base Model Automatic condition, where it improves both CER and WER relative to Current context. For From Scratch Automatic, the pattern is mixed: Neighboring context gives better CER reductions, but Current context gives slightly better WER with the External-Heavy mixture. Overall, these results show that the benefit of Neighboring context varies across conditions.

8 Discussion

8.1 Post-Correction as Pipeline Intervention

Manual line polygons give the best absolute HTR quality, but require time-consuming layout correction and leave little room for post-correction. In some settings, the model even over-corrects. Automatic output is worse, but is also where segmentation artifacts accumulate and post-correction has a clearer effect. Within this case study, post-correction is therefore most appropriate as a robustness layer for noisy automated HTR output, not as a replacement for better TLD or HTR.

8.2 Synthetic Error Modeling

The validation-derived corruption model pools all three validation HTR runs and reflects character confusions, whitespace edits, bounded multi-character span edits, and line-level severity. Calibration nevertheless yields fewer bounded multi-character span edits per line and larger whitespace-edit shares in synthetic data than in validation (Table 7), illustrating how difficult it is for text-level corruptions alone to simulate more severe errors caused by layout or line contamination.

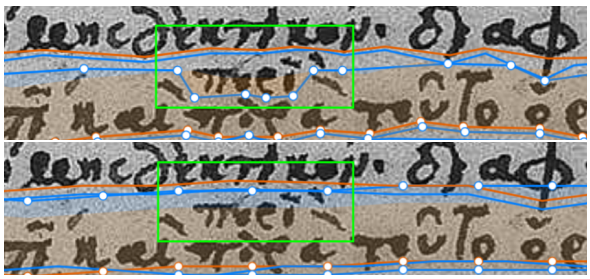

Figure 5: Scribal correction in Vat. gr. 2228. Manually corrected polygons (top) exclude it, whereas Automatic TLD polygons (bottom) include it and add segmentation noise to the HTR input.

9 Conclusion

This controlled single-manuscript study shows that ByT5-small post-correction trained on synthetic data generated from validation-derived HTR error profiles can recover part of the error introduced by automatic line detection. Gains are most consistent for WER, and External-Heavy performs best in most test conditions, but manually corrected line polygons remain substantially better in absolute terms. These findings clarify the trade-off between upstream manual layout correction and downstream text correction in this workflow.

10 Limitations

The evaluation uses the same small, fixed set of held-out pages across all configurations, so absolute estimates may be sensitive to the textual and layout characteristics of those particular pages.

The validation set supplies the error profiles, synthetic calibration, and checkpoint selection, while the test set is otherwise reserved for final reporting. Relying on a single validation split for these development decisions may tailor the corruption model to that split's particular error distribution. Moreover, although each HTR strategy has three separately trained models, each ByT5 configuration was fine-tuned only once. Reported variation therefore reflects differences among HTR-run outputs rather than repeated post-correction training. Because variability across additional seeds and folds is not estimated, robustness to post-correction training randomness and alternative page splits remains unknown. Evidence that the validation-specific corruption model transfers to other HTR error distributions is likewise limited.

The experiments cover one comparatively regular manuscript, Vat. gr. 2228, and one post-correction architecture, ByT5-small. Transfer to other manuscripts, scripts, layouts, or model architectures remains open.

Finally, the approach corrects text only after HTR. It does not repair line polygons. Automatic errors may arise from included scribal corrections (Figure 5) or from missing text that the HTR model never saw. Such line-structure interactions cannot be fully solved by text-only post-correction.

11 Acknowledgements

This work was supported by the Carlsberg Foundation (grant CF24-1999). We thank Byron MacDougall and Ugo Valori for data curation.

References

Nicklas Sindlev Andersen, Byron MacDougall, Tariq Yousef, and Aglae Pizzone. 2026. From manuscript to model: Developing HTR for Medieval Greek. In *Proceedings of the Fourth Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA) @ LREC 2026*, pages 139–151. European Language Resources Association.

Emanuela Boros, Maud Ehrmann, Matteo Romanello, Sven Najem-Meyer, and Frédéric Kaplan. 2024. Post-correction of historical text transcripts with large language models: An exploratory study. In *Proceedings of the 8th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2024)*, pages 133–159, St. Julians, Malta. Association for Computational Linguistics.

