LREC 2026

The Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026

Workshop Proceedings

Editor
Kyle Gorman

12 May, 2026

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026

978-2-493814-92-0

Preface

Welcome to the third meeting of the CAWL workshop, featuring an invited talk by Prof. Zev Handel of the University of Washington.

This is the second meeting of CAWL under the aegis of ACL Special Interest Group on Writing Systems and Written Language (SIGWrit), which will continue to organize future CAWL meetings. The present officers of SIGWrit are president Kyle Gorman, vice president Constantine Lignos, and secretary-treasurer Zoey Liu, and student member Claytone Sikasote. For more information about the SIG, see `https://sigwrit.org/`.

Organizing Committee

Kyle Gorman, CUNY Graduate Center and Google Research
Constantine Lignos, Brandeis University
Zoey Liu, University of Florida
Claytone Sikasote, University of Cape Town

Program Committee

David Ifeoluwa Adelani, University College London
Sina Ahmadi, George Mason University
Enas Albasiri, Nvidia Corporation
Cecilia Overdotter Alm, Rochester Institute of Technology
Steven Bedrick, Oregon Health & Science University
Alexander Gutkin, Google Research
Nizar Habash, NYU Abu Dhabi
Yannis Haralambous, IMT Atlantique and CNRS Lab-STICC
Christo Kirov, Google Research
Yuval Pinter, Ben-Gurion University of the Negev
Djamé Seddah, Sorbonne University and Inria
Brian Roark, Google Research
Maria Ryskina, Massachusetts Institute of Technology

Table of Contents

Workshop Program

9:00–9:30 *Opening remarks*
Kyle Gorman

9:30–10:30 *Everything you wanted to know about East Asian writing but didn't think to ask: The history and structure of scripts for Chinese, Japanese, Korean, and Vietnamese*
Zev Handel

11:00–11:15 *The Degree of Language Diacriticity and Its Effect on Tasks*
Adi Cohen and Yuval Pinter

11:15–11:30 *Private-Use Area Characters in the Wild: Signal or Noise?*
Alexander Gutkin, Adrian Benton, Christo Kirov, Brian Roark and Lawrence Wolf-Sonkin

11:30–11:45 *HAnnoI: A Handwriting Annotation Interface to Extract Data for Linguistic Analyses of Graphetic Detail*
Joshua Wieler, Simon Petitjean, Kristian Berg, Henriette Huber and Stefan Hartmann

11:45–12:00 *SoriGraph: A New Database of Visual Feature-Level Descriptions of Written Korean*
Wednesday Bushong, Hala Habahbeh, Ryan Jiang and Yoolim Kim

12:00–13:00 *Confusable Characters as Endangered Language Markers: The Case of North Caucasus Writing Systems*
Alexander Gutkin, Adrian Benton, Christo Kirov and Brian Roark

12:00–13:00 *Prompting Approaches to Abbreviation Expansion*
Kyle Gorman

12:00–13:00 *Evaluating Data Augmentation Strategies for Training Spanish Misspelling Detection Models*
Manuel Castillo-Sancho, Jordi Porta and Asunción Gómez-Pérez

12:00–13:00 *Large Language Model-Based Post-OCR Correction for Low-Resource Kazakh Scripts*
Henry Gagnier

12:00–13:00 *G&P2P: A Multi-Source Approach to Grapheme-to-Phoneme Conversion*
Chun-Yi Jerry Peng

12:00–13:00 *A Lightweight N-gram Approach to Abbreviation Expansion in Large Corpora*
Tjaša Šoltes and Marko Bajec

12:00–13:00 *Inverse Text Normalization for Arabic Numbers in Streaming ASR*
Enas Albasiri, Myungjong Kim, Nourchene Ferchichi and Oluwatobi Olabiyi

The Degree of Language Diacriticity and Its Effect on Tasks

Adi Cohen, Yuval Pinter
Institute for Applied AI Research
Ben-Gurion University of the Negev
Beer Sheva, Israel
{adibc@post, uvp@cs}.bgu.ac.il

Abstract

Diacritics are orthographic marks that clarify pronunciation, distinguish similar words, or alter meaning. They play a central role in many writing systems, yet their impact on language technology has not been systematically quantified across scripts. While prior work has examined diacritics in individual languages, there's no cross-linguistic, data-driven framework for measuring the degree to which writing systems rely on them and how this affects downstream tasks. We propose a data-driven framework for quantifying diacritic complexity using corpus-level, information-theoretic metrics that capture the frequency, ambiguity, and structural diversity of character-diacritic combinations. We compute these metrics over 24 corpora in 15 languages, spanning both single- and multi-diacritic scripts. We then examine how diacritic complexity correlates with performance on the task of diacritics restoration, evaluating BERT- and RNN-based models. We find that across languages, higher diacritic complexity is strongly associated with lower restoration accuracy. In single-diacritic scripts, where character-diacritic combinations are more predictable, frequency-based and structural measures largely align. In multi-diacritic scripts, however, structural complexity exhibits the strongest association with performance, surpassing frequency-based measures. These findings show that measurable properties of diacritic usage influence the performance of diacritic restoration models, demonstrating that orthographic complexity is not only descriptive but functionally relevant for modeling.

Keywords: diacritics, writing systems, character-level models, character tagging

1. Introduction

Diacritics have been known to affect performance of language technology systems, but a thorough data-driven survey has yet to be done. Gorman and Pinter (2025) started to explore this issue, presenting anecdotal evidence and providing rules of thumb for NLP practitioners.

In this study, we consider the presence of diacritics a property of the writing system of a language and build a bottom-up framework for quantifying the degree to which diacritics play a role in corpora of a language, leading to a mechanism for examining the *degree* to which diacritics affect task performance, for a task expected to present such variation, namely diacritization itself.

Our approach considers "diacritifulness" of a langauge along a spectrum: for example, in Spanish vowels occasionally feature one kind of an accent mark (e.g., á), and one base consonant character (n) is shared by two phonemes (n and ñ). In Vietnamese, both vowel quality and tone are marked with co-inciding marks. In Hebrew, vowels are marked on phonetically preceding consonant characters and exist in large combinatorial distribution. Once this variation is accounted for quantitatively, the true effects of a language writing system on technology performance can be examined via tools like correlation.

Concretely, we propose the first-ever attempt to quantify the degree to which a language uses diacritics in its canonical writing system. Our approach is data-driven and information-theoretic: we calculate corpus-level metrics based on the distributions of diacritic marks, characters, and their combinations in each language. We assess the effect this property of "diacritifulness" has on the closest task to the subject matter: *diacritization* of undiacritized text, measured by word-level and character-level accuracy. Our results, identifying the correlation between a language's diacritifulness and success of various neural diacritization models, suggest that consideration of whether each of a script's characters admits one or more diacritic at a time, which we term single- vs. multi-diacritic systems, affects this relationship greatly, particularly for transformer-based diacritization models. We hope that our framework encourages more work on gauging written properties of languages and their effects on tasks, perhaps more downstream ones such as question answering and machine translation.[1]

2. Background

Diacritics are marks attached to base letters that encode additional phonological and/or grammatical information. They may indicate vowel quality or length, tone, gemination, or morphological distinctions. In some languages, like Vietnamese, diacritics are essential and form part of standard spelling.

[1] Our code, models and data are available at https://github.com/MeLeLBGU/Diacriticity.

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 1–8
12 May, 2026.

In other languages, such as Arabic and Hebrew, diacritics encode important phonological or grammatical distinctions but are often omitted in everyday writing. Writing systems that omit some or all diacritic marks are often referred to as defective writing systems, since the written text does not fully specify pronunciation or lexical identity.

The omission of diacritics introduces ambiguity. A single undiacritized form may correspond to multiple valid interpretations, requiring the inference of the intended meaning from context. This has been cited as one major reason that speech and language technologies for Arabic and Hebrew seemingly lag behind other, similarly-resourced languages (e.g., Tsarfaty et al., 2019).

In text-based systems, including machine translation, different diacritizations may have several valid lexical or grammatical meanings. Despite this, many NLP pipelines remove diacritics during preprocessing, especially in multilingual transformer-based models such as mBERT (Clark et al., 2022).

Diacritization, the task of restoring missing diacritics, has been extensively studied, particularly for Arabic and Hebrew. Early approaches relied on rules, lexicons or morphological analyzers (e.g., Darwish et al., 2017; Krstev et al., 2018). More recent work formulates diacritization as a sequence tagging or sequence-to-sequence problem using neural architectures such as RNNs and transformers (e.g., Belinkov and Glass, 2015; Shmidman et al., 2020; Gershuni and Pinter, 2022; Náplava et al., 2018; Náplava et al., 2021).

While these models perform well within individual languages, most work remains focused on language-specific or single writing system, such as Latin-based scripts. As a result, cross-script comparisons remain limited. There has also been relatively little systematic investigation of how orthographic properties themselves, such as diacritic frequency, structural diversity or ambiguity, affect restoration difficulty across languages.

Languages differ in how diacritics are used. In some scripts, each character can take only a single diacritic, while in others, base characters can carry multiple simultaneous marks. We refer to characters carrying exactly one diacritic as **single-diacritic**, and to those carrying two or more simultaneous marks as **multi-diacritic**. At the language level, we classify a writing system as multi-diacritic if it permits any character to bear multiple simultaneous diacritics; otherwise, we classify it as single-diacritic.

Beyond frequency and function, languages also vary in the predictability and diversity of their diacritic patterns. These differences suggest that diacritic usage varies along a spectrum of structural complexity. We propose quantitative corpus-level metrics to capture this variation and examine how it relates to neural diacritization performance across languages.

3. Metrics

Let C be the set of base characters of a language (e.g., the Latin letters a–z), and let D be the set of diacritic marks that may attach to them. We define a *rune* as a base character together with zero or more diacritics, so that both unmarked and diacritized characters are treated uniformly. For example, the Spanish character á can be viewed as the base letter a with an acute accent, while the Hebrew form בְּ consists of the base letter ב with two diacritic marks ◌ְ and ◌ּ. In both cases, we treat the base character together with its diacritics as a single rune. A diacritized string is then a sequence of runes $r_1, \ldots, r_n$, where each rune r_i consists of a base character $c_i \in C$ and a (possibly empty) subset of diacritics from D.

We follow previous work in defining a stripping function σ that removes diacritics and returns the corresponding base character sequence $c_1, \ldots, c_n$.

To capture the deeper properties of writing scripts and diacritics, we introduce a set of information-theoretic metrics that quantify the relationship between base characters and their diacritized realizations.

We present examples for our metrics on Hebrew and Spanish in Table 1.

Rune Surprisal. Rune surprisal (RS) quantifies the level of uncertainty associated with a diacritized version based on its base character. It is high when a character appears with multiple competing diacritic forms of similar probability, and low when a single form clearly dominates.

Formally, let r be a rune with base character $\sigma([r]) = c$, we define:

$$P(r \mid c) = \frac{\#(r)}{\sum_{r' : \text{base}(r')=c} \#(r')}.$$

The suprisal of a rune is:

$$RS(r) = -\log P(r \mid c).$$

Languages that show higher average rune surprisal when calculated over a corpus display greater ambiguity per character, which we expect to make diacritization more difficult.

Diacritic Token Surprisal. Diacritic token surprisal (DTS) breaks each rune into its individual diacritics marks and measures how unexpected those marks are given the base character. Rather than treating the full diacritized form as a single unit, this

"Corpus"	Chars	Diacs	Uniq. Diacs	RS	DTS	DSS
El niño bebió café en la mañana	25	4	2	0.28	0.15	0.11
הַיֶּלֶד שָׁתָה קָפֶה בַּבֹּקֶר	14	14	6	0.33	0.54	0.52

Table 1: Example sentences in Spanish and Hebrew illustrating how diacritic-based metrics are computed. The table reports the number of characters, total diacritics, number of unique diacritics, and the resulting RS, DTS, and DSS values.

metric evaluates the contribution of each individual mark.

Let d be a diacritic mark appearing as part of a rune r with base character c, We define:

$$P(d \mid c) = \frac{\#(d, c)}{\sum_{d' \in D} \#(d', c)},$$

where $\#(d, c)$ is the frequency of mark d with character c.

The diacritic token surprisal of rune r is:

$$DTS(r) = -\sum_{d \in r} \log P(d \mid c).$$

This metric captures token-level diacritic unpredictability. It is especially informative in systems where multiple diacritics can occur on the same character at the same time.

Diacritic Structural Surprisal. Diacritic structural surprisal (DSS) measures structural competition independently of frequency. Instead of asking how often a mark appears, it asks how many distinct diacritized forms it appears in.

For a base character c, let $T(c)$ denote the set of distinct runes formed from c, and let $T_d(c) \subseteq T(c)$ denote the subset of those runes that contain diacritic d. We define:

$$P_\delta(d \mid c) = \frac{|T_d(c)|}{|T(c)|}.$$

The structural surprise of rune r is:

$$DSS(r) = -\sum_{d \in r} \log P_\delta(d \mid c).$$

This metric captures type-level structural complexity. In single-diacritic systems, structural and token-level effects tend to align. In multi-diacritic systems, they can diverge substantially.

Diacritic Density. As a baseline surface measure, we define *diacritic density* as the proportion of diacritic tokens relative to base character tokens in the corpus:

$$\text{Density} = \frac{\sum_d \#(d)}{\sum_c \#(c)}.$$

This metric captures the overall diacritic load of the writing system, without conditioning on character identity or structural competition. While density does not measure ambiguity directly, it serves as a baseline indicator of how heavily diacritics are used in the writing system.

4. Data

We perform experiments and statistical analysis on fifteen languages across various scripts and typological characteristics. We include Latin-derived writing systems along with scripts like Arabic, Hebrew, and Bengali. For various languages, we provide multiple corpora. Our dataset includes 24 corpora, sourced from a mix of Universal Dependencies (UD; de Marneffe et al., 2021) and extensive web corpora. All corpora are fully diacritized. To the best of our knowledge, all diacritics have been manually added to the texts by either original authors or professionals, with the exception of some portions of the Hebrew data which were semi-automatically diacritized using Dicta's Nakdan API (Shmidman et al., 2020) followed by manual correction (see Gershuni and Pinter (2022)).

To ensure comparability across languages and corpora, we implemented a fixed-size sampling approach. For every corpus, we sample roughly 300,000 characters of diacritized text. Sampling occurred at sentence level: corpus sentences were shuffled and gradually accumulated until the target character count was achieved. For corpora that are smaller than the target (e.g., Latin), all available data was used and augmented by resampling sentences as needed.

Across various scripts, the mapping between characters and diacritics differs. In certain writing systems, a rune aligns with a single Unicode code point. For example, some Latin exist as precomposed characters: á (U+00E1, 'Latin Small Letter A with Acute'). Even letters with two diacritics may be encoded as a single code point, such as ắ (U+1EAF, 'Latin Small Letter A with Breve and Acute'). In other scripts, a rune is represented as a base character followed by one or more combining marks. For example, the Hebrew rune בָּ consists of the base character U+05D1 (HEBREW LETTER BET) together with U+05BC (HEBREW

Language Source	Diacritic Density %	Multi Diacritics %	% Words	% Lines	Mean Diacs/Word	# Runes	System
German	1.357	0.000	8.011	67.021	1.017	3	Single
UD 2.15 (GSD), 2.10 (HDR)							
Spanish	2.336	0.000	11.349	86.454	1.008	7	Single
UD2.10 (AnCora), 2.9 (GSD)							
Croatian	2.607	0.000	13.812	73.601	1.052	4	Single
Náplava et al. (2018)							
Galician	2.984	0.000	15.894	71.931	1.003	6	Single
UD 2.12 (CTG)							
Portuguese	3.700	0.000	16.229	64.003	1.140	12	Single
UD 2.13 (GSD, Cintil)							
French	3.845	0.000	17.539	90.684	1.119	13	Single
UD 2.7 (GSD), 2.2 (FTB)							
Romanian	5.999	0.000	29.488	84.026	1.141	5	Single
UD 2.8 (RRT, SiMoNERo)							
Turkish	6.510	0.000	30.971	74.863	1.347	9	Single
UD 2.8 (Penn, Kenet)							
Lithuanian	7.155	0.000	39.789	94.730	1.010	9	Single
UD 2.8 (Alksnis)							
Latin	11.118	0.000	53.338	96.053	1.207	6	Single
Chapters 1–6 of Vergil's Aeneid from Pharr's reader, digitized by Kyle Gorman							
Czech	14.505	0.000	55.119	93.514	1.510	15	Single
UD 2.15 (CAC, PDTC)							
Vietnamese	25.216	10.200	83.090	99.571	1.480	64	Multi
Náplava et al. (2018)							
Bengali	58.907	4.826	92.467	100.000	2.439	390	Multi
Leipzig (Goldhahn et al., 2012): Bengali News 2020 100k, Wiki 2021 100k							
Hebrew	66.243	14.096	99.823	99.761	3.607	325	Multi
Nakdimon (Gershuni and Pinter, 2022): Lit, news							
Arabic	82.813	9.297	99.552	89.221	3.686	353	Multi
Tashkeela (Zerrouki and Balla, 2017)							

Table 2: Corpus-level diacritic usage across languages. For languages represented by multiple corpora, statistics are averaged across corpora. The columns present overall diacritic density (ordering key), proportion of multi-diacritic characters, percentage of words and lines that are diacritized, mean diacritics per **diacritized** word, corpus-wide number of distinct character–diacritic combinations (runes), and classification as single- or multi-diacritic writing systems.