Kartik Chincholikar, Shagun Dwivedi, Kaushik Gopalan, and Tarinee Awasthi. 2025. A case study of handwritten text recognition from early modern Sanskrit manuscripts. In *Computational Sanskrit and Digital Humanities - World Sanskrit Conference 2025*, pages 52–69, Kathmandu, Nepal. Association for Computational Linguistics.

Sávio Santos de Araújo, Byron Leite Dantas Bezerra, and Arthur Flor de Sousa Neto. 2025. A proposal of post-OCR spelling correction using monolingual byte-level language models. In *Proceedings of the 2025 ACM Symposium on Document Engineering*, DocEng '25, pages 1–4, New York, NY, USA. Association for Computing Machinery.

Andreas Evangelatos, Konstantinos Palaiologos, Basilis Gatos, Panagiotis Kaddas, Aikaterini Christopoulou, Vassilis Katsouros, and Andreas Kakridis. 2026. Using LLMs for improving the OCR accuracy of Old Greek handwritten documents. In *New Trends in Theory and Practice of Digital Libraries*, volume 2694 of *Communications in Computer and Information Science*, pages 100–109, Cham. Springer Nature Switzerland.

Svanhvít Lilja Ingólfsdóttir, Petur Ragnarsson, Haukur Jónsson, Haukur Simonarson, Vilhjalmur Thorsteinsson, and Vésteinn Snæbjarnarson. 2023. Byte-level grammatical error correction using synthetic and curated corpora. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7299–7316, Toronto, Canada. Association for Computational Linguistics.

Jenna Kanerva, Cassandra Ledins, Siiri Käpyaho, and Filip Ginter. 2025. OCR error post-correction with LLMs in historical documents: No free lunches. In *Proceedings of the Third Workshop on Resources and Representations for Under-Resourced Languages and Domains (RESOURCEFUL-2025)*, pages 38–47, Tallinn, Estonia. University of Tartu Library, Estonia.

Salvatore Lilla. 1985. *Codices Vaticani Graeci: Codices 2162–2254 (Codices Columnenses)*. Bibliotheca Apostolica Vaticana, Vatican City.

Viktoria Löfgren and Dana Dannélls. 2024. Post-OCR correction of digitized Swedish newspapers with ByT5. In *Proceedings of the 8th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2024)*, pages 237–242, St. Julians, Malta. Association for Computational Linguistics.

Yahya Momtaz, Lorenza Laccetti, and Guido Russo. 2025. Modular pipeline for text recognition in early printed books using Kraken and ByT5. *Electronics*, 14(15):3083.

John Pavlopoulos, Vasiliki Kougia, Esteban Garces Arias, Paraskevi Platanou, Stepan Shabalin, Konstantina Liagkou, Emmanouil Papadatos, Holger Essler, Jean-Baptiste Camps, and Franz Fischer. 2024. Challenging error correction in recognised Byzantine Greek. In *Proceedings of the 1st Workshop on Machine Learning for Ancient Languages (ML4AL 2024)*, pages 1–12, Hybrid in Bangkok, Thailand and online. Association for Computational Linguistics.

John Pavlopoulos, Vasiliki Kougia, Paraskevi Platanou, and Holger Essler. 2023. Detecting erroneously recognized handwritten Byzantine text. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 7818–7828, Singapore. Association for Computational Linguistics.

Jean Pool Pereyra Principe, Andreas Fischer, and Anna Scius-Bertrand. 2025. Post-correction of handwriting recognition using large language models. In Wray Buntine, Morten Fjeld, Truyen Tran, Minh-Triet Tran, Binh Huynh Thi Thanh, and Takumi Miyoshi, editors, *Information and Communication Technology*, volume 2351 of *Communications in Computer and Information Science*, pages 106–118. Springer Nature Singapore, Singapore.

READ-COOP SCE. 2025. Transkribus. URL: `https://www.transkribus.org`. Accessed: 2026-08-12.

Christian Walz, editor. 1835. *Rhetores Graeci*, volume 2. J. G. Cotta, Stuttgart and Tübingen.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Tariq Yousef, Chiara Palladino, Farnoosh Shamsian, Anise d'Orange Ferreira, and Michel Ferreira dos Reis. 2022. An automatic model and Gold Standard for translation alignment of Ancient Greek. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 5894–5905, Marseille, France. European Language Resources Association.

A Text Normalization

The same text-normalization protocol is applied symmetrically to manuscript GT, raw HTR output, and post-correction predictions before scoring, and to text used in post-correction training-data preparation. It normalizes Unicode composition to NFC and applies the ordered transformations in Table 5.

B External Clean Text & Sampling

The external corpus was provided by Yousef et al. (2022). It contains Ancient Greek material drawn from five source collections used in that study, with annotations and other ancillary data stripped: Perseus canonical Greek[4], First1KGreek[5], PROIEL[6], Perseus Treebank Data[7], and Gorman Greek Dependency Trees.[8] We call the resulting plain Ancient Greek content *external text* and the short line-like chunks derived from it *external pseudo-lines*.

To construct the clean-text inputs shared by the 16 post-correction configurations, the pipeline proceeds from the external text to pseudo-line candidates, a filtered external pool, and finally two source mixtures. Normalized manuscript training GT supplies both the manuscript component of those mixtures and a reference for selecting external pseudo-lines with broadly comparable length, script composition, and formatting. Overall, the procedure has three steps:

1. **Form pseudo-line candidates.** Apply the text-normalization protocol in Appendix A, split the external text into sentence-like content units, remove duplicate content units before chunking, and divide the remainder into consecutive chunks of at most 12 words. This limit is the median line length of the 1,507 lines in manuscript training GT (Table 1).

2. **Construct the filtered external pool.** Use manuscript training GT as a reference for excluding clear length and script-composition outliers while allowing the external pseudo-lines to vary beyond the manuscript's central range. For length, use the 1st and 99th percentiles of manuscript word and character counts as broad empirical anchors and expand that interval with fixed tolerances to admit nearby lengths. For character composition, use a relaxed manuscript-derived minimum for the share of Greek-script characters, including combining diacritics, and configured maximum shares for Latin characters, digits, and punctuation. Apply these filters, remove candidates containing any configured exclusion character,[9] and exclude exact matches to complete normalized GT lines from the training, validation, or test split (this complete-line comparison removed seven candidate lines). Then deduplicate the retained pseudo-lines. Together, these operations reduce 1,390,010 candidates to a pool of 813,619 unique pseudo-lines.

3. **Construct the source mixtures.** Sample pseudo-lines without replacement from the filtered external pool and lines from manuscript training GT with replacement. External-Heavy contains 37,500 external and 12,500 manuscript lines. Manuscript-Heavy reverses these counts. Each mixture therefore contains 50,000 clean lines.

For reproducibility, the screening rule, together with the 12-word chunk limit, retained pseudo-lines with 6–12 words and 38–111 characters. Among non-whitespace characters, retained candidates contained at least 81.05% Greek-script characters, including combining diacritics, and at most 3% Latin characters, 3% digits, and 20% punctuation.

To quantify shared vocabulary after this processing, a token is one whitespace-delimited occurrence and a token form is its distinct exact string. For example, the illustrative sequence `alpha alpha beta` contains three token occurrences but two token forms. Because token forms are compared as exact whitespace-delimited strings and punctuation is not removed, a word without following punctuation (e.g., `alpha`) and the same word followed by a comma (e.g., `alpha,`) count as different forms. Manuscript training GT contains 18,619 occurrences and 5,999 forms, while the filtered external pool contains 8,920,871 occurrences and 703,742 forms. The corpora share 3,012 distinct forms. As Figure 6 shows, these represent 50.21% of the manuscript vocabulary but only 0.43% of

[4] https://github.com/PerseusDL/canonical-greekLit

[5] https://github.com/OpenGreekAndLatin/First1KGreek

[6] https://github.com/proiel/proiel-treebank

[7] https://github.com/PerseusDL/treebank_data

[8] https://github.com/vgorman1/Greek-Dependency-Trees

[9] Angle, square, or curly brackets; forward slashes or backslashes; and `#`, `@`, `$`, `=`, or `|`.