POINT DAGESH OR MAPIQ) and U+05B8 (HEBREW POINT QAMATS). Although this sequence is encoded as three unicode code points (בּ + ◌ + ◌ָ), it functions as a single letter with diacritics.

For scripts where diacritics are encoded as combining marks (Arabic, Hebrew, Bengali), we normalize all text to a consistent decomposed representation and compute statistics with respect to the underlying base character. Importantly, we treat each base character together with its associated diacritics as a single orthographic unit (rune), rather than as separate characters, even though they are encoded as multiple code points in Unicode.

4.1. Diacritic Usage Statistics

The statistics in Table 2 characterize the frequency, distribution, and combinatorial diversity of diacritics across writing systems. However, they do not directly capture predictability or ambiguity in character-diacritic mapping, which we aim to quantify using the information theoretic metrics introduced in Section 3.

5. Experimental Setup

We evaluate two established diacritization architectures in order to examine how properties of writing systems relate to model behavior across scripts. Our goal is not to propose a new modeling approach, but to analyze performance variation as a function of script-level characteristics.

BERT-based Diacritizer. We use the BERT-based (Devlin et al., 2019) sequence labeling model introduced by Náplava et al. (2021), which has been shown to achieve strong performance on Latin-script languages. Separate models were trained for each corpus in our dataset. For Latin-based scripts, we apply the original imple-

mentation without modification. For non-Latin scripts (Arabic, Hebrew, Bengali), we adapted the pipeline to work with writing systems in which the diacritics are encoded as separate combining marks rather than as precomposed characters. The transformer architecture, tokenization strategy (character-level), and training regime remain as in the original work.

For languages represented by multiple corpora, we additionally conducted cross-corpus evaluation. The model was trained on one corpus and evaluated on a different corpus from the same language in order to check for domain robustness.

RNN-based Diacritizer. For Latin-script languages only, we evaluate the character-level RNN model of Náplava et al. (2018), designed specifically for Latin alphabets where diacritics are precomposed in Unicode. Its encoding assumptions do not extend to Arabic, Hebrew and Bengali, so we restrict it to Latin-script corpora.

Both models use the original training objectives and decoding strategies from the reference implementations.

5.1. Evaluation Metrics

We evaluate diacritization performance using two complementary accuracy measures:

- **Word-level accuracy**, defined as exact match of full word's diacritization.
- **Rune-level accuracy**, defined as correct restoration of individual character–diacritic combinations.

Word-level accuracy reflects end-user correctness in downstream applications, while rune-level accuracy provides a more fine-grained view of model behavior and error patterns within words.

6. Results

6.1. Correlation Between Diacritic Properties

We first examine the inter-correlation between the proposed theoretical metrics. For each corpus, we compute the mean token-level complexity score for each metric and present it in Table 3.

Across all corpora, the metrics are extremely highly correlated ($|r| > 0.97$ for all pairwise comparisons), suggesting that at a broad cross-linguistic level they capture a shared underlying dimension of orthographic complexity. However, when separating the languages by script type, clearer structural differences emerge.

In multi-diacritic scripts, Rune Surprisal (RS) and Diacritic Token Surprisal (DTS) remain strongly correlated ($r = 0.93$), since both are based directly on observed corpus frequencies. Diacritic Structural Surprisal (DSS) is almost perfectly correlated with diacritic density ($r = 0.987$). This means that in these systems, languages with more diacritics overall also tend to allow more distinct diacritic combinations. Frequency-based measures correlate only moderately with structural measures ($r \sim 0.75$), suggesting that unpredictability of individual diacritics and the diversity of combination patterns are related but not identical properties.

In single-diacritic scripts, all metrics remain uniformly high ($r > 0.83$), with RS and DTS nearly identical ($r = 0.98$). This reflects the fact that when only one diacritic may attach to a character, frequency-based and structural measures collapse into a single dominant dimension.

6.2. Model Performance as a Function of Diacritic Properties

We examine the relationship between corpus-level diacritic metrics and diacritization accuracy across all languages and corpora using the BERT-based model. Table 4 reports Pearson correlations between each complexity metric and both word-level and rune-level accuracy.

Across all languages, word-level and rune-level accuracy for the BERT-based diacritizer shows a strong and highly significant negative correlation with diacritic complexity, which indicates that languages with higher orthographic complexity tend to score lower on restoration accuracy. This pattern holds across all metrics. Restricting the analysis to Indo-European languages show the same overall trend. This suggests that the effect is not limited to cross-family differences but also remains within a single language family.

Single-Diacritic Scripts. We next restrict the analysis to single-diacritic scripts, comprising 11 languages and 17 corpora. As observed above, in these scripts, the statistical metrics are mostly aligned.

For the BERT model, at both word and rune-level, DTS and DSS metrics show moderate correlations with accuracy. At rune-level, the Rune Suprisal metric is also moderately correlated. However, the overall effect is smaller and less significant than all languages.

In contrast, the RNN model exhibits strong and consistent negative correlations between diacritics complexity and accuracy across all metrics. At the word-level, correlations remain moderate, with RS showing the strongest association. This pattern suggests that the RNN architecture is more sensitive than BERT to increases in complexity even

Language	Family	Density	RS	DTS	DSS	RNN-Word	RNN-Rune	BERT-Rune	BERT-Word
German	Indo-Euro	1.374	0.044	0.031	0.009	94.245	99.088	99.045	94.888
Spanish	Indo-Euro	2.330	0.092	0.069	0.018	91.513	98.426	98.299	94.038
Croatian	Indo-Euro	2.583	0.058	0.039	0.023	93.753	98.911	99.336	96.613
Galician	Indo-Euro	2.985	0.111	0.082	0.021	92.488	98.717	99.142	95.963
Portuguese	Indo-Euro	3.694	0.150	0.114	0.047	94.805	99.033	99.170	96.526
French	Indo-Euro	3.763	0.131	0.096	0.058	91.539	98.405	97.627	91.421
Romanian	Indo-Euro	6.043	0.172	0.118	0.056	90.492	98.291	98.957	94.913
Turkish	Turkic	6.536	0.122	0.071	0.057	93.939	98.831	99.284	96.338
Lithuanian	Indo-Euro	7.127	0.193	0.133	0.068	99.659	97.335	97.048	85.726
Latin	Indo-Euro	11.181	0.237	0.145	0.072	90.140	98.320	98.957	94.729
Czech	Indo-Euro	14.434	0.327	0.207	0.121	77.613	95.567	97.052	87.353
Vietnamese	Austroasiatic	25.234	0.783	0.586	0.513	67.787	92.369	91.475	74.152
Bengali	Indo-Euro	58.635	1.593	1.207	1.168	–	–	80.668	60.770
Hebrew	Afro-Asiatic	66.026	1.706	1.366	1.487	–	–	77.996	54.479
Arabic	Afro-Asiatic	82.017	1.430	1.350	1.784	–	–	67.064	35.637

Table 3: Language-level averages of diacritic complexity metrics and diacritization performance, ordered by increasing diacritic density. For languages represented by multiple corpora, values are averaged across corpora. RS = Rune Surprisal; DTS = Diacritic Token Surprisal; DSS = Diacritic Structural Surprisal. Model performance is reported as word-level and rune-level accuracy for RNN and BERT-based diacritizers.

when structural variation is limited to a single diacritic per character.

Multi-Diacritic Scripts. We next examine multi-diacritic scripts, comprising 4 languages and 6 corpora, numbers which affect the power of our analysis. In these systems, base characters may carry multiple simultaneous diacritics, and frequency-based and structural measures diverge.

For the BERT model, a clear pattern emerges at the word and rune level despite the small sample size. Structural measures show substantially stronger negative associations with accuracy than frequency-based measures. In contrast to the global and single-diacritic analyses, where the metrics largely behave similarly, structural complexity becomes the dominant predictor of performance. Although the analysis includes only six corpora, the direction of the effects is consistent with the findings so far.

This pattern suggests that, in multi-diacritic scripts, BERT performance is more strongly affected by structural complexity than by token-level unpredictability alone. In other words, the model is more sensitive to how many structurally distinct combinations there are than to how skewed the distribution of diacritics is.

Overall, the results indicate that multi-diacritic scripts introduce a qualitatively different form of orthographic complexity, which reveals that structural combinatorics plays a distinct and predictive role in model performance.

6.3. Cross-Corpus Evaluation

We examine cross-domain robustness within individual languages. All non-Hebrew corpora were drawn from Universal Dependencies (UD; de Marneffe et al., 2021), while the Hebrew corpora were taken from the fully diacritized dataset introduced by Gershuni and Pinter (2022).

Cross-corpus evaluation was conducted for the following language pairs: Hebrew (literary / news), French (FTB, news / GSD, web), Spanish (AnCora, news / GSD, web), Turkish (Kenet, grammar examples / Penn, news), Romanian (RRT, news and web / SiMoNERo, medical), and Portuguese (CINTIL, mixed genres / GSD, web). Across languages, cross-corpus evaluation shows a modest decrease in performance relative to in-domain results. On average, rune-level accuracy drops by approximately 1.5%, and word-level accuracy by around 3%. The drop is reasonable, as different domains can contain unique vocabularies and may vary in the distribution of character-diacritic combinations. These domain differences can affect restoration accuracy, particularly given the relatively limited size of each sampled corpus (approximately 300K characters) At the same time, the relatively small decrease suggests that diacritic usage is largely consistent within a language. Although specific words change across corpora, the patterns remain stable, allowing the model to generalize well across domains.

7. Conclusion

We presented the first comparative study of the orthographic phenomenon of diacritization, ana-

Regime	Metric	BERT-Rune	BERT-Word	RNN-Rune	RNN-Word
Single-diacritic scripts					
	RS	−0.47	**−0.55***	**−0.80*****	**−0.62****
	DSS	**−0.50***	**−0.57***	**−0.76*****	**−0.57***
Multi-diacritic scripts					
	RS	−0.59	−0.50	–	–
	DSS	**−0.95****	**−0.92****	–	–
Indo-European					
	RS	**−0.98*****	**−0.97*****	–	–
	DSS	**−0.99*****	**−0.96*****	–	–
All Languages					
	RS	**−0.94*****	**−0.94*****	–	–
	DSS	**−0.99*****	**−0.98*****	–	–

Table 4: Pearson correlations between diacritization accuracy and diacritic complexity metrics: Rune Surprisal (RS) and Diacritic Structural Surprisal (DSS). Significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$.

lyzed through statistical and information-theoretic constructs pertaining to the intrinsic properties of diacritized text across two dozen corpora. We show that, as expected, the statistical presence and surprisal of diacritic marks across texts in a language affects the ability of state-of-the-art off-the-shelf diacritization models to restore marks in that language. In future work, we will act on our actionable conclusions and provide practical recommendations for diacritics restoration and for handling undiacritized text in downstream NLP systems, building also on the principles demarcated by Gorman and Pinter (2025).

An additional avenue for future work would be to enrich the presented study by incorporating additional features for correlation analysis, such as the functional role of diacritics in a language. For example, diacritics may encode phonological, morphological, or tonal information; these differences can help explain variation in model performance beyond structural complexity alone.

Acknowledgements

We would like to thank Kyle Gorman and the anonymous reviewers for valuable comments on earlier drafts. This research was supported by grant no. 2022215 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

8. Bibliographical References

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2281–2285, Lisbon, Portugal. Association for Computational Linguistics.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

Kareem Darwish, Hamdy Mubarak, and Ahmed Abdelali. 2017. Arabic diacritization: Stats, rules, and hacks. In *Proceedings of the Third Arabic Natural Language Processing Workshop*, pages 9–17, Valencia, Spain. Association for Computational Linguistics.

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal Dependencies. *Computational Linguistics*, 47(2):255–308.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Elazar Gershuni and Yuval Pinter. 2022. Restoring Hebrew diacritics without a dictionary. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1010–1018, Seattle, United States. Association for Computational Linguistics.

Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. 2012. Building large monolingual dictionaries at the Leipzig corpora collection:

From 100 to 200 languages. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 759–765, Istanbul, Turkey. European Language Resources Association (ELRA).

Kyle Gorman and Yuval Pinter. 2025. Don't touch my diacritics. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 285–291, Albuquerque, New Mexico. Association for Computational Linguistics.

Cvetana Krstev, Ranka Stanković, and Duško Vitas. 2018. Knowledge and rule-based diacritic restoration in Serbian. In *Proceedings of the Third International Conference on Computational Linguistics in Bulgaria (CLIB 2018)*, pages 41–51, Sofia, Bulgaria. Department of Computational Linguistics, Institute for Bulgarian Language, Bulgarian Academy of Sciences.

Jakub Náplava, Milan Straka, and Jana Straková. 2021. Diacritics Restoration using BERT with Analysis on Czech language. *The Prague Bulletin of Mathematical Linguistics*, 116:27–42.

Jakub Náplava, Milan Straka, Pavel Straňák, and Jan Hajič. 2018. Diacritics restoration using neural networks. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Avi Shmidman, Shaltiel Shmidman, Moshe Koppel, and Yoav Goldberg. 2020. Nakdan: Professional Hebrew diacritizer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 197–203, Online. Association for Computational Linguistics.

Reut Tsarfaty, Shoval Sadde, Stav Klein, and Amit Seker. 2019. What's wrong with Hebrew NLP? and how to make it right. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 259–264, Hong Kong, China. Association for Computational Linguistics.

Taha Zerrouki and Amar Balla. 2017. Tashkeela: Novel corpus of arabic vocalized texts, data for auto-diacritization systems. *Data in Brief*, 11:147–151.

Private-Use Area Characters in the Wild: Signal or Noise?

Alexander Gutkin[◦] **Adrian Benton**[†] **Christo Kirov**[†]
Brian Roark[‡] **Lawrence Wolf-Sonkin**[†]
Google Research, [◦]London, UK; [†]New York; [‡]Portland, OR
{agutkin,adbenton,ckirov,roark,wolfsonkin}@google.com

Abstract

The Private-Use Area (PUA) designation plays an important role in the Unicode standard. It covers several ranges of Unicode code points with no official character assignments. A PUA range is primarily used as a temporary representation mechanism for characters falling outside the official standard, to facilitate text entry and display of orthographies that are not otherwise adequately represented. The primary downside of PUA use is that characters lose their semantics if the pairing with the corresponding display font is broken, in which case they cannot be faithfully displayed in a general setting. Large-scale multilingual web corpora invariably contain PUA code points of unclear provenance, which may commonly be treated as noise and discarded. We investigate the distribution of PUA characters within large-scale web corpora, and analyze the resulting distributions across both scripts and writing systems. We show that, while the proportion of PUA-bearing paragraphs in the original corpora are small, PUA-bearing tokens can signal texts from under-represented languages. We additionally explore whether an off-the-shelf large language model (LLM) can classify PUA characters as constituting relevant orthographic signals versus punctuation or other noise. Our methods identify millions of paragraphs making use of such characters, and we argue that such data is important for the long tail of data-scarce orthographies. Moreover, as a primary Unicode mechanism for poorly represented writing systems, PUA characters are here to stay.

Keywords: text noise, Unicode, Private-Use Area characters, web-crawled corpora, fonts

1. Introduction

The information loss in digital communication due to legacy character encoding standards has been extensively studied by specialists in digital language preservation (Bird and Simons, 2003; Brown and Woods, 2009; Dombrowski et al., 2024), and natural language processing (Stefanovitch, 2022; Yi and Bowern, 2023; Karagöz et al., 2024). Despite wider adoption of the Unicode character encoding standard along with the OpenType format for representing scalable and programmable computer fonts (Haralambous, 2007; Hossain, 2024), the continued use of many incompatible fonts with legacy and custom character encoding schemes—requiring specialized algorithms for font identification and conversion—has been noted for South Asian scripts and beyond (Lehal et al., 2012; Mahi and Verma, 2015; Pine and Turin, 2018; Bradley and Blokland, 2023). Some communities even choose custom encoding schemes and specialized input methods based on these schemes over Unicode because of the representational inadequacies of the latter, as is the case with the traditional Mongolian script (Batjargal et al., 2011; Wang et al., 2016).[1]

The continued evolution of the Unicode standard partially addresses the "digital divide" between established languages and their writing systems on the one hand and endangered ones on the other (Kornai, 2013; Joshi et al., 2019; Zaugg et al., 2022; Simons et al., 2022). Information loss, however, is still possible in a pure Unicode ecosystem. One Unicode feature potentially responsible for such information loss is the set of private-use characters. These are valid code points from a private use area (PUA), which have no interpretable semantics within the Unicode standard. Individuals or organizations, such as font foundries, can assign meaning to such characters by private agreement (Unicode Consortium, 2024, Section 23.5). PUA characters provide a popular mechanism for representing glyphs from low-resource, historic and endangered language orthographies with no official assignments in the Unicode standard at implementation time (Priest, 2007). While flexible, PUA characters lose their meaning in a digital document without an association to (and availability of) the corresponding font, resulting in information loss quite similar to that observed with legacy character encoding schemes mentioned earlier (Anderson, 2018). Furthermore, the same slot in the PUA table may be inadvertently shared by multiple font designers, preventing general interchange (Kempgen, 2008).