Step	Input class	Characters (code points)	Normalized output
1	Zero-width characters	Invisible (U+200B, U+200C, U+200D, U+FEFF)	Remove
2	Apostrophe-like marks	’ (U+02BC), ՚ (U+055A), ᾽ (U+1FBD), ι (U+1FBF), ‘ (U+2018), ’ (U+2019), ‛ (U+201B), ′ (U+2032)	ASCII apostrophe ' (U+0027)
3a	Greek question mark	; (U+037E)	ASCII semicolon ; (U+003B)
3b	Ano teleia	· (U+0387)	Middle dot · (U+00B7)
4a	Hyphen/dash/minus	‐ (U+2010), ‑ (U+2011), ‒ (U+2012), – (U+2013), ― (U+2212)	ASCII hyphen - (U+002D)
4b	Curly double quotes	“ (U+201C), ” (U+201D), „ (U+201E)	ASCII double quote " (U+0022)
5	Latin/Greek omicron	Latin o (U+006F) or O (U+004F) when immediately adjacent to a Greek character	Greek ο (U+03BF) or Ο (U+039F)
6	Unicode whitespace	Invisible spacing characters	ASCII space (U+0020); trim boundaries and collapse repetitions

Table 5: Ordered text-normalization transformations after NFC. Lettered rows share a step; glyphs are paired with code points, while invisible inputs are listed by code point.

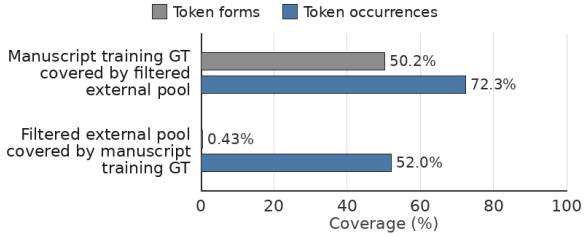

Figure 6: Directional lexical coverage between normalized manuscript training GT and the filtered external pool, shown for exact token forms and their occurrences.

the much larger external-pool vocabulary. Nevertheless, the shared forms account for 72.28% of manuscript token occurrences and 52.03% of external-pool token occurrences, indicating substantial overlap among frequently occurring forms. The manuscript side of this comparison uses normalized training GT only, while the external side is the filtered pool described above. The two source mixtures provide the clean-text input to the calibration and corruption procedure described next.

C Error Profiles, Calibration & Generation

The detailed procedure follows the same six steps as Section 5.2:

1. **Pool validation HTR–GT pairs.** For each HTR strategy m and line polygon condition p, pool the validation pairs across the three HTR runs.
2. **Estimate the error profile.** Use the pooled pairs to estimate $(\pi_{m,p}, \phi_{m,p})$, where $\pi_{m,p}$ is the five-bin severity distribution and $\phi_{m,p}$ is the edit profile. The bins are *identity* ($s = 0$), *low* ($0 < s \leq t_{50}$), *medium* ($t_{50} < s \leq t_{90}$), *high* ($t_{90} < s \leq t_{95}$), and *extreme* ($s > t_{95}$), where t_{50}, t_{90}, and t_{95} are the 50th, 90th, and 95th percentiles of positive validation line-level CER. Table 6 reports the profile-specific thresholds and distributions.
3. **Select the local-edit multiplier.** Pair each error profile with each 50,000-line source mixture from Appendix B. For a given pairing, calibration holds three validation-derived inputs fixed: the severity distribution $\pi_{m,p}$, which determines how often each severity bin is requested; the edit profile $\phi_{m,p}$, which supplies the local edit patterns; and numerical error properties of individual validation HTR–GT pairs, which may guide some *high* and *extreme* candidates. Calibration selects only the multiplier λ for that pairing.

   Calibration subset. Uniformly sample 2,000 clean lines without replacement from the relevant source mixture and draw one requested severity bin from $\pi_{m,p}$ for each sampled line. Hold both the lines and requested bins fixed when comparing candidate values of λ.

   Candidate search. For every λ in the prespecified grid

   $$\Lambda = \{0.75, 1, 1.25, 1.5, 2, 2.5, 3, 4, 6\},$$

   apply the bounded candidate search used in Step 5. An *identity* request leaves the clean line unchanged. For every other request, each of at most six attempts uses $\phi_{m,p}$ to produce one candidate by applying sampled character, whitespace, and short-span edits within the clean line. If an applicable local edit has probability q, the multiplier gives

   $$q_\lambda = \min(1, \lambda q).$$

HTR strategy	Line polygons	t_{50}	t_{90}	t_{95}	Identity	Low	Medium	High	Extreme
Base Model	Manual	0.0375	0.0949	0.1160	10.33	45.42	35.28	4.48	4.48
Base Model	Automatic	0.0759	0.2393	0.3475	3.51	48.34	38.40	4.87	4.87
From Scratch	Manual	0.0395	0.1026	0.1200	10.14	46.00	34.89	4.48	4.48
From Scratch	Automatic	0.0897	0.3723	0.5377	2.92	48.93	38.40	4.87	4.87

Table 6: Validation-derived severity-bin thresholds and distributions (513 validation HTR–GT pairs per error profile). Thresholds are CER proportions among positive-CER lines; the Identity–Extreme columns report $\pi_{m,p}$ in percent.

Thus, λ scales how often the character, whitespace, and short-span edit mechanisms in $\phi_{m,p}$ are invoked, while the validation-derived relative probabilities within event groups remain unchanged. For a *high* or *extreme* request, the candidate set may also include an *error-template-guided candidate* whose generation is not scaled by λ. Such a candidate is guided by the numerical properties of one validation HTR–GT pair in the requested bin—its CER, edit-type proportions, whitespace-edit share, and edit dispersion. No validation text is copied: characters and edit locations are generated anew for the clean line from the validation-derived edit patterns in $\phi_{m,p}$.

Multiplier selection. Rescore each candidate with Equation (1), retaining the first candidate in the requested bin or, if none reaches the bin within six attempts, the candidate with the smallest CER distance to that bin. Let $\hat{\pi}_\lambda(b)$ be the resulting proportion of calibration outputs in bin b. Select the multiplier that minimizes

$$L(\lambda) = \sum_b w_b \left| \hat{\pi}_\lambda(b) - \pi_{m,p}(b) \right|,$$

where b ranges over the five severity bins and the weights are $(1, 1, 1, 2, 2)$ from *identity* through *extreme*. Giving greater weight to *high* and *extreme* prevents deviations in these less frequent bins from being dominated by the more common bins. Ties favor the smaller multiplier.

Calibration does not train another model or alter the validation-derived error profile.

For the reported configurations, calibration selected $\lambda = 1.0$ for Manuscript-Heavy Manual under both HTR strategies and $\lambda = 0.75$ for the other six error-profile–source-mixture pairings.

4. **Draw requested severity bins.** For each clean line in the complete source mixture and each available non-empty neighboring line, independently draw a bin from $\pi_{m,p}$.
5. **Generate corruptions within the requested bins.** Leave an *identity* request unchanged. Otherwise, repeat the bounded candidate search from Step 3 using the selected λ. For *high* and *extreme* requests, the candidate set may again include error-template-guided candidates. Retain the first generated candidate that falls within the requested bin or, after six unsuccessful attempts, the candidate with the smallest CER distance to that bin. Error-template-guided candidates allow the validation-derived CER target to vary multiplicatively by up to $\pm 8\%$, while fixed safeguards bound the total number and dispersion of edits. These safeguards keep the generated corruption proportional to the new line's length and prevent it from being divided across too many separate regions.
6. **Form Current and Neighboring training pairs.** Form x_i^{cur} from the corrupted line alone or x_i^{ctx} from that line and the available corrupted neighboring lines, using the clean line g_i as the target. Current and Neighboring reuse the same corruptions and targets; only the tagged corrupted lines included in the source differ.

Figure 7 compares $\pi_{m,p}$ with the full-output $\hat{\pi}_\lambda$, estimated separately for each complete set of 50,000 generated pairs. It is a post-generation diagnostic. λ is selected only on the fixed 2,000-line calibration subset. Table 7 adds all-line CER and structural diagnostics. Matching bin proportions need not imply structural agreement.

D Transkribus HTR & TLD Settings

Following Andersen et al. (2026), Table 8 reports the HTR training and TLD training/inference settings used. Most settings retained their Transkribus platform defaults. Transkribus documents HTR model setup and training,[10] TLD training,[11] and TLD inference settings.[12]

[10] Model Setup and Training: https://help.transkribus.org/model-setup-and-training

[11] Baselines Models: https://help.transkribus.org/baselines-models

[12] Layout Configuration Settings: https://help.transkribus.org/advanced-layout-configuration-settings