With the advent of large language models (LLMs), large-scale corpora based on web-crawled data, such as MADLAD-400 (Kudugunta

[1] Schillo and Turin (2022) discuss typographic issues in representing North American Indigenous language orthographies stemming from the imprecise specification of Unicode character semantics.

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 9–32
12 May, 2026.

et al., 2023), GlotCC (Kargaran et al., 2024), FineWeb2 (Penedo et al., 2025) and DCAD-2000 (Shen et al., 2025), have become important sources of model pre-training data, especially for low-resource languages (Caswell et al., 2020; Bapna et al., 2022). When analyzing such web data one inevitably encounters PUA characters. Since PUA characters often fall outside the standard orthography of natural languages, they are usually treated as noise, and noise can degrade model quality (Cooper Stickland et al., 2023; Wang et al., 2024; Sperduti and Moreo, 2025; Meng et al., 2025).

This paper offers a preliminary study of characters from Unicode PUA ranges found in popular web-crawled corpora, a phenomenon which is under-explored in the literature. We mine text containing PUA characters from two popular large-scale web-crawled datasets of over 7 billion web documents and analyze PUA character distributions over script and writing system at the token and paragraph levels. We find that at most 0.01% of paragraphs contain PUA characters, and thus these are unlikely to harm the performance of downstream models on well-resourced languages. However, we hypothesize that this may not be the case for languages for which only small amounts of data are generally available. In such scenarios, text with PUA characters should receive more attention, especially when prioritizing effective modeling for under-resourced languages. We substantiate this argument with examples from under-represented Cyrillic script orthographies. Inferring the identity of a given PUA code point in the absence of the original character semantics is difficult, but the character context can be informative. We conclude by offering a preliminary step towards this inference task, by assessing the ability of an off-the-shelf LLM to follow instructions when asked to discriminate between orthographic ("interesting") and non-orthographic instances of PUA characters.

The overarching goal of this descriptive analysis is to provide actionable insights for NLP practitioners. Specifically, by mapping PUA distributions, we aim to: (1) prevent the inadvertent destruction of unstandardized scripts in text cleaning pipelines; (2) provide a method for discovering "hidden" low-resource corpora masked by legacy font encodings; and (3) highlight active but poorly supported digital language communities. Ultimately, this analysis is a preliminary step towards bridging the gap between raw web scraping and the targeted curation of inclusive NLP datasets.

2. Background

Unicode has three PUA ranges with 137,468 code points reserved for PUA characters. PUA assignments are typically used as a temporary mechanism by academics, governments and vendors for representing new, as yet un-encoded characters before their official assignment to valid non-PUA Unicode code points. For example, among government-endorsed allocations are the Chinese national standard "Tibetan coded character set – Extensions A and B" for representing precomposed Tibetan script ligatures exclusively using PUA code points.[2] Since approvals to get new characters accepted into the next version of the Unicode standard take considerable time, and some proposed character sets may not be accepted for various reasons, documenting the available PUA characters in use by different entities in order to avoid character collisions and improve coordination becomes important.

One example of such a PUA character allocation registry is the ConScript Unicode Registry,[3] which, among dozens of allocations, originally documented the PUA-based encoding of the Phaistos Disc signs before they were officially adopted as part of the Unicode standard—at which point a mapping from the original PUA code points to code points with relevant semantics was provided (Everson, 2008). In the context of historic writing systems and associated fonts there are additional prominent PUA character allocations for Early Slavonic (Cleminson et al., 2010), medieval orthographies (Emiliano, 2012; Haugen, 2013), and in particular Runic (Magin and Smith, 2023) as part of Medieval Unicode Font Initiative (MUFI),[4] among others.

Fonts covering a wide range of rare language orthographies without recourse to PUA are uncommon, mostly confined to academic publishers such as Brill (Rietbroek, 2021), who have painstakingly avoided PUA since the earliest versions of their fonts covering Cyrillic, Greek and Latin scripts. Rietbroek (2014, p. 1) states: "The Unicode Standard is rigorously adhered to: there is no dependence on the Private Use Area (PUA), as it happens frequently in other fonts with regard to characters carrying rare diacritics or combinations of diacritics. Instead, all alphabetic characters can carry any diacritic or combination of diacritics, even stacked, with automatic correct positioning."

An additional often neglected use case of PUA should be noted. Font designers sometimes prefer PUA code points as an economical shortcut to introducing new glyphs even in cases where the glyphs can be encoded using official Unicode assignments via a longer sequence of code points. For example, there is no single Unicode code point for representing the small *O with*

[2] GB/T 20542-2006 and GB/T 22238-2008.

[3] https://www.evertype.com/standards/csur

[4] https://mufi.info

macron in the Cyrillic orthography of the Mansi language (Bradley and Skribnik, 2021). Instead the canonical way of representing this letter is a combination of U+043E (*Cyrillic small letter O*) and U+0304 (*combining macron*). Some Cyrillic fonts, however, represent this glyph as a single PUA code point. One possible explanation of the font designer motivation for such a choice is that it allows them to treat this glyph similarly to its Latin counterpart (*Latin small letter O with macron*), for which a valid single Unicode code point (U+014D) is defined in the Unicode standard.

Before documenting PUA prevalence in large web text corpora, we first present our data mining methods for extracting text samples.

3. Data Mining Methods

Corpora MADLAD-400 is a general domain multilingual dataset based on CommonCrawl[5] that spans 419 languages (Kudugunta et al., 2023). The dataset has `noisy` and `clean` partitions, the latter obtained by filtering the former with a variety of noise-reducing heuristics. Since we are interested in text containing PUA characters, which may have been deemed noise by prior filters, we make use of the `noisy` partition, which includes 5 trillion tokens over 7.2 billion web documents.

DCAD-2000 (Shen et al., 2025) is a recently introduced large-scale multilingual corpus, significantly larger than MADLAD-400 both in size and coverage, supporting 2,282 language-script pairs. It includes a more recent CommonCrawl dump[6] and other multilingual sources such as MaLA (Lin et al., 2024) and FineWeb-2 (Penedo et al., 2025). Similar to MADLAD-400, we process the unfiltered union of the `keep` and `remove` DCAD-2000 partitions, spanning 8B web documents.[7]

We chose these two datasets as sufficiently different, in terms of both recency (DCAD-2000 is significantly newer given the fast pace of appearance of various web-crawled datasets) and scope, both factors that may influence the PUA character distributions found in the datasets.

Pipeline Our PUA data mining pipeline consists of four steps, similar to the data preparation pipeline in Benton et al. (2026). First, each web document is split at newline characters into what we will call *paragraphs*. Paragraphs with less than four whitespace-delimited tokens are merged with the previous paragraph if that one has less than 30 tokens. Unmerged paragraphs with less than four tokens are discarded. We next discard all paragraphs where more than 40% of whitespace-delimited tokens start with a hashtag character. We then discard paragraphs without any valid PUA-containing tokens, where tokens are considered valid PUA-containing if they (a) have at least one PUA character; and (b) contain non-PUA characters, all of which come from the same script (after stripping leading and trailing punctuation). In other words, tokens with multiple scripts or only PUA characters do not count when deciding whether to retain the paragraph. Finally, we apply the state-of-the-art off-the-shelf LID model GlotLID (Kargaran et al., 2023) that covers 2,282 language-script pairs to the resulting paragraphs.[8]

When determining valid PUA-containing tokens above, PUA characters are allowed in any position of the token (*Anywhere*). This may be too permissive as we are primarily interested in retrieving orthographic tokens corresponding to words in natural language, while PUA characters at the token periphery may potentially encode non-linguistic glyphs, such as list delimiters or other forms of general punctuation.[9] Motivated by this observation, we also pursue an alternate strategy, where valid PUA-containing tokens must contain the PUA characters word-internally (*Internal*). In the next section, we compare the general distributions obtained with both strategies.

Filter	MADLAD-400	DCAD-2000
Documents	7 158 832 435	8 033 425 484
Paragraphs	145 459 293 186	159 081 108 657
(1) Too short	−1 100 076 575	−7 191 309 323
(2) Hashtags	−135 492 904	−25 226 491
(3) No PUA	−144 155 975 436	−151 795 290 723
(3) Invalid PUA	−52 546 928	−60 163 490
Output	15 201 343	9 118 630

Table 1: Counts of filters removing paragraphs without valid PUA-containing tokens. Other than initial document count, all counts are paragraphs.

Filter performance Table 1 presents counts from the use of the above pipeline to mine paragraphs with PUA characters in MADLAD-400 and DCAD-2000, where PUA characters are allowed *Anywhere* in the token. The number of input documents and newline-delimited paragraphs are given in the first two rows. Rows three to six present the number of paragraphs discarded due to the various filtering conditions described above. The number of paragraphs passing all filters and retained by the pipeline are displayed in the final row.

[5] Snapshots before August 2022.

[6] CC-MAIN-2024-46: November 2024.

[7] This number excludes documents in Chinese varieties and Japanese which we removed to preserve disk space. We expect PUA use within these sub-corpora to be similar to MADLAD-400.

[8] The same model was used for document-level LID to build DCAD-2000. The presence of PUA characters may inject some noise into predictions by the LID model.

[9] See the end of Section 4 for examples of PUA characters representing linguistic and non-linguistic glyphs.

As can be seen from the table, the "Invalid PUA" filter for MADLAD-400 drops 78% of candidate paragraphs, while for DCAD-2000 87% of the paragraphs are ignored. This is not surprising because the presence of one "invalid" token, by our definition, is enough to discard the entire paragraph. While the number of useful paragraphs thus removed is likely high, it nevertheless suits our purpose to opt for higher precision and lower recall before investigating the resulting PUA character distributions in the next sections.

4. PUA Character Distribution

Here we take a general look at the top scripts associated with PUA characters followed by the individual top writing systems using PUA character tokens. For each dataset, we consider filtering by either the permissive *Anywhere* or restrictive token-*Internal* PUA character placement strategies.

Scripts The distribution of PUA character tokens among different scripts in MADLAD-400 and DCAD-2000 is shown in Table 2 for the two PUA character placement strategies (*Anywhere* in the left and *Internal* in the right table), resulting in four configurations overall. For each configuration, the top five scripts ranked in terms of number of tokens with PUA characters are shown. Counts from the remaining scripts are aggregated under "Other". Each script, represented by its ISO 15924 code (ISO, 2022), is shown with the count of PUA character tokens and their percentage of the overall number of PUA tokens.

Although DCAD-2000 is significantly larger than MADLAD-400, it yields fewer PUA character tokens. This observation holds for both *Anywhere* and *Internal* token selection strategies. One contributing factor is likely that DCAD-2000 includes fresher CommonCrawl snapshots, containing fewer web pages with broken characters (Shen et al., 2025). This noise reduction may be due in part to existing web pages being updated to newer versions, since older page fonts make use of more PUA assignments in their inventory. As can be seen from the table, all four resulting datasets are heavily skewed towards Latin script tokens with PUA characters, alone accounting for close to (or over) 60% of the respective distributions. In three out of four configurations, Cyrillic script tokens are ranked as the second most frequent. The third most frequent PUA character tokens are encountered in Thai script, with the Greek script as an outlier in one of the conditions. We also note that restricting the allowed placement of PUA characters from *Anywhere* to *Internal* results in 2.5 times fewer PUA tokens for MADLAD-400 and 7.2 times fewer PUA tokens for DCAD-2000.

The contents of the *Other* bins differ among the four configurations. The distributions of PUA character tokens among the scripts (excluding the top five) that belong to this category are shown in Table 3. Each configuration is shown along with the name of the dataset, the PUA character placement strategy, the number of scripts in the category N_S and the statistics for the number of PUA characters per-script that includes the mean μ and logarithm of variance $\log(\sigma^2)$.[10] The MADLAD-400 configurations in general have more distinct scripts with PUA character tokens than DCAD-2000 regardless of the character placement strategy. Unsurprisingly, the more restrictive PUA character placement strategy (*Internal*) results in a smaller set of scripts than the more flexible (*Anywhere*) strategy. Nevertheless, in all configurations there is a very diverse mix of both "well-resourced", under-represented and historic scripts that comprise the *Other* bin. For example, the MADLAD-400 61-script configuration obtained with permissive PUA character placement includes reasonably high-frequency (in terms of PUA character tokens n) scripts such as Armenian (`Armn`, n=13625), Hiragana (`Hira`, n=5232), and Coptic (`Copt`, n=4628), while the long tail of the distribution has Saurashtra (`Saur`, n=11), Cuneiform (`Xsux`, n=2), and Tagalog (`Tglg`, n=1). The collection has a single token in Hanifi Rohingya script (`Rohg`), which in its romanized form "Ruwainggya" means the name of the script itself. The token is 10 characters long with a single PUA character that likely corresponds to an older encoding of a nasal marker (Pandey, 2015). In other cases like the two Cuneiform (`Xsux`) examples, the use of PUA characters is non-orthographic. The full PUA token rank-frequency distribution for all scripts and conditions is provided in Appendix A.

Writing systems Next we consider the distribution of PUA paragraphs over writing systems. We count PUA paragraphs rather than tokens since the writing system is inferred at the paragraph rather than token level. Even modern LID models such as GlotLID typically require at least a sentence worth of text to accurately predict language.

The top-10 writing systems, in terms of the number of paragraphs with at least one PUA character token, are shown in Table 4. The two sub-tables on the left display language rankings for the permissive PUA character placement strategy (*Anywhere*) for MADLAD-400 and DCAD-2000, while the two sub-tables on the right show the rankings obtained allowing only token-internal placement (*Internal*). For each writing system, designated by the ISO 639-3 language code and the

[10] The distribution mostly consists of low counts and a few very high counts, hence the logarithm of variance.

PU character location: *Anywhere* in token

MADLAD-400			DCAD-2000		
Script	Tokens	%	Script	Tokens	%
Latn	32 240 234	80.4	Latn	12 835 255	81.0
Cyrl	2 803 337	7.0	Cyrl	1 071 969	6.8
Thai	2 672 535	6.7	Grek	644 169	4.1
Hani	825 219	2.1	Arab	396 244	2.5
Arab	507 019	1.3	Thai	296 941	1.9
Other	1 056 045	2.6	*Other*	616 509	3.9
Total	40 104 389			15 861 087	

PU character location: Token-*internal* only

MADLAD-400			DCAD-2000		
Script	Tokens	%	Script	Tokens	%
Latn	11 739 280	73.4	Latn	1 283 007	58.6
Thai	2 092 819	13.1	Cyrl	464 904	21.2
Cyrl	1 266 526	7.9	Thai	144 511	6.6
Hani	440 618	2.8	Arab	115 386	5.3
Arab	161 666	1.0	Hang	67 066	3.1
Other	287 384	1.8	*Other*	116 425	5.3
Total	15 988 293			2 191 299	

Table 2: Distribution of top-5 scripts among the tokens with PUA characters in MADLAD-400 paragraphs (left) and DCAD-2000 (right) under both *Anywhere* (left table) and *Internal* (right table) strategies.

Dataset details		Script-token distribution		
Name	PU Position	N_S	μ	$\log(\sigma^2)$
MADLAD-400	*Anywhere*	61	17 312.2	22.1
	Internal	51	6386.3	19.8
DCAD-2000	*Anywhere*	49	12 581.8	20.6
	Internal	33	3528.0	17.1

Table 3: Scripts and PUA tokens in the *Other* bin.

ISO 15924 script code, the total number of paragraphs N_P that contain PUA character token(s) are shown along with the corresponding percentage of the total number of such paragraphs observed in the data. In addition to the top-10 ranked writing systems, the information for the remaining writing systems is accumulated in the *Other* bin. The numbers of writing systems N_L in this bin for each resulting collection are shown in Table 5.

The PUA paragraph distributions in Table 4 are dominated by the Latin script and, in particular, English, Spanish and French. One outlier among the top-3 in the DCAD-2000 collection obtained with *Internal* character placement is Czech (ces) which, along with Polish, is also present in the top-10 of DCAD-2000 *Anywhere*. This collection also prominently features Modern Greek (ell), which is not present in any other top-10 ranking. Another interesting entry obtained for MADLAD-400 *Internal* collection is the relatively high-frequency group of paragraphs marked as "no linguistic content" (zxx) by GlotLID, and shown in red. These likely correspond to paragraphs of questionable provenance, such as *mojibake*, text graphics or intentionally obfuscated content. We also note that in addition to Latin, the only other script that features in all four collections is Cyrillic, with the surprising inclusion of low-resource Mansi (mns) orthography in the DCAD-2000 *Internal* collection. Finally, the Thai, Mandarin Chinese and Korean scripts also appear frequently in the top-10 writing systems in three out of four collections. There is a very long tail of well over 700 other language-script pairs present in the *Other* bin (Table 5) that corresponds to about one third or more of the total number of paragraphs with PUA characters obtained for all the collections.

Does size matter? The low number of paragraphs extracted by our pipeline indicates that PUA characters are relatively infrequent in MADLAD-400 and DCAD-2000. The largest collection of paragraphs with PUA characters extracted from MADLAD-400 with permissive PUA character placement (*Anywhere*) consists of 15M paragraphs (see Table 1), which is equivalent to 0.01% of all the paragraphs examined by our pipeline for this condition. For the smallest collection, DCAD-2000 in restrictive PUA character placement mode (*Internal*), this value reduces to 0.006%. Given how infrequently these PUA characters are used, they are likely to have negligible effect on any downstream models constructed using such text for a range of well-resourced languages. It is also likely that the situation will be different for under-represented languages, especially from the long tail of the distribution, for which very small amounts of data are present in such corpora. Importantly, however, the presence of such characters may be quite useful as markers of under-represented orthographies or orthographies "in flux".

Rank-proportion view The analysis in this section relies on the rank-frequency distribution induced by the absolute counts. An alternative is to normalize by the count of tokens and paragraphs in the original corpora, ranking by proportions instead of raw counts. The resulting ranking better highlights the heavy-tail structure and the relative prevalence of PUAs in lower-resourced scripts and writing systems (detailed in Appendix B).

Cyrillic at a glance To demonstrate the above point on the usefulness of this type of data for mining under-represented orthographies, we zoom in on the Cyrillic script portion of the PUA character paragraphs extracted from MADLAD-400 using the permissive character placement strategy. The ranking of the top-10 Cyrillic writing systems in terms of number of paragraphs before and after applying our PUA filters is shown in Table 6. Beyond the obvious presence of well-resourced orthographies, PUA filtering exposes new low-resource orthographies of Khanty, Mansi and Church Slavic,

PU character location: *Anywhere* in token							
MADLAD-400				DCAD-2000			
Lang.	Script	N_P	%	Lang.	Script	N_P	%
eng	Latn	4 424 545	29.1	eng	Latn	1 877 073	20.6
spa	Latn	1 444 067	9.5	fra	Latn	767 446	8.4
fra	Latn	979 665	6.4	spa	Latn	607 533	6.7
rus	Cyrl	918 580	6.0	ell	Grek	597 827	6.6
por	Latn	736 590	4.8	pol	Latn	550 285	6.0
deu	Latn	468 212	3.1	deu	Latn	537 344	5.9
ind	Latn	396 848	2.6	nld	Latn	424 145	4.7
ita	Latn	388 052	2.5	ces	Latn	377 946	4.1
nld	Latn	350 477	2.3	ita	Latn	370 327	4.0
tha	Thai	271 180	1.8	rus	Cyrl	363 222	3.9
Other		4 823 127	31.7	*Other*		2 645 482	29.0
Total		15 201 343				9 118 630	

PU character location: Token-*internal* only							
MADLAD-400				DCAD-2000			
Lang.	Script	N_P	%	Lang.	Script	N_P	%
eng	Latn	1 243 931	29.7	ces	Latn	247 631	24.9
spa	Latn	382 776	9.1	eng	Latn	104 765	10.5
fra	Latn	362 742	8.6	fra	Latn	69 341	6.9
tha	Thai	207 384	4.9	spa	Latn	59 096	5.9
deu	Latn	159 843	3.8	rus	Cyrl	35 634	3.6
zxx	Zzzz	158 484	3.7	mns	Cyrl	30 605	3.1
rus	Cyrl	152 232	3.6	kor	Hang	30 210	3.0
por	Latn	119 231	2.8	pol	Latn	27 436	2.8
cmn	Hani	118 495	2.8	ita	Latn	26 381	2.6
ita	Latn	117 797	2.8	tha	Thai	25 315	2.5
Other		1 171 210	28.0	*Other*		339 636	34.1
Total		4 194 125				996 050	

Table 4: Distribution of top-10 languages among the paragraphs with PUA characters in MADLAD-400 (left) and DCAD-2000 (right) under both *Anywhere* (left table) and *Internal* (right table) strategies.

Name	*Anywhere* (N_L)	*Internal* (N_L)
MADLAD-400	1480	1204
DCAD-2000	1021	716

Table 5: Unique writing systems with PUA characters in the *Other* bin.

Original Distribution		PUA Distribution		Change
Language	%	Language	%	
Russian	88.6	Russian	75.9	↓
Ukrainian	5.2	Ukrainian	9.6	↑
Bulgarian	3.1	Mansi	2.6	↑
Serbian	0.6	Bulgarian	2.4	↓
Kazakh	0.4	Khanty	1.6	↑
Macedonian	0.3	Old Russian	1.6	↑
Belarusian	0.3	Belarusian	0.8	↑
Mongolian	0.2	Kazakh	0.7	↑
Old Russian	0.2	Church Slavic	0.7	↑
Undetermined	0.2	Mongolian	0.6	↑
Other	0.9	*Other*	3.3	↑

Table 6: Top-10 Cyrillic writing systems in MADLAD-400 before and after PUA filtering.

while the proportion of some of the "big" orthographies, such as Russian and Bulgarian, is reduced. Also note the more prominent presence of Old Russian, which we consider low-resource (Franklin, 1985; Schaeken, 2018).

It turns out that lower-resource orthographies can be discovered simply by manually inspecting the data. A sample of Cyrillic script paragraphs with PUA characters from 12 distinct low-resource writing systems is presented in Table 7 along with the corresponding language assignments. In addition to a Cyrillic rendition of Vedic Sanskrit shown in the table, we could identify other historic orthographies such as Middle Bulgarian and Middle/Old Russian, as well as more modern but now somewhat obsolete Cyrillic orthographies of the Azerbaijani and Uzbek languages (Ergun, 2010; Hasanova, 2020). Not all the uses of PUA characters shown in the table are linguistic. For example, while the PUA character <U+F529> in a Mansi token "Н<U+F529>врамыт" is likely a "я" with macron (U+044F + U+0304) from the original Mansi word for "children" (Balandin and Vahrusheva, 1958, p. 68), the character <U+F074> in the Adyghe token "<U+F074>Убалъэм" represents an itemized list delimiter.[11] PUA characters in Cyrillic text can frequently be traced to specific fonts, as detailed in Appendix C.

5. PUA Character Class Prediction

Determining the original semantics of PUA characters in the resulting collections is not trivial. Simple cases like tokens consisting of contiguous runs of identical PUA characters or singleton PUA characters are weeded out by the filters described in Section 3. However, even in cases of singleton PUA character tokens, how can one be certain that it does not represent a valid orthographic word in some writing system, encoded using a single Unicode code point? In general, determining the intended glyphs requires linguistic knowledge of its context. For example, deciding whether the first token in the Adyghe sentence in Table 7 has a non-orthographic vs. linguistically informative leading PUA character is straightforward based on morphology—in the Northwest Caucasian languages Kabardian and Adyghe this is a lexeme of a lemma "убалъэ" meaning "mortar". Hence, we can assume this leading character is some non-linguistic grapheme, e.g., a list delimiter or some element of graphical embellishment.

Manual disambiguation of PUA characters in large collections, as in the example above, is difficult. In this section, we investigate whether this task could be automated by an off-the-shelf LLM which, for each input paragraph in the collection, is prompted to disambiguate each PUA character in the paragraph. Given that LLMs are pre-trained on large quantities of multilingual text, the model may have enough linguistic and orthographic knowledge to correctly classify each character.

[11] Adyghe source: https://apkbr.ru/node/293.

Sample Sentence	Language		
	Name	Code	Family
<U+F074>Убалъэм ихуар къикIыжыгъуафIэ хъуркъым.	Adyghe	ady	Northwest Caucasian
<U+F0AB>Республика йәштәр мәғлүмәт үҙәге<U+F0BB>дәүләт учреждениеһы филиалы сығыш яһаны	Bashkir	bak	Turkic
Шина са<U+F019>гIе хьаьдда молла, цхьа а доцуш, висна	Chechen	che	Northeast Caucasian
Қикви морыкы, мэкгыпы а<U+F62D>қагыргын морыкы вагъэ?	Chukot	ckt	Chukotko-Kamchatkan
Налксема пак<U+10FC00>сянть ваксс тееви кирькс-те нарму<U+10FC00>нень пекстамка кардо.	Erzya	myv	Uralic
Күүкд улсас т<U+F09A>р<U+F09A>н ворошиловск мөрч болсинь Бугшан Чимидова.	Kalmyk	xal	Mongolic
Дыв “Буран” нӈемпи тӳтӓӈ хопн<U+F52B>ӈхлдӓт.	Khanty	kca	Uralic
Чилик хик<U+F049>из, къудгъун жезва цавариз.	Lezgian	lez	Northeast Caucasian
Н<U+F529>врамыт ӯщлахтын колытт л<U+F511>ккарыт вос рӯпит<U+F523>гыт.	Mansi	mns	Uralic
Садйо х<U+F115>дй аварудхйате ‘тра.	Sanskrit	san	Indo-European
<U+F034>байырлалга албан чемненир, шимченир болза эки.	Tuvan	tyv	Turkic
Олунньу 13 кҮнэ – тереебут тыл уонна сурук-бичик кҮнэ<U+F4D6>	Yakut	sah	Turkic

Table 7: Sample of MADLAD-400 Cyrillic sentences with PUA character tokens in different languages.

1. Role

You are a specialist in Unicode characters with the special expertise on private-use area (PUA) characters. The PUA character is a character in one of three Unicode ranges: `U+E000`–`U+F8FF`, `U+F0000`–`U+FFFFD`, `U+100000`–`U+10FFFD`.

2. Instructions

- The sentence below contains one or more PUA characters. The number of PUA characters in the sentence is provided to you after the sentence, separated by a semicolon.
- Determine the language of the sentence. Ignore *mojibake*.
- Find each PUA character and annotate it with one of the following 2 labels:
  1. `"LETTER"`: The character is a valid letter in a word that belongs to the language of the sentence.
  2. `"OTHER"`: The character belongs to numbers, punctuation, icons, list delimiters, emoji, symbols.

Your output should consist of only a list of labels, one for each PUA character in the sentence, preceded by analysis of each label. The labels should be output in the same order as the PUA characters in the input sentence.

3. Few-shot examples

INPUT: ⊠Убалъэм ихуар ; 1
ANALYSIS: There is one PUA character. This character is a list delimiter because it occurs at the beginning of the word “Убалъэм”, which is a valid word.
OUTPUT: `[OTHER]`

INPUT: рӯпит⊠гыт үҙәге⊠дәүләт ; 2
ANALYSIS: Two PUA characters. The first PUA character is a valid substring in a word “рӯпитәгыт”. The second PUA character is punctuation in “үҙәге» дәүләт”.
OUTPUT: `[LETTER, OTHER]`

INPUT: loo na spolupráci s⊠Michalem\nSuchánkem a⊠Richardem ; 2
ANALYSIS: Two PUA characters. The first and second PUA characters are punctuation in “s Michalem” and “a Richardem”.
OUTPUT: `[OTHER, OTHER]`

INPUT: Тыныӈ омакем, с⊠рни омакем ; 1
ANALYSIS: The PUA character in “сӧрни” is a valid letter “cyrillic o with macron”.
OUTPUT: `[LETTER]`

INPUT: ɑıU⊠uѡɔ ; 1
ANALYSIS: One PUA character. The full word without the garbled character is “tu̇uѡɔ”.
OUTPUT: `[LETTER]`

INPUT: }rdf⊠lh⊠⊠hgf fj}sgrk⊠ ; 4
ANALYSIS: Four PUA characters. This is not natural language and is probably a *mojibake*.
OUTPUT: `[OTHER, OTHER, OTHER, OTHER]`

4. Input

`SENTENCE: {{text}} ; {{num_puas}}`

Figure 1: LLM prompt for classifying PUA characters into two basic types.

$\|S\|$	N_S	N_T	A	E_U	E_O	R_C	R_L
32K	991 309	8 060 123	98.0	1.2	0.9	1.9	40.7
512	954 470	6 537 913	98.2	1.0	0.8	1.9	45.9
256	916 663	5 526 594	98.4	0.9	0.7	1.8	44.1
128	875 300	4 785 262	98.5	0.8	0.7	1.7	41.3
64	817 177	4 021 097	98.7	0.7	0.6	1.6	36.6

Table 8: LLM instruction following: input length limit ($|S|$), number of paragraphs N_S satisfying the limit, total number of PUA characters N_T observed, and the values of five corresponding metrics.

We judge the suitability of LLM for this task in two ways. First, we assess whether the number of predictions generated by the model matches the number of PUAs in the input paragraph that are to be disambiguated. Being able to produce the same number of labels as there are PUA characters in the input paragraph is a necessary precondition for using an LLM for PUA character class prediction. Second, we assess an LLM's ability to disambiguate PUA characters into those that encode LETTERs, diacritics, or other linguistic characters, or OTHER non-linguistic characters such as list delimiters, whitespace, or unintentional corruption. We judge the performance of the LLM with respect to a 100 paragraph reference set, doubly annotated and reconciled manually by three of the paper authors.

We use Gemini 2.5 Flash (Comanici et al., 2025) as the off-the-shelf LLM this experiment. Gemini 2.5 Flash is a multi-billion parameter transformer model (Vaswani et al., 2017), using a sparse mixture-of-experts (Shazeer et al., 2017) with a long context window and thinking capabilities. We frame the task as binary classification of PUA character types—valid LETTERs in a word or OTHER forms of punctuation or non-linguistic graphemes—using few-shot prompting (Brown et al., 2020) with six examples, as shown in Figure 1. For each example (in Latin, Cyrillic and Thai scripts), in addition to input (a paragraph), the number of relevant PUA characters, and corresponding model output (a list of types, one per-character), we include a sample of an intermediate manual reasoning stage to guide the model (Wei et al., 2022). We arrived at this version of the prompt after several iterations of prompt engineering. Devising a more detailed taxonomy of character tags (e.g., general punctuation, other forms of delimiters, *emoji*) or restoring the originally intended characters where possible, introduces additional complexity beyond our relatively simple current setup, which we leave for future work.

Matching the Correct Label Count For this evaluation, we examine the smallest of four collections of paragraphs, corresponding to DCAD-2000 with token-internal PUA characters. We evaluate how well the LLM follows the instructions given by the prompt in Figure 1 by comparing the number of PUA characters in each input paragraph with the number of character classes (LETTER or OTHER) produced by the model. The findings are presented in Table 8. Each row in the table corresponds to the maximum number of tokens in the input paragraph denoted $|S|$. For each configuration, the total number of paragraphs N_S satisfying the input sequence limit and the total number of PUA characters N_T observed in this data are shown along with five additional metrics. The first is the measure A of how well the model adheres to instructions, *i.e.*, the percentage of paragraphs where the number of input PUA characters in a sequence matches the number of predicted classes. Two additional complementary metrics measure the percentage of sequence mismatches, where the number of per-sequence predicted character classes N_P^s is either under-generated E_U ($N_P^s<N_T^s$) or over-generated E_O ($N_P^s>N_T^s$). The measure R_C is a ratio between the total number of generated character classes N_P and the total number of PUA characters observed in the data N_T. Finally, R_L represents the percentage of LETTER class predictions of the total number of generated classes.

The adherence rate A shown in Table 8 is quite high, ranging from 98.9% for paragraphs less than 64 tokens long to 98.7% considering paragraphs in their entirety. While the LLM tends to under/over-predict at similar rates (E_U=1.2% vs. E_O=0.9%), it tends to generate many more labels than expected when it overpredicts, as evidenced by R_C=1.9. Overall, the LLM tends to label PUA characters as LETTER 40.7% of the time. While the LLM does not adhere perfectly to the instructions, it adheres close enough that one could consider using it to resolve PUA character class.

We also compute the above measures on a paragraph script basis in Appendix D. We determine the script from the top language-script hypothesis for a paragraph provided by the LID model. There are 156 scripts overall reported by the LID model for the paragraphs in the collection analyzed in this section. Among the highest frequency scripts reported in Table 2, the model adheres closely to the instructions for all of them. The only exception is the Han (Hani) script, for which the model only has 90.9% adherence. For the other scripts that adherence rate ranges from 96.8% for Thai up to 99.5% for Arabic. While poor adherence in Hani examples may be a product of relatively little pre-training data, the adherence rates could also be a function of the example length and number of PUA characters present per paragraph by script. More PUA characters per paragraph and longer paragraphs make it harder for the LLM to adhere to its instruc-

gold/pred	letter	other
letter	192	10
other	3	180

Table 9: Confusion matrix for predicting character class of individual PUA characters.

tions. Similar to our observations above, adherence decreases slightly as the limit on the number of input sequence tokens is raised. For example, adherence decreases for Latin script paragraphs from 98.9% when limiting to 64-token examples to 98.2% for all paragraphs.

LLM prompt wording is critical in producing the correct number of labels. During our exploration, detailed in Appendix E, we found that using a prompt that does not explicitly include the number of PUA characters in the input paragraph results in only 64.8% of examples having the correct number of predicted labels, with 72.0% adherence even with 64-token inputs. In addition, this prompt yielded an R_C of 31.7, suggesting massive overgeneration of predicted labels on average. Without experimenting with LLM prompts, mismatch in predicted label count would preclude using LLMs to solve this task at even the most basic level.