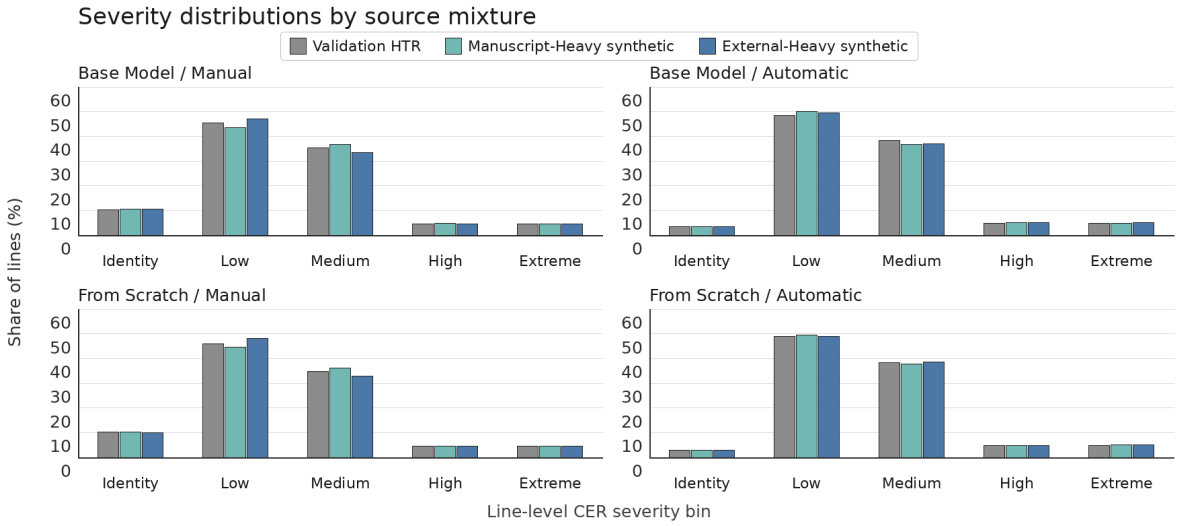

Figure 7: Validation severity distribution $\pi_{m,p}$ and realized full-output distributions $\hat{\pi}_{\lambda}$ for both source mixtures, using the multiplier selected for each error-profile–source-mixture pairing.

Source mixture	HTR strategy	Line polygons	CER (%)		CER 95th (%)		Whitespace edits (%)		Avg. spans/line	
			Val.	Synth.	Val.	Synth.	Val.	Synth.	Val.	Synth.
Manuscript-Heavy	Base Model	Manual	4.24	4.34	11.54	11.25	14.99	17.57	0.44	0.35
		Automatic	10.34	9.64	34.42	33.33	18.32	24.58	1.41	1.07
	From Scratch	Manual	4.46	4.55	11.63	11.59	14.76	17.80	0.42	0.35
		Automatic	14.38	12.71	52.87	53.97	20.35	29.19	2.17	1.42
External-Heavy	Base Model	Manual	4.24	4.15	11.54	11.25	14.99	17.71	0.44	0.31
		Automatic	10.34	9.76	34.42	34.09	18.32	24.35	1.41	1.01
	From Scratch	Manual	4.46	4.33	11.63	11.69	14.76	17.91	0.42	0.32
		Automatic	14.38	12.71	52.87	53.54	20.35	28.89	2.17	1.32

Table 7: Validation HTR and realized synthetic diagnostics. CER is micro-averaged over characters, and CER 95th pct. includes unchanged lines. Whitespace edits (%) is the share of edit operations involving whitespace, and Avg. spans/line is the mean bounded multi-character span-edit count, restricted to spans with at most six GT and six HTR characters.

Setting	Recorded value
HTR training	
Training cycles	250
Early stopping	25
Use existing line polygons	Yes
Augmentation method	None
Image Type	Originals
Convert images to black & white	Yes
TLD training	
Training cycles	100
Learning rate	0.001
Training scaling (min./max.)	0.2 / 0.5
Validation scaling	0.33
Affine transformation	Enabled (0.025)
Skeletonization / dilations	Enabled / 1
Rotation / 90-degree rotation	Disabled
Elastic transformation	Disabled
TLD inference	
Number of text regions	One
Image scaling	Default
Minimal baseline length	Medium (25)
Baseline Accuracy threshold	Medium (55)
Use trained separators	No (-1)
Max-dist for merging baselines	Medium

Table 8: Recorded Transkribus settings for HTR training, TLD training, and TLD inference.