Predicting Character Class We sampled 100 paragraphs from the set of paragraphs where the LLM adhered to the labeling instructions, restricted to a maximum character length of 2,000 and number of PUA characters of 20. Over 95% of paragraphs satisfied these two criteria, and they limited the difficulty of the annotation task. For each PUA, in each paragraph, two annotators manually labeled its character class independently. Any disagreements were reconciled by discussion after an independent annotation. Before discussion, annotators had an agreement rate of 91% at the paragraph level and 97.1% at the PUA character level.

Table 9 gives the confusion matrix for the LLM's character-level class prediction. Overall, the LLM achieves 91% paragraph-level accuracy and 96.6% at the character level. This compares favorably with the annotator agreement rate prior to reconciliation: 91% at the paragraph level and 97.1% at the character level. Note that a majority classifier, labeling each PUA as LETTER would have achieved only 52.5% accuracy at this binary prediction task. While the LLM appears to be able to resolve PUA characters to character class at human agreement rates, resolution of PUA to glyph remains untested.

It is worth noting that only 2 out of the 100 paragraphs were labeled heterogeneously, i.e., with both LETTER and OTHER classes being assigned at least once to PUA characters in the paragraph. This suggests that for most instances, predicting a single label of LETTER vs. OTHER for all PUA characters in a paragraphs is sufficient. The two examples with heterogeneous labels were: "*B*☒☒☒☒☒☒☒*F*☒☒☒☒☒*JM AND BRAZ-F* ☒☒☒☒☒*R. 2001. Bowdenol*", a reference where most PUA characters encode the letters in the authors' names, but some represent token delimiters and "*stylem, logem i*☒*typogra*☒*í je Studio Najbrt.*", where the first character is a non-breaking whitespace character typically inserted after Czech prepositions and the second is likely a Latin letter "f".

6. Conclusions

This work explored the landscape of Unicode PUA characters within MADLAD-400 and DCAD-2000, two large-scale, web-crawled corpora frequently used to pre-train popular LLMs. To extract this data, we introduced a simple pipeline using heuristic-based filters, alongside two position-based strategies for defining "valid" PUA tokens. Our analysis revealed that the extracted paragraph-level data exhibits notable rank-frequency and rank-proportion distributions across scripts and writing systems—a dynamic we highlighted using under-represented Cyrillic orthographies. Although PUA paragraphs constitute a minute fraction of the overall data, they hold significant potential for under-represented languages, provided the identity of underlying glyphs can be accurately resolved. To address this, we tested an off-the-shelf LLM's ability to classify PUA character data into orthographic versus non-linguistic graphemes. We found that Gemini 2.5 Flash generates well-formed predictions for 98% of input paragraphs and distinguished between orthographic and non-orthographic PUA usage with 96.6% accuracy on a small sample, closely tracking our human agreement rate of 97.1%. While these initial classification results are highly encouraging, the ability of current LLMs to reliably resolve PUA characters to their exact intended graphemes remains an open question for future research.

Ideally some dynamic community resource would become available to document attested and/or discovered PUA assignments, with detailed information about the intended grapheme (and perhaps the origin of the assignment), to assist in grapheme recovery when links to fonts are lost. Such a resource would have to be dynamic to allow for ongoing use of PUA characters – in contrast to some of the more static (and sometime stale) resources cited earlier in the paper – and would likely be augmented through semi-automatic means, i.e., computer-assisted discovery and labeling, followed by human validation. This paper presents a hint of what at least some part of such an undertaking would entail.

7. Limitations

Without reliable automated tools, resolving PUA characters like those in Table 7 (Section 4) remains a time-consuming and brittle manual process. Establishing a standard methodology is difficult because each instance demands a unique approach. For example, identifying a Mansi PUA character as a valid letter required running the paragraph through a search engine, examining matching documents to find the context in which the keyword appears, and finally looking it up in an online dictionary.[12] In contrast, identifying the Adyghe PUA character as a list delimiter simply required locating the source newspaper article and examining its visual layout. As reviewers noted, incorporating richer metadata context could significantly improve both this manual effort and the automated LLM-based pipeline (Section 5). Valuable metadata signals include source URLs, web-page font specifications, and established PUA character allocation registries (Section 2).

We presented an exploratory LLM-based approach that classifies PUA characters into two broad categories. While this serves as a foundational step, a significantly more useful disambiguation system would go beyond these broad classes toward a finer-grained character taxonomy. As noted by a reviewer, a proper grapholinguistic analysis requires establishing the graphetic identity of the glyph, its script affiliation, and its relationship to existing Unicode code points (*e.g.*, distinguishing between a historically attested character awaiting encoding and an idiosyncratic font shortcut). These nuanced requirements significantly complicate prompt structure. It remains to be seen whether modern LLMs possess sufficient internal grapholinguistic knowledge to resolve this information independently, without resorting to the external metadata discussed earlier. Consequently, rigorous prompt engineering will remain critical—as demonstrated in this paper, iterative prompt refinement is essential for significantly improving instruction adherence even in binary classification tasks.

Accurate and detailed models to disambiguate PUA characters will ultimately facilitate NLP and grapholinguistic research across several avenues: (1) mining new sources of fully disambiguated, clean text for low-resource languages from large-scale web corpora for inclusion in NLP models; (2) constructing data-driven knowledge graphs of grapholinguistic data encountered in the wild; and (3) augmenting, cross-referencing, and tagging hand-curated PUA character allocation registries. Finally, as aptly noted by a reviewer, the development of robust PUA disambiguation methods is crucial for the primary users of these characters—the communities actively developing or reforming their orthographies.

[12]The most detailed results are returned by Yandex search engine for such queries. Also, finding an online dictionary for a low-resource language is challenging.

8. Ethics Statement

The goal of this work was to provide a preliminary study of the Unicode Private-Use Area characters as found in online data. This type of data tends to be treated as noise and ignored in natural language processing yet, as we argue, may provide a useful source of linguistic signals. Developing techniques to categorize such characters and "fix" them by automatically providing valid Unicode alternative(s), where possible, may help provide additional sources of clean data in scenarios where the need for such data is acutely felt, especially for under-represented orthographies or orthographies "in flux".

9. Acknowledgements

The authors thank Cibu Johny and the anonymous reviewers for many useful comments on this paper.

10. Bibliographical References

Robert Alessi and Antonis Tsolomitis. 2023. Old Standard: A Unicode font for classical and medieval studies. Technical Report v2.7–2023/12/15, The Comprehensive TeX Archive Network.

Deborah Anderson. 2018. Bridging the divide: Supporting the minority and historic scripts in fonts: Problems and recommendations. In *Proceedings of the 2018 Digital Humanities Conference (DH2018)*, pages 328–330, Mexico City, Mexico.

Aleksandr Andreev and Nikita Simmons. 2020. Church Slavonic fonts. Technical Report 2.2, Ponomar Project, Slavonic Computing Initiative.

A. N. Balandin and M. P. Vahrusheva. 1958. *Mansi-Russian dictionary with lexical parallels from Southern Mansi (Kondin) dialect*. Ministry of Education of Russian Soviet Federative Socialist Republic (RSFSR), Leningrad, USSR. In Russian. Russian title: *Mansijsko-russkij slovar' s leksicheskimi parallelyami iz yuzhno-mansijskogo (kondinskogo) dialekta*.

Ankur Bapna, Isaac Caswell, Julia Kreutzer, Orhan Firat, Daan van Esch, Aditya Siddhant,

Mengmeng Niu, Pallavi Baljekar, Xavier Garcia, Wolfgang Macherey, Theresa Breiner, Vera Axelrod, Jason Riesa, Yuan Cao, Mia Xu Chen, Klaus Macherey, Maxim Krikun, Pidong Wang, Alexander Gutkin, Apurva Shah, Yanping Huang, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2022. Building machine translation systems for the next thousand languages. *arXiv preprint arXiv:2205.03983*.

Biligsaikhan Batjargal, Garmaabazar Khaltarkhuu, Fuminori Kimura, and Akira Maeda. 2011. A study of traditional Mongolian script encodings and rendering: Use of Unicode in OpenType fonts. *International Journal on Asian Language Processing*, 21(1):23–44.

Adrian Benton, Alexander Gutkin, Christo Kirov, and Brian Roark. 2026. Mining naturally romanized seed corpora without romanizations. In *Proceedings of the 2026 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC 2026)*, Palma de Mallorca, Spain. Language Resources Association (ELRA). To appear.

Steven Bird and Gary Simons. 2003. Seven dimensions of portability for language documentation and description. *Language*, 79(3):557–582.

Rogier Blokland. 2023. Zyrian Komi. In *The Uralic Languages*, 2nd edition, chapter 14, pages 614–664. Routledge, London, UK.

Jeremy Bradley and Rogier Blokland. 2023. Mansi et al. in print before and under Unicode. *Linguistica Uralica*, 59(4):243–257.

Jeremy Bradley and Elena Skribnik. 2021. The many writing systems of Mansi: challenges in transcription and transliteration. *Multilingual Facilitation*, pages 12–24.

Geoffrey Brown and Kam Woods. 2009. Born broken: Fonts and information loss in legacy digital documents. In *Proceedings of 6th International Conference on Preservation of Digital Objects (iPRES)*, pages 30–37, San Francisco, California.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of the 34th Conference on Advances in Neural Information Processing Systems (NeurIPS 2020)*, volume 33, pages 1877–1901. Curran Associates, Inc.

Isaac Caswell, Theresa Breiner, Daan van Esch, and Ankur Bapna. 2020. Language ID in the wild: Unexpected challenges on the path to a thousand-language web text corpus. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6588–6608, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Ralf Cleminson, Victor Baranov, Achim Rabus, David Birnbaum, and Heinz Mitlas. 2010. Proposal for a unified encoding of Early Cyrillic glyphs in the Unicode Private Use Area. *Scripta & e-Scripta: the Journal of Interdisciplinary Mediaeval Studies*, 8:9–26. Sofia: Bulgarian Academy of Sciences.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Asa Cooper Stickland, Sailik Sengupta, Jason Krone, Saab Mansour, and He He. 2023. Robustification of multilingual language models to real-world noise in crosslingual zero-shot settings with robust contrastive pretraining. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 1375–1391, Dubrovnik, Croatia. Association for Computational Linguistics.

Quinn Dombrowski, Manish Goregaokar, Ben Joeng (Yang), and Abeera Kamran. 2024. Encoding multilingualism: Technical affordances of multilingual publication from manuscripts to Unicode and OpenType. *The Journal of Electronic Publishing (JEP)*, 27(1):309–329.

António Emiliano. 2012. Issues in the typographic representation of medieval primary sources. In Yuji Kawaguchi, Makoto Minegishi, and Wolfgang Viereck, editors, *Corpus-based Analysis and Diachronic Linguistics*, Tokyo University of Foreign Studies. Studies in Linguistics 3, pages 153–173. John Benjamins Publishing Company.

Ayça Ergun. 2010. Politics of romanisation in Azerbaijan (1921–1992). *Journal of the Royal Asiatic Society*, 20(1):33–48.

Michael Everson. 2008. Phaistos ConScript to Unicode table. ConScript Unicode Registry (CSUR). Table version 0.01, Unicode version 5.1.

Simon Franklin. 1985. Literacy and documentation in early medieval Russia. *Speculum*, 60(1):1–38.

Yannis Haralambous. 2007. *Fonts & Encodings*. O'Reilly Media, Sebastopol, CA.

Dilia Hasanova. 2020. Linguistic landscape of Uzbekistan: The rise and fall of Uzbek, Russian, Tajik, and English. In Stanley D. Brunn and Roland Kehrein, editors, *Handbook of the Changing World Language Map*, pages 3015–3028. Springer International Publishing, Cham, Switzerland.

Odd Einar Haugen. 2013. Dealing with glyphs and characters: Challenges in encoding medieval scripts. *Document numérique*, 16(3):97–111.

Anushah Hossain. 2024. Text standards for the "rest of world"': The making of the Unicode standard and the OpenType format. *IEEE Annals of the History of Computing*, 46(1):20–33.

ISO. 2022. ISO 15924: Codes for the representation of names of scripts. https://www.iso.org/obp/ui/#iso:std:iso:15924:ed-2:v1:en. International Organization for Standardization, Technical Committee ISO/TC 46, Information and Documentation.

Pratik Joshi, Christain Barnes, Sebastin Santy, Simran Khanuja, Sanket Shah, Anirudh Srinivasan, Satwik Bhattamishra, Sunayana Sitaram, Monojit Choudhury, and Kalika Bali. 2019. Unsung challenges of building and deploying language technologies for low resource language communities. In *Proceedings of the 16th International Conference on Natural Language Processing*, pages 211–219, International Institute of Information Technology, Hyderabad, India. NLP Association of India.

Fatih Karagöz, Berat Doğan, and Şaziye Betül Özateş. 2024. Towards a clean text corpus for Ottoman Turkish. In *Proceedings of the First Workshop on Natural Language Processing for Turkic Languages (SIGTURK 2024)*, pages 62–70, Bangkok, Thailand and Online. Association for Computational Linguistics.

Amir Hossein Kargaran, Ayyoob Imani, François Yvon, and Hinrich Schuetze. 2023. GlotLID: Language identification for low-resource languages. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6155–6218, Singapore. Association for Computational Linguistics.

Amir Hossein Kargaran, François Yvon, and Hinrich Schütze. 2024. GlotCC: An open broad-coverage commoncrawl corpus and pipeline for minority languages. In *The Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada.

Sebastian Kempgen. 2008. Unicode 5.1, Old Church Slavonic, remaining problems–and solutions, including OpenType features. In *Slovo: Towards a Digital Library of South Slavic Manuscripts; Proceedings of the International Conference*, Sofia, Bulgaria. Bulgarian Academy of Sciences.

András Kornai. 2013. Digital language death. *PloS One*, 8(10):e77056.

Alexey Kryukov. 2011. Old Standard: A Unicode font for classical and medieval studies. Technical Report 2.3, The Comprehensive TeX Archive Network. Edited for CTAN by Bob Tennent.

Sneha Kudugunta, Isaac Caswell, Biao Zhang, Xavier Garcia, Derrick Xin, Aditya Kusupati, Romi Stella, Ankur Bapna, and Orhan Firat. 2023. MADLAD-400: A multilingual and document-level large audited dataset. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:67284–67296.

Gurpreet Singh Lehal, Tejinder Singh Saini, and Savleen Kaur Chowdhary. 2012. An omni-font Gurmukhi to Shahmukhi transliteration system. In *Proceedings of COLING 2012: Demonstration Papers*, pages 313–320, Mumbai, India. The COLING 2012 Organizing Committee.

Peiqin Lin, Shaoxiong Ji, Jörg Tiedemann, André F. T. Martins, and Hinrich Schütze. 2024. MaLA-500: Massive language adaptation of large language models. *arXiv preprint arXiv:2401.13303*.

Elisabeth Maria Magin and Marcus Smith. 2023. "(R)Unicode: Encoding and sustainability issues in runology". *Digital Humanities in the Nordic and Baltic Countries Publications (DHNB)*, 5(1):121–136.

Gurjot Singh Mahi and Amandeep Verma. 2015. Wrecked Indian fonts: A problem for digitalization of Indic documents. In *Proceedings of 60th International Conference on Embedded Librarianship and Technological challenges in Digital Age*, pages 905–914, Chandigarh, India. Indian Library Association.

Yan Meng, Di Wu, and Christof Monz. 2025. How to learn in a noisy world? self-correcting the real-world data noise in machine translation. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 7451–7467, Albuquerque, New Mexico. Association for Computational Linguistics.

Janusz Marian Nowacki. 2005. Antykwa Toruńska. Technical Report 2.03, The Polish TEX Users Group, GUST. Type designer: Zygfryd Gardzielewski, font author: Janusz Marian Nowacki.

Anshuman Pandey. 2015. Proposal to encode the Hanifi Rohingya script in Unicode. 2015-10-27 L2/15-278, Unicode Consortium.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. 2025. FineWeb2: One pipeline to scale them all–adapting pre-training data processing to every language. *arXiv preprint arXiv:2506.20920*.

Aidan Pine and Mark Turin. 2018. Seeing the Heiltsuk orthography from font encoding through to Unicode: A case study using Convertextract. In *Proceedings of the LREC 2018 Workshop "CCURL 2018–Sustaining knowledge diversity in the digital age"*, pages 27–30, Miyazaki, Japan. European Language Resources Association.

Lorna A. Priest. 2007. Unicode on the front lines: Endangered languages and Unicode. In *31st Internationalization and Unicode Conference*, pages 1–24, San José, CA. The Object Management Group, Standards Development Organization. Presentation.

Pim Rietbroek. 2014. The Brill typeface user guide & complete list of characters. Technical Report Brill Typeface Version 2.06, Brill, Leiden, Netherlands.

Pim Rietbroek. 2021. Brill typeface version 4.0 character list. Technical Report Brill Typeface Version 4.0, De Gruyter Brill, Leiden, Netherlands.

Jos Schaeken. 2018. *Voices on birchbark: Everyday communication in medieval Russia*, volume 43 of *Studies in Slavic and General Linguistics*. De Gruyter Brill, Leiden, The Netherlands.

Julia Schillo and Mark Turin. 2022. Type right: Examining the underlying causes of common typeface and font errors for Indigenous orthographies, and a possible path forward. *Language Documentation & Conservation*, 16:364–398.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of 5th International Conference on Learning Representations*, Toulon, France. Poster.

Yingli Shen, Wen Lai, Shuo Wang, Xueren Zhang, Kangyang Luo, Alexander Fraser, and Maosong Sun. 2025. DCAD-2000: A multilingual dataset across 2000+ languages with data cleaning as anomaly detection. *arXiv preprint arXiv:2502.11546*.

Gary F. Simons, Abbey L. L. Thomas, and Chad K. K. White. 2022. Assessing digital language support on a global scale. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4299–4305, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Gianluca Sperduti and Alejandro Moreo. 2025. Misspellings in natural language processing: A survey. *arXiv preprint arXiv:2501.16836*.

Nicolas Stefanovitch. 2022. Recovering text from endangered languages corrupted PDF documents. In *Proceedings of the Fifth Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 78–82, Dublin, Ireland. Association for Computational Linguistics.

Unicode Consortium. 2024. The Unicode Standard (version 16.0.0). Technical report, Unicode Consortium, South San Francisco, CA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, pages 5998–6008, Long Beach, CA. Curran Associates Inc.

Bin Wang, Chengwei Wei, Zhengyuan Liu, Geyu Lin, and Nancy F. Chen. 2024. Resilience of large language models for noisy instructions. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 11939–11950, Miami, Florida, USA. Association for Computational Linguistics.

Boli Wang, Xiaodong Shi, and Yidong Chen. 2016. Coping with problems of Unicoded traditional Mongolian. In *Proceedings of the 4th International Symposium on Natural Language Processing Based on Naturally Annotated Big Data*, pages 125–131, Yantai, China. Springer-Verlag, Berlin, Heidelberg.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th Conference on Advances in Neural Information Processing Systems (NeurIPS 2022)*, volume 35, pages 24824–24837, New Orleans, LA, USA. Curran Associates, Inc.

Christopher Wyrod. 2008. A social orthography of identity: the N'ko literacy movement in West Africa. *International Journal of the Sociology of Language*, 2008(192).

Irene Yi and Claire Bowern. 2023. FileLingR: An R script validation tool for depositors and users of digital language collections. In *Proceedings of the Sixth Workshop on the Use of Computational Methods in the Study of Endangered Languages*, pages 84–88, Remote. Association for Computational Linguistics.

Isabelle A. Zaugg, Anushah Hossain, and Brendan Molloy. 2022. Digitally-disadvantaged languages. *Internet Policy Review*, 11(2):1–11.

A. PUA Tokens Across Scripts: Full Rank-Frequency Distribution for Absolute Counts

Table 2 provided the top-5 items of the rank-frequency distribution of PUA character tokens across scripts. The remaining scripts were grouped under the "Other" category (Table 3). This distribution is based on the absolute counts of tokens with PUA characters in the relevant corpora under various PUA character placement conditions.

This appendix provides a fuller picture of this distribution by including the PUA token distributions across all scripts essentially "expanding" the "Other" bin. The distributions for tokens with PUA characters for the permissive (*Anywhere*) PUA character placement condition across scripts are shown in Table 10 for MADLAD-400 (left) and DCAD-2000 (right). As can be seen from the tables, MADLAD-400 has a greater variety of scripts than DCAD-2000. For the restrictive PUA character placement condition (*Internal*) the distributions of PUA character tokens across scripts are shown in Table 11 for MADLAD-400 (left-hand side) and DCAD-2000 (right-hand side).

B. PUA Tokens and Paragraphs Across Scripts and Languages: Rank-Proportion Distribution

In our analysis in Section 4, the distributions considered in Tables 2 to 5, as well as the supplementary data in Appendix A, involve rank-frequency distributions based on absolute counts of PUA character tokens or paragraphs containing such tokens. We found that for these distributions the "interesting" low-resource script and writing system signals are mostly confined to the "Other" bin because the heads of the distributions are dominated by well-resourced scripts and languages.

Using the permissive PUA character placement strategy for MADLAD-400 we also computed the ratio of PUA tokens to all the original tokens for each of the scripts. Table 12 shows the ranking of all scripts with over 100,000 original (non-PUA) tokens ranked in terms of decreasing proportion of the number of PUA tokens, shown in the third column, to the number of all the original tokens for each script. Beyond the tokens recognized as code (`Zinh` and `Zyyy`), the top-ranking scripts include liturgical (Coptic, `Copt` and Syriac, `Syrc`), historic (Ancient Egyptian, `Egyp`) and Southeast Asian scripts, such as Khmer, Lao and Thai. Interestingly, the well-resourced scripts, such as Latin, Cyrillic and Arabic rank significantly lower.

For the same permissive PUA character placement in the MADLAD-400 configuration we computed the ratio of PUA paragraphs to all the original paragraphs for each of the language-script pairs emitted by GlotLID, where the count of original paragraphs exceeds 5,000. The results are shown in Table 13, where the rows are sorted according to decreasing PUA paragraph ratio. The top-ranking language-script pairs are completely dominated by low-resource (Mansy, Khanty, Tsakhur, Abaza and Northern Kurdish) orthographies in the Cyrillic script, among othera. There is a strong presence of historic orthographies such as Coptic and Church Slavic, as well as an unknown language in Old Permic script, which is likely medieval Komi (Blokland, 2023).

C. Legacy and Modern Cyrillic Fonts

Beyond specialized fonts from academic publishers, such as Brill fonts mentioned in Section 2, the use of PUA characters is prevalent in public Cyrillic fonts available with most modern operating systems. Some of the fonts distributed with the popular *TeX Live* typesetting package that include support for Cyrillic script are shown in Table 14.[13] The table shows four popular and well-maintained font

[13] https://tug.org/texlive/

Code	Name	Count
Latn	Latin	32 240 234
Cyrl	Cyrillic	2 803 337
Thai	Thai	2 672 535
Hani	Han (Hanzi, Kanji, Hanja)	825 219
Arab	Arabic	507 019
Hang	Hangul (Hangŭl, Hangeul)	423 992
Grek	Greek	240 064
Hebr	Hebrew	95 085
Deva	Devanagari (Nagari)	76 671
Zinh	Code for inherited script	45 175
Taml	Tamil	44 904
Zyyy	Code for undetermined script	19 883
Beng	Bengali	17 326
Khmr	Khmer	14 829
Armn	Armenian	13 625
Kana	Katakana	11 270
Geor	Georgian (Mkhedruli)	11 022
Mlym	Malayalam	5812
Hira	Hiragana	5232
Copt	Coptic	4628
Laoo	Lao	4092
Knda	Kannada	3992
Sinh	Sinhala	3909
Gujr	Gujarati	3653
Ethi	Ethiopic (Ge'ez)	2163
Telu	Telugu	1901
Guru	Gurmukhi	1761
Mymr	Myanmar (Burmese)	1544
Cans	Unified Canadian Aboriginal Syllabics	1524
Syrc	Syriac	403
Orya	Oriya	280
Glag	Glagolitic	241
Ogam	Ogham	161
Cher	Cherokee	145
Tibt	Tibetan	143
Tfng	Tifinagh (Berber)	131
Lana	Tai Tham (Lanna)	80
Bopo	Bopomofo	70
Linb	Linear B	55
Thaa	Thaana	43
Egyp	Egyptian hieroglyphs	35
Nkoo	N'Ko	29
Bali	Balinese	29
Yiii	Yi	26
Java	Javanese	19
Mong	Mongolian	16
Runr	Runic	13
Samr	Samaritan	13
Saur	Saurashtra	11
Sund	Sundanese	9
Bamu	Bamum	9
Lisu	Lisu (Fraser)	4
Merc	Meroitic Cursive	3
Cham	Cham	3
Bass	Bassa Vah	2
Xsux	Cuneiform, Sumero-Akkadian	2
Tale	Tai Le	2
Limb	Limbu	2
Rohg	Hanifi Rohingya	2
Hung	Old Hungarian (Hungarian Runic)	1
Tglg	Tagalog (Baybayin, Alibata)	1
Hano	Hanunoo (Hanunóo)	1
Tavt	Tai Viet	1
Kali	Kayah Li	1
Rjng	Rejang (Redjang, Kaganga)	1
Mtei	Meitei Mayek (Meithei, Meetei)	1
All		40 104 389

Code	Name	Count
Latn	Latin	12 835 255
Cyrl	Cyrillic	1 071 969
Grek	Greek	644 169
Arab	Arabic	396 244
Thai	Thai	296 941
Hang	Hangul (Hangŭl, Hangeul)	166 927
Zinh	Code for inherited script	133 628
Hani	Han (Hanzi, Kanji, Hanja)	42 923
Hebr	Hebrew	36 066
Deva	Devanagari (Nagari)	27 268
Armn	Armenian	24 780
Knda	Kannada	21 990
Ethi	Ethiopic (Ge'ez)	21 144
Sinh	Sinhala	19 588
Khmr	Khmer	19 261
Geor	Georgian (Mkhedruli)	13 067
Gujr	Gujarati	12 891
Laoo	Lao	12 572
Mlym	Malayalam	10 966
Mymr	Myanmar (Burmese)	8764
Java	Javanese	8428
Taml	Tamil	7143
Guru	Gurmukhi	6623
Beng	Bengali	5251
Telu	Telugu	4318
Orya	Oriya	3977
Zyyy	Code for undetermined script	3945
Lana	Tai Tham (Lanna)	2295
Copt	Coptic	1305
Kana	Katakana	510
Cans	Unified Canadian Aboriginal Syllabics	291
Thaa	Thaana	197
Tibt	Tibetan	118
Egyp	Egyptian hieroglyphs	82
Mong	Mongolian	76
Hira	Hiragana	33
Syrc	Syriac	29
Nkoo	N'Ko	10
Sund	Sundanese	9
Cher	Cherokee	6
Tfng	Tifinagh (Berber)	6
Yiii	Yi	5
Dsrt	Deseret (Mormon)	2
Lina	Linear A	2
Tang	Tangut	2
Bopo	Bopomofo	2
Bamu	Bamum	2
Vaii	Vai	1
Xsux	Cuneiform, Sumero-Akkadian	1
Limb	Limbu	1
Batk	Batak	1
Runr	Runic	1
Mero	Meroitic Hieroglyphs	1
Bugi	Buginese	1
All		15 861 087

Table 10: The full rank-frequency distribution of PUA character tokens in MADLAD-400 (left) and DCAD-2000 (right) with *permissive* PUA character placement: ISO 15924 script codes, names and token counts.

Code	Name	Count
Latn	Latin	11 739 280
Thai	Thai	2 092 819
Cyrl	Cyrillic	1 266 526
Hani	Han	440 618
Arab	Arabic	161 666
Hang	Hangul	122 167
Hebr	Hebrew	51 355
Grek	Greek	35 904
Deva	Devanagari	25 680
Taml	Tamil	18 031
Khmr	Khmer	10 958
Beng	Bengali	4208
Laoo	Lao	3417
Copt	Coptic	3271
Geor	Georgian	1590
Zyyy	Code	1584
Kana	Katakana	1449
Mlym	Malayalam	1324
Zinh	Code	911
Telu	Telugu	844
Hira	Hiragana	758
Cans	Unified	568
Gujr	Gujarati	568
Sinh	Sinhala	548
Guru	Gurmukhi	400
Knda	Kannada	388
Mymr	Myanmar	336
Ethi	Ethiopic	248
Orya	Oriya	186
Armn	Armenian	170
Glag	Glagolitic	157
Syrc	Syriac	137
Ogam	Ogham	80
Tfng	Tifinagh	44
Tibt	Tibetan	23
Lana	Tai	18
Egyp	Egyptian	14
Cher	Cherokee	13
Runr	Runic	9
Bopo	Bopomofo	5
Mong	Mongolian	5
Samr	Samaritan	4
Nkoo	N'Ko	3
Rohg	***	2
Bamu	Bamum	2
Tavt	Tai	1
Sund	Sundanese	1
Cham	Cham	1
Thaa	Thaana	1
Bali	Balinese	1
All		15 988 293

Code	Name	Count
Latn	Latin	1 283 007
Cyrl	Cyrillic	464 904
Thai	Thai	144 511
Arab	Arabic	115 386
Hang	Hangul	67 066
Hebr	Hebrew	23 255
Khmr	Khmer	12 373
Grek	Greek	11 775
Ethi	Ethiopic	10 594
Java	Javanese	8427
Hani	Han	8390
Laoo	Lao	7931
Deva	Devanagari	7057
Mymr	Myanmar	3382
Sinh	Sinhala	3342
Orya	Oriya	3058
Guru	Gurmukhi	2879
Knda	Kannada	2043
Gujr	Gujarati	1760
Zinh	Code	1585
Mlym	Malayalam	1295
Geor	Georgian	1293
Lana	Tai	1220
Telu	Telugu	1093
Copt	Coptic	963
Beng	Bengali	892
Taml	Tamil	731
Zyyy	Code	495
Armn	Armenian	433
Mong	Mongolian	52
Cans	Unified	46
Tibt	Tibetan	29
Kana	Katakana	17
Sund	Sundanese	9
Bopo	Bopomofo	2
Hira	Hiragana	2
Thaa	Thaana	1
Xsux	Cuneiform	1
All		2 191 299

Table 11: The full rank-frequency distribution of PUA character tokens in MADLAD-400 (left) and DCAD-2000 (right) with *word-internal* PUA character placement: ISO 15924 script codes, names and token counts.

families containing multiple fonts with a good support for Cyrillic glyphs (among other scripts, if Latin and Greek are also provided). However, a lot of these glyphs are mapped to PUA characters as can be seen from the values of population mean μ, which represents the average number of PUA characters per font. It is likely that a reasonably high proportion of such characters end up in the resulting digital documents that use these fonts and therefore introduce character-level noise into the web-crawled corpora.

Arguably the widest coverage of modern Cyrillic writing systems is provided by the ParaType type foundry. According to the ParaType metadata, 81 living Cyrillic writing systems are supported by its

Code	Name	Ratio (%)	Tokens PUA	Tokens Orig.
Copt	Coptic	0.896	4628	516 649
Zinh	Code	0.326	45 175	13 871 088
Tibt	Tibetan	0.092	143	155 482
Khmr	Khmer	0.066	14 829	22 302 230
Bopo	Bopomofo	0.058	70	119 873
Zyyy	Code	0.047	19 883	41 876 698
Thai	Thai	0.047	2 672 535	5 643 856 638
Cher	Cherokee	0.034	145	429 135
Egyp	Egyptian	0.026	35	135 049
Laoo	Lao	0.019	4092	21 654 614
Syrc	Syriac	0.015	403	2 685 983
Tfng	Tifinagh	0.013	131	1 011 416
Cans	Unified	0.012	1524	12 906 115
Mong	Mongolian	0.011	16	141 788
Hani	Han	0.009	825 219	9 433 282 806
Hang	Hangul	0.004	423 992	9 938 441 074
Hira	Hiragana	0.003	5232	173 292 558
Taml	Tamil	0.003	44 904	1 582 361 376
Mymr	Myanmar	0.003	1544	55 233 135
Armn	Armenian	0.002	13 625	725 541 909
Ethi	Ethiopic	0.002	2163	125 084 695
Geor	Georgian	0.001	11 022	848 345 475
Hebr	Hebrew	0.001	95 085	7 406 031 175
Grek	Greek	0.001	240 064	20 328 695 640
Kana	Katakana	0.001	11 270	1 038 104 139
Deva	Devanagari	0.001	76 671	7 791 717 690
Latn	Latin	0.001	32 240 234	3 349 046 897 307
Knda	Kannada	0.001	3992	423 791 112
Sinh	Sinhala	0.001	3909	424 759 844
Beng	Bengali	0.001	17 326	2 103 453 752
Cyrl	Cyrillic	0.001	2 803 337	344 698 119 171
Mlym	Malayalam	0.001	5812	763 444 007
Arab	Arabic	0.001	507 019	69 314 190 045
Gujr	Gujarati	0.001	3653	527 343 472
Guru	Gurmukhi	0.001	1761	261 678 655
Orya	Oriya	0.001	280	55 081 256
Telu	Telugu	0.000	1901	683 656 693
Thaa	Thaana	0.000	43	77 112 730

Table 12: The proportion of PUA *tokens* (shown in the third column) to all tokens for 38 scripts that have over 100,000 tokens, ranked in decreasing order.

Lang.	Script	Lang. Name	Ratio (%)	PUA	Original
mns	Cyrl	Mansi	32.344	31 639	97 819
kca	Cyrl	Khanty	19.276	19 796	102 697
und	Perm	Undetermined	1.071	57	5323
cop	Copt	Coptic	1.036	740	71 446
tkr	Cyrl	Tsakhur	0.928	53	5713
und	Khmr	Undetermined	0.886	3184	359 403
chu	Cyrl	Church Slavic	0.856	8544	998 622
mdy	Ethi	Male	0.596	35	5870
und	Shaw	Undetermined	0.557	39	6998
abq	Cyrl	Abaza	0.535	101	18 874
und	Kthi	Undetermined	0.531	512	96 358
kmr	Cyrl	Northern Kurdish	0.530	242	45 623
und	Mand	Undetermined	0.504	56	11 105
und	Sidd	Undetermined	0.494	360	72 840
und	Thai	Undetermined	0.475	1357	285 724
und	Nshu	Undetermined	0.430	1781	414 466
zsm	Arab	Standard Malay	0.397	31	7817
chr	Cher	Cherokee	0.394	189	47 988
und	Cham	Undetermined	0.392	106	27 053
yrk	Cyrl	Nenets	0.370	144	38 925
bbj	Latn	Ghomálá'	0.366	40	10 928
crk	Cans	Plains Cree	0.356	19	5340
dwr	Latn	Dawro	0.349	586	168 050
und	Copt	Undetermined	0.345	345	99 872
nnw	Latn	Southern Nuni	0.318	26	8182
nio	Cyrl	Nganasan	0.312	20	6410
und	Mong	Undetermined	0.301	768	255 042
und	Limb	Undetermined	0.296	148	50 010
und	Cher	Undetermined	0.295	345	116 813
itv	Latn	Itawit	0.274	45	16 448
und	Lyci	Undetermined	0.260	168	64 570
wsg	Telu	Adilabad Gondi	0.260	21	8076
und	Linb	Undetermined	0.258	638	247 520
sgh	Cyrl	Shughni	0.252	181	71 825
und	Grek	Undetermined	0.247	3583	1 452 750
und	Tang	Undetermined	0.244	3636	1 492 143
und	Hmnp	Undetermined	0.243	179	73 603
und	Sarb	Undetermined	0.224	19	8464

Table 13: The proportion of PUA *paragraphs* (shown in the fourth column) to all paragraphs for top 38 unique language-script pairs that have over 5,000 paragraphs, ranked in decreasing order.

fonts.[14] While the majority of glyph inventories for these writing systems map trivially to the assigned Cyrillic Unicode code points, there are 27 orthographies, shown in Table 15, that include some glyphs that involve PUA allocations. While the presence of PUA characters in well-resourced language orthographies (e.g., Russian) can be attributed to ParaType support for historic versions of these orthographies, for other languages, such as Avar, Even and Selkup, the relatively high number of PUA characters correlates with how established these orthographies are in terms of their Unicode support over time as well as the orthographic complexity of these languages.

A centralized mapping of ParaType glyph-to-PUA character assignments across all writing systems includes 109 unique assignments. These assignments are shown in Table 16 along with the frequency of appearance N in ParaType orthographies. As can be seen from the figure, the most frequent glyphs are CYRILLIC CAPITAL/SMALL LETTER A WITH MACRON and CYRILLIC CAPITAL/SMALL LETTER YA WITH MACRON, which appear in the orthographies of 10 languages. We hypothesize that such characters were added for compatibility with the existing non-PUA Unicode assignments for Latin script for which similar mappings (such as LATIN CAPITAL LETTER A WITH MACRON) form part of the Unicode standard.

[14] https://github.com/paratype/paratype.github.io/. Accessed: 25th December 2024.

D. Per-script LLM Instruction Metrics

This appendix provides a per-script breakdown of various LLM instruction following metrics discussed in Section 5 for the collection of paragraphs with PUA token-internal characters mined from DCAD-2000. This collection has paragraphs in 156 scripts as determined by GlotLID. Table 17 shows the relevant metrics for configuration with a longest (32K-token) limit imposed on input paragraphs. The entries are sorted by decreasing adherence A (fourth column). Table 18 shows the relevant metrics for the same collection obtained for a shortest (64-token) input sequence limit. The entries in this table are similarly ranked by decreas-

Family Name	Documentation	Scripts	# Fonts	PUA	
				μ	σ^2
Antykwa Toruńska	Nowacki (2005)	Cyrillic, Greek, Latin	16	401.2	1.1
Computer Modern Unicode	Multiple sources	Cyrillic, Greek, Latin	33	271.0	116.4
Church Slavonic	Andreev and Simmons (2020)	Cyrillic	18	180.3	337.5
Old Standard	Kryukov (2011) Alessi and Tsolomitis (2023)	Cyrillic, Greek, Latin	5	73.0	44.0

Table 14: Examples of publicly available popular modern typefaces with extensive Cyrillic support. Some font families support multiple scripts, while others focus on Cyrillic only. Each typeface is shown along with the number of fonts it contains as well as the mean and population variance for number of PUA code points for individual fonts. Note that for multi-script fonts not all PUA code points necessarily represent Cyrillic letters.

Orthography	Code(s)	Family	# PUAs
Avar	ava	Northeast Caucasian	32
Bashkir	bak	Turkic	6
Bulgarian	bul	Indo-European	4
Chechen	che	Northeast Caucasian	12
Chuvash	chv	Turkic	8
Enets	enf, env	Uralic	2
Even (Lamut)	eve	Tungusic	20
Evenki (Tungus)	evn	Tungusic	16
Ingush	inh	Northeast Caucasian	4
Itelmen	itl	Chukotko-Kamchatkan	2
Judeo-Tat	jdt	Indo-European	2
Karachay-Balkar	krc	Turkic	2
Khanty	kca	Uralic	18
...	...	...	...

Orthography	Code(s)	Family	# PUAs
Macedonian	mkd	Indo-European	2
Mansi	mns	Uralic	14
Eastern Mari	mhr	Uralic	2
Nanai	gld	Tungusic	16
Negidal	neg	Tungusic	18
Nenets (Yurak)	yrk	Uralic	2
Nganasan	nio	Uralic	4
Nivkh (Gilyak)	niv	Language isolate	2
Russian	rus	Indo-European	22
Kildin Saami	sjd	Uralic	14
Selkup	sel	Uralic	26
Serbian	srp	Indo-European	6
Ulch	ulc	Tungusic	16
Yakut	sah	Turkic	2

Table 15: PUA code points in ParaType Cyrillic orthographies of some languages. Each orthography in the table is shown along the corresponding ISO 639-3 language code, language family and the even number of PUA code points it uses to represent lower and upper-case letters. The higher-resource language orthographies in the Table (Bulgarian, Macedonian, Russian and Serbian) include historic characters.

ing adherence. Adherence is high for almost all scripts. For over half of the unique scripts, adherence is over 98% over all paragraphs. While restricting to shorter paragraphs tends to improve adherence slightly, *e.g.*, Latin script adherence moves to 98.9% from 98.2%, the gains are modest.

There is a reasonably short tail of scripts that do not display high adherence. If the adherence threshold for acceptable performance is set to 80%, there are 24 such scripts for the first condition (Table 17), and 19 scripts for the second condition (Table 18). For both sequence length limit conditions, among the scripts with over 500 observed PUA paragraphs, the worst performers include Yi (Yiii), Ancient Egyptian (Egyp), and Tangut (Tang), all three representing either logographic writing systems or, in the case of Yi, possibly a syllabary if the source sentences belong to modern Yi orthography. Ignoring the 500 paragraph limit, the worst-performing script in terms of LLM instruction adherence is a modern N'Ko (Nkoo) alphabet originally developed for the Bambara language (Wyrod, 2008).

E. LLM Instruction Following with Number of PUA Characters Excluded from the Prompt

This appendix provides overall and per-script breakdown of the LLM instruction following metrics discussed in Section 5 when the number of PUA characters is not explicitly given in the prompt. Table 19 gives metrics for LLM instruction following with this older, exploratory prompt.

Without explicitly giving the number of PUA characters in the prompt, this task is not trivial. In fact, the performance heavily depends on the input sequence length. Overall instruction adherence is quite low with the best adherence of 72% obtained for the shortest input sequences with a limit of 64 tokens. For all the conditions apart from the longest input sequence limit, the model under- and over-generates the PUA class tags roughly equally according to the E_U and E_O metrics. However, according to R_C, when the model over-generates, it does so heavily and this holds for all the conditions and especially the first one, where for the 32K sequence limit the model hallucinates nearly 32 times more PUA characters than the number

Code	N	Description
U+F43A	1	CAPITAL LETTER STRAIGHT U WITH MACRON
U+F43B	1	SMALL LETTER STRAIGHT U WITH MACRON
U+F460	2	CAPITAL LETTER A WITH ACUTE
U+F461	2	CAPITAL LETTER IE WITH ACUTE
U+F462	2	CAPITAL LETTER I WITH ACUTE
U+F463	2	CAPITAL LETTER O WITH ACUTE
U+F464	2	CAPITAL LETTER U WITH ACUTE
U+F465	2	CAPITAL LETTER YERU WITH ACUTE
U+F466	2	CAPITAL LETTER E WITH ACUTE
U+F467	2	CAPITAL LETTER YU WITH ACUTE
U+F468	2	CAPITAL LETTER YA WITH ACUTE
U+F469	2	CAPITAL LETTER IE WITH DIAERESIS AND ACUTE
U+F46A	2	SMALL LETTER A WITH ACUTE
U+F46B	2	SMALL LETTER IE WITH ACUTE
U+F46C	2	SMALL LETTER I WITH ACUTE
U+F46D	2	SMALL LETTER O WITH ACUTE
U+F46E	2	SMALL LETTER U WITH ACUTE
U+F46F	2	SMALL LETTER YERU WITH ACUTE
U+F470	2	SMALL LETTER E WITH ACUTE
U+F471	2	SMALL LETTER YU WITH ACUTE
U+F472	2	SMALL LETTER YA WITH ACUTE
U+F473	2	SMALL LETTER IE WITH DIAERESIS AND ACUTE
U+F498	1	CAPITAL LETTER YU WITH DIAERESIS
U+F499	1	SMALL LETTER YU WITH DIAERESIS
U+F49A	3	CAPITAL LETTER BARRED O WITH MACRON
U+F49B	3	SMALL LETTER BARRED O WITH MACRON
U+F49C	1	CAPITAL LETTER SCHWA WITH MACRON
U+F49D	1	SMALL LETTER SCHWA WITH MACRON
U+F49E	1	CAPITAL LETTER SHHA WITH BAR
U+F49F	1	SMALL LETTER SHHA WITH BAR
U+F4C6	1	CAPITAL LETTER CHE WITH HOOK
U+F4C7	1	SMALL LETTER CHE WITH HOOK
U+F4CC	1	CAPITAL LETTER U WITH ACUTE
U+F4CD	1	SMALL LETTER U WITH ACUTE
U+F4D2	2	CAPITAL LETTER O WITH GRAVE
U+F4D3	2	SMALL LETTER O WITH GRAVE
U+F4D4	1	CAPITAL LETTER ER WITH CARON
U+F4D5	1	SMALL LETTER ER WITH CARON
U+F4D6	1	CAPITAL LETTER E WITH DOT ABOVE
U+F4D7	1	SMALL LETTER E WITH DOT ABOVE
U+F4D8	2	CAPITAL LETTER YA WITH DIAERESIS
U+F4D9	2	SMALL LETTER YA WITH DIAERESIS
U+F50A	2	CAPITAL LETTER ZE WITH TAIL
U+F50B	2	SMALL LETTER ZE WITH TAIL
U+F50C	3	CAPITAL LETTER ES WITH CEDILLA
U+F50D	3	SMALL LETTER ES WITH CEDILLA
U+F50E	10	CAPITAL LETTER A WITH MACRON
U+F50F	10	SMALL LETTER A WITH MACRON
U+F510	9	CAPITAL LETTER IE WITH MACRON
U+F511	9	SMALL LETTER IE WITH MACRON
U+F512	7	CAPITAL LETTER IE WITH DIAERESIS AND MACRON
U+F513	7	SMALL LETTER IE WITH DIAERESIS AND MACRON
U+F514	1	CAPITAL LETTER IE WITH DIAERESIS AND BREVE
U+F515	1	SMALL LETTER IE WITH DIAERESIS AND BREVE
...	...	...

Code	N	Description
...	...	...
U+F516	1	CAPITAL LETTER ZE WITH CARON
U+F517	1	SMALL LETTER ZE WITH CARON
U+F518	9	CAPITAL LETTER O WITH MACRON
U+F519	9	SMALL LETTER O WITH MACRON
U+F51A	3	CAPITAL LETTER O WITH BREVE
U+F51B	3	SMALL LETTER O WITH BREVE
U+F51C	1	CAPITAL LETTER BARRED O WITH BREVE
U+F51D	1	SMALL LETTER BARRED O WITH BREVE
U+F51E	1	CAPITAL LETTER BARRED O WITH DIAERESIS AND BREVE
U+F51F	1	SMALL LETTER BARRED O WITH DIAERESIS AND BREVE
U+F520	7	CAPITAL LETTER YERU WITH MACRON
U+F521	7	SMALL LETTER YERU WITH MACRON
U+F522	9	CAPITAL LETTER E WITH MACRON
U+F523	9	SMALL LETTER E WITH MACRON
U+F524	1	CAPITAL LETTER E WITH BREVE
U+F525	1	SMALL LETTER E WITH BREVE
U+F526	1	CAPITAL LETTER UKRAINIAN IE WITH DIAERESIS
U+F527	1	SMALL LETTER UKRAINIAN IE WITH DIAERESIS
U+F528	10	CAPITAL LETTER YA WITH MACRON
U+F529	10	SMALL LETTER YA WITH MACRON
U+F52A	1	CAPITAL LETTER YA WITH BREVE
U+F52B	1	SMALL LETTER YA WITH BREVE
U+F52C	9	CAPITAL LETTER YU WITH MACRON
U+F52D	9	SMALL LETTER YU WITH MACRON
U+F52E	1	CAPITAL LETTER YU WITH BREVE
U+F52F	1	SMALL LETTER YU WITH BREVE
U+F532	1	CAPITAL LETTER SHORT I WITH HOOK
U+F533	1	SMALL LETTER SHORT I WITH HOOK
U+F536	2	CAPITAL LETTER YERU WITH BREVE
U+F537	2	SMALL LETTER YERU WITH BREVE
U+F538	1	CAPITAL LETTER REVERSED ZE WITH DIAERESIS
U+F539	1	SMALL LETTER REVERSED ZE WITH DIAERESIS
U+F53C	1	CAPITAL LETTER GHE WITH STROKE
U+F53D	1	SMALL LETTER GHE WITH STROKE
U+F53E	1	CAPITAL LETTER ES WITH TAIL
U+F53F	1	SMALL LETTER ES WITH TAIL
U+F830	1	CAPITAL LETTER A WITH CIRCUMFLEX
U+F831	1	SMALL LETTER A WITH CIRCUMFLEX
U+F833	1	CAPITAL LETTER IE WITH CIRCUMFLEX
U+F834	1	SMALL LETTER IE WITH CIRCUMFLEX
U+F839	1	CAPITAL LETTER O WITH CIRCUMFLEX
U+F83A	1	SMALL LETTER O WITH CIRCUMFLEX
U+F86F	1	CAPITAL LETTER KA WITH MACRON
U+F870	1	SMALL LETTER KA WITH MACRON
U+F872	1	CAPITAL LETTER EL WITH MACRO
U+F873	1	SMALL LETTER EL WITH MACRON
U+F875	1	CAPITAL LETTER ES WITH MACRO
U+F876	1	SMALL LETTER ES WITH MACRON
U+F878	1	CAPITAL LETTER HA WITH MACRO
U+F879	1	SMALL LETTER HA WITH MACRON
U+F87B	1	CAPITAL LETTER TSE WITH MACRO
U+F87C	1	SMALL LETTER TSE WITH MACRON
U+F87E	1	CAPITAL LETTER CHE WITH MACRO
U+F87F	1	SMALL LETTER CHE WITH MACRON

Table 16: Inventory of PUA assignments from ParaType type foundry along with the number of Cyrillic writing systems N, where the particular code point is used. The prefix CYRILLIC has been omitted from the original character names.

of characters found in the actual data. The overall number of hallucinations decreases as one imposes shorter limits on the input sequence length. According to the R_L metric, in the first condition the OTHER class heavily dominates the outputs with LETTER classes comprising only 13.4% of the overall predictions. For rest of the conditions this measure is reasonably constant in the 42–45% ballpark.

Table 20 shows the relevant metrics for configuration with a longest (32K-token) limit imposed on input paragraphs. The entries are sorted by decreasing adherence (fourth column). Table 21 shows the relevant metrics for the same collection obtained for a shortest (64-token) input sequence limit. The entries in this table are similarly ranked by decreasing adherence.

As can be seen from these tables, the performance of adherence and other metrics is highly script- and paragraph length-specific. Among the best consistently performing scripts there are some scripts with very low counts of paragraphs and tokens, such as Elymaic (Elym), Palmyrene (Palm) and Marchen (Marc). For these scripts the metrics do not provide any useful evidence. Among the scripts with sufficient counts, Odiya script (Orya) consistently performs best among all the scripts for both configurations of input para-

Code	N_S	N_T	A	E_U	E_O	R_C	R_L
Adlm	9	60	100.0	0.0	0.0	1.0	6.7
Phnx	3	81	100.0	0.0	0.0	1.0	75.3
Phli	3	98	100.0	0.0	0.0	1.0	19.4
Perm	2	10	100.0	0.0	0.0	1.0	20.0
Palm	1	2	100.0	0.0	0.0	1.0	100.0
Ogam	5	198	100.0	0.0	0.0	1.0	53.0
Nbat	13	318	100.0	0.0	0.0	1.0	96.5
Nand	2	22	100.0	0.0	0.0	1.0	18.2
Nagm	2	32	100.0	0.0	0.0	1.0	90.6
Mult	7	212	100.0	0.0	0.0	1.0	24.1
Modi	25	537	100.0	0.0	0.0	1.0	64.2
Mero	3	76	100.0	0.0	0.0	1.0	21.1
Merc	12	296	100.0	0.0	0.0	1.0	59.5
Medf	403	2892	100.0	0.0	0.0	1.0	1.2
Aghb	2	46	100.0	0.0	0.0	1.0	100.0
Mand	1	22	100.0	0.0	0.0	1.0	100.0
Maka	3	85	100.0	0.0	0.0	1.0	67.1
Kthi	2	47	100.0	0.0	0.0	1.0	85.1
Kali	12	160	100.0	0.0	0.0	1.0	31.9
Ital	3	109	100.0	0.0	0.0	1.0	0.0
Hmng	10	309	100.0	0.0	0.0	1.0	97.4
Phlp	1	62	100.0	0.0	0.0	1.0	0.0
Prti	1	40	100.0	0.0	0.0	1.0	0.0
Hano	1	44	100.0	0.0	0.0	1.0	0.0
Rjng	1	40	100.0	0.0	0.0	1.0	0.0
Yezi	6	105	100.0	0.0	0.0	1.0	32.4
Wcho	2	22	100.0	0.0	0.0	1.0	0.0
Ugar	1	6	100.0	0.0	0.0	1.0	0.0
Toto	3	60	100.0	0.0	0.0	1.0	83.3
Tnsa	8	191	100.0	0.0	0.0	1.0	88.0
Tirh	3	130	100.0	0.0	0.0	1.0	86.2
Thaa	8	310	100.0	0.0	0.0	1.0	16.8
Tglg	2	29	100.0	0.0	0.0	1.0	24.1
Tfng	7	135	100.0	0.0	0.0	1.0	31.9
Tavt	8	205	100.0	0.0	0.0	1.0	47.3
Takr	4	265	100.0	0.0	0.0	1.0	9.8
Tagb	5	79	100.0	0.0	0.0	1.0	72.2
Soyo	4	77	100.0	0.0	0.0	1.0	0.0
Sogo	3	71	100.0	0.0	0.0	1.0	18.3
Sogd	3	39	100.0	0.0	0.0	1.0	71.8
Sind	1	3	100.0	0.0	0.0	1.0	0.0
Shaw	3	29	100.0	0.0	0.0	1.0	69.0
Sarb	1	54	100.0	0.0	0.0	1.0	0.0
Samr	3	122	100.0	0.0	0.0	1.0	65.6
Hatr	2	24	100.0	0.0	0.0	1.0	0.0
Marc	7	30	100.0	0.0	0.0	1.0	0.0
Buhd	1	7	100.0	0.0	0.0	1.0	0.0
Dsrt	13	295	100.0	0.0	0.0	1.0	50.8
Gong	4	92	100.0	0.0	0.0	1.0	35.9
Bhks	7	269	100.0	0.0	0.0	1.0	43.1
Bali	3	37	100.0	0.0	0.0	1.0	40.5
Elym	3	9	100.0	0.0	0.0	1.0	55.6
Dogr	2	8	100.0	0.0	0.0	1.0	0.0
Diak	9	157	100.0	0.0	0.0	1.0	69.4
Elba	3	13	100.0	0.0	0.0	1.0	92.3
Armi	6	182	100.0	0.0	0.0	1.0	52.2
Ahom	5	72	100.0	0.0	0.0	1.0	98.6
Batk	3	24	100.0	0.0	0.0	1.0	0.0
Gran	2	17	100.0	0.0	0.0	1.0	0.0
Orya	2904	3366	100.0	0.0	0.0	1.0	88.4
Khmr	6568	26 009	99.6	0.2	0.2	1.0	86.8
Armn	396	1948	99.5	0.3	0.3	1.0	30.0
Guru	1901	11 160	99.5	0.3	0.2	1.0	25.5
Arab	10 674	52 799	99.5	0.2	0.3	2.1	30.1
Gujr	903	6560	99.4	0.3	0.2	1.9	19.6
Deva	5404	24 033	99.4	0.3	0.3	2.0	77.2
Geor	870	4887	99.1	0.7	0.2	1.0	12.5
Mlym	719	4186	99.0	0.6	0.4	1.0	21.5
Mymr	1664	11 196	98.9	0.4	0.7	1.5	30.0
Grek	7616	44 845	98.8	0.8	0.4	1.7	40.1
Telu	1485	4240	98.8	0.8	0.4	2.9	6.9
Beng	7830	11 531	98.8	1.0	0.3	1.2	30.2
Laoo	1993	23 184	98.7	0.8	0.6	1.6	94.5
Hang	30 204	186 766	98.7	0.9	0.4	1.4	72.6
Taml	1120	4933	98.5	1.0	0.5	3.6	92.4
Latn	760 493	4 840 954	98.2	1.0	0.8	1.8	29.2
Sinh	1296	22 935	98.1	1.5	0.3	1.0	85.6
Cyrl	100 166	910 319	98.1	1.5	0.4	1.3	76.5
...	...	...	...	...	...	...	...

Code	N_S	N_T	A	E_U	E_O	R_C	R_L
...	...	...	...	...	...	...	...
Plrd	47	1240	97.9	0.0	2.1	1.0	75.8
Knda	835	10 982	97.8	1.1	1.1	1.7	42.5
Jpan	359	5178	97.2	1.4	1.4	1.6	13.9
Thai	25 533	342 758	96.8	2.3	0.8	1.6	83.8
Talu	31	815	96.8	0.0	3.2	1.0	88.8
Newa	26	838	96.2	3.8	0.0	1.0	84.7
Nshu	52	2280	96.2	3.8	0.0	0.9	41.8
Osge	25	496	96.0	0.0	4.0	1.0	49.1
Sidd	50	1185	96.0	4.0	0.0	1.0	92.5
Hebr	9410	125 431	95.8	3.8	0.4	1.0	12.2
Xpeo	20	713	95.0	0.0	5.0	1.0	40.6
Sund	18	960	94.4	0.0	5.6	13.0	2.3
Brah	35	1004	94.3	0.0	5.7	16.9	14.0
Copt	196	4919	93.9	4.1	2.0	1.0	53.6
Mtei	16	649	93.8	0.0	6.2	1.0	63.2
Lana	104	3766	93.3	5.8	1.0	0.9	92.1
Bamu	257	16 334	93.0	3.9	3.1	0.8	52.1
Cprt	14	367	92.9	0.0	7.1	1.1	56.6
Glag	14	700	92.9	7.1	0.0	0.4	25.7
Dupl	41	2476	92.7	4.9	2.4	7.2	95.0
Sgnw	48	2154	91.7	2.1	6.2	1.7	79.3
Java	12	915	91.7	0.0	8.3	1.6	2.0
Hani	1784	55 538	90.9	4.6	4.5	2.3	20.1
Avst	11	209	90.9	0.0	9.1	1.0	40.3
Bopo	31	2333	90.3	3.2	6.5	1.4	23.8
Cpmn	40	1879	90.0	2.5	7.5	1.1	84.8
Vith	10	325	90.0	10.0	0.0	1.0	37.5
Cakm	10	342	90.0	0.0	10.0	1.0	43.2
Tibt	95	4306	89.5	5.3	5.3	1.9	62.0
Cans	111	6090	89.2	4.5	6.3	2.1	29.7
Hung	18	442	88.9	11.1	0.0	1.0	63.6
Mong	96	4290	88.5	6.2	5.2	2.0	88.5
Brai	43	2924	88.4	4.7	7.0	2.8	67.5
Khar	8	406	87.5	12.5	0.0	1.0	18.5
Ethi	978	20 769	86.7	9.0	4.3	1.2	86.8
Syrc	66	10 461	86.4	4.5	9.1	1.4	24.5
Orkh	7	538	85.7	0.0	14.3	1.1	79.1
Lyci	14	408	85.7	14.3	0.0	1.0	85.6
Mend	27	1304	85.2	11.1	3.7	1.0	42.2
Kits	70	4204	84.3	4.3	11.4	1.6	73.6
Runr	12	528	83.3	0.0	16.7	1.0	47.1
Lepc	6	261	83.3	16.7	0.0	1.0	77.3
Mani	6	828	83.3	0.0	16.7	7.4	1.0
Pauc	6	141	83.3	0.0	16.7	1.1	53.7
Cham	11	529	81.8	18.2	0.0	1.0	40.3
Linb	65	4158	81.5	9.2	9.2	5.9	13.4
Phag	37	812	81.1	13.5	5.4	1.0	93.4
Saur	131	9763	80.2	11.5	8.4	1.4	30.3
Zanb	5	383	80.0	20.0	0.0	1.0	35.9
Osma	5	253	80.0	0.0	20.0	1.0	73.6
Gonm	5	121	80.0	20.0	0.0	1.0	71.7
Shrd	24	1308	79.2	8.3	12.5	1.0	31.5
Hira	33	4035	78.8	12.1	9.1	1.9	7.3
Kana	37	2421	78.4	10.8	10.8	0.9	39.8
Lina	75	5787	77.3	14.7	8.0	1.7	34.8
Limb	38	11 146	76.3	10.5	13.2	0.9	10.2
Cher	29	3313	75.9	13.8	10.3	2.3	36.1
Mahj	4	115	75.0	0.0	25.0	1.3	43.1
Rohg	8	522	75.0	25.0	0.0	0.7	31.0
Olck	7	295	71.4	14.3	14.3	1.0	99.7
Cari	17	1359	70.6	11.8	17.6	8.2	4.3
Kawi	10	495	70.0	20.0	10.0	5.0	7.3
Wara	64	6591	68.8	20.3	10.9	3.5	71.7
Xsux	480	71 375	67.7	12.9	19.4	2.9	51.2
Zzzz	54	6707	64.8	5.6	29.6	12.2	3.3
Vaii	53	12 064	64.2	15.1	20.8	2.0	71.4
Egyp	829	182 167	63.6	11.8	24.6	2.9	48.6
Hluw	348	52 052	62.1	9.8	28.2	3.3	40.3
Tang	1588	322 750	60.0	14.4	25.6	3.4	49.3
Hmnp	12	2246	58.3	16.7	25.0	1.9	1.1
Mroo	9	1258	55.6	11.1	33.3	3.1	97.2
Yiii	1790	502 871	52.0	17.2	30.9	3.0	52.0
Lisu	10	714	50.0	30.0	20.0	3.2	2.3
Ougr	2	49	50.0	50.0	0.0	0.8	0.0
Sora	2	167	50.0	50.0	0.0	1.0	31.2
Bass	12	1927	41.7	16.7	41.7	3.4	0.0
Nkoo	45	867	28.9	71.1	0.0	0.5	66.7
All	991 309	8 060 123	98.0	1.2	0.9	1.9	40.7

Table 17: LLM instruction following metrics sorted by adherence: 32K-token input sequence limit.

Code	N_S	N_T	A	E_U	E_O	R_C	R_L
Adlm	9	60	100.0	0.0	0.0	1.0	6.7
Mero	3	76	100.0	0.0	0.0	1.0	21.1
Phli	3	98	100.0	0.0	0.0	1.0	19.4
Perm	2	10	100.0	0.0	0.0	1.0	20.0
Palm	1	2	100.0	0.0	0.0	1.0	100.0
Ogam	5	198	100.0	0.0	0.0	1.0	53.0
Nbat	13	318	100.0	0.0	0.0	1.0	96.5
Nand	2	22	100.0	0.0	0.0	1.0	18.2
Nagm	2	32	100.0	0.0	0.0	1.0	90.6
Mult	7	212	100.0	0.0	0.0	1.0	24.1
Modi	25	537	100.0	0.0	0.0	1.0	64.2
Merc	12	296	100.0	0.0	0.0	1.0	59.5
Hano	1	44	100.0	0.0	0.0	1.0	0.0
Medf	403	2892	100.0	0.0	0.0	1.0	1.2
Aghb	2	46	100.0	0.0	0.0	1.0	100.0
Mani	4	153	100.0	0.0	0.0	1.0	37.9
Mand	1	22	100.0	0.0	0.0	1.0	100.0
Maka	3	85	100.0	0.0	0.0	1.0	67.1
Kthi	2	47	100.0	0.0	0.0	1.0	85.1
Kali	12	160	100.0	0.0	0.0	1.0	31.9
Ital	2	55	100.0	0.0	0.0	1.0	0.0
Hmng	10	309	100.0	0.0	0.0	1.0	97.4
Phlp	1	62	100.0	0.0	0.0	1.0	0.0
Phnx	3	81	100.0	0.0	0.0	1.0	75.3
Prti	1	40	100.0	0.0	0.0	1.0	0.0
Rjng	1	40	100.0	0.0	0.0	1.0	0.0
Yezi	6	105	100.0	0.0	0.0	1.0	32.4
Wcho	2	22	100.0	0.0	0.0	1.0	0.0
Ugar	1	6	100.0	0.0	0.0	1.0	0.0
Toto	3	60	100.0	0.0	0.0	1.0	83.3
Tnsa	8	191	100.0	0.0	0.0	1.0	88.0
Tirh	3	130	100.0	0.0	0.0	1.0	86.2
Thaa	8	310	100.0	0.0	0.0	1.0	16.8
Tglg	2	29	100.0	0.0	0.0	1.0	24.1
Tfng	6	80	100.0	0.0	0.0	1.0	53.8
Tavt	8	205	100.0	0.0	0.0	1.0	47.3
Takr	4	265	100.0	0.0	0.0	1.0	9.8
Tagb	5	79	100.0	0.0	0.0	1.0	72.2
Soyo	4	77	100.0	0.0	0.0	1.0	0.0
Sogo	3	71	100.0	0.0	0.0	1.0	18.3
Sogd	3	39	100.0	0.0	0.0	1.0	71.8
Sind	1	3	100.0	0.0	0.0	1.0	0.0
Shaw	3	29	100.0	0.0	0.0	1.0	69.0
Sarb	1	54	100.0	0.0	0.0	1.0	0.0
Samr	3	122	100.0	0.0	0.0	1.0	65.6
Hatr	2	24	100.0	0.0	0.0	1.0	0.0
Marc	7	30	100.0	0.0	0.0	1.0	0.0
Batk	3	24	100.0	0.0	0.0	1.0	0.0
Dsrt	13	295	100.0	0.0	0.0	1.0	50.8
Dogr	2	8	100.0	0.0	0.0	1.0	0.0
Ahom	5	72	100.0	0.0	0.0	1.0	98.6
Bali	3	37	100.0	0.0	0.0	1.0	40.5
Buhd	1	7	100.0	0.0	0.0	1.0	0.0
Gong	4	92	100.0	0.0	0.0	1.0	35.9
Elba	3	13	100.0	0.0	0.0	1.0	92.3
Bopo	28	1072	100.0	0.0	0.0	1.0	55.0
Bhks	7	269	100.0	0.0	0.0	1.0	43.1
Diak	9	157	100.0	0.0	0.0	1.0	69.4
Elym	3	9	100.0	0.0	0.0	1.0	55.6
Gran	2	17	100.0	0.0	0.0	1.0	0.0
Armi	6	182	100.0	0.0	0.0	1.0	52.2
Orya	2006	2327	100.0	0.0	0.0	1.0	84.5
Beng	377	843	99.7	0.3	0.0	1.0	18.8
Khmr	6436	22 846	99.7	0.1	0.2	1.1	86.1
Arab	7220	34 923	99.6	0.1	0.2	1.0	29.8
Geor	687	3492	99.6	0.4	0.0	1.0	8.8
Hang	14 278	65 885	99.5	0.3	0.1	1.0	68.7
Deva	3411	15 501	99.5	0.3	0.2	1.1	59.4
Armn	366	1779	99.5	0.3	0.3	1.0	30.0
Gujr	842	6173	99.4	0.4	0.2	2.0	17.5
Guru	1075	8364	99.3	0.4	0.3	1.0	11.6
Grek	7209	34 234	99.3	0.5	0.2	1.2	18.3
Mymr	1570	8673	99.3	0.3	0.4	1.5	32.8
Mlym	707	4004	99.2	0.4	0.4	1.0	22.4
Laoo	1919	20 651	99.1	0.6	0.4	1.7	95.1
Latn	657 219	2 447 839	98.9	0.5	0.6	1.5	20.2
Telu	344	1578	98.8	0.9	0.3	0.9	27.8
Cyrl	80 795	596 194	98.6	1.2	0.2	1.0	81.5
...	...	...	...	...	...	...	...

Code	N_S	N_T	A	E_U	E_O	R_C	R_L
...	...	...	...	...	...	...	...
Sinh	1215	22 198	98.3	1.6	0.2	1.0	85.5
Hebr	742	9846	98.2	0.9	0.8	1.3	84.8
Thai	18 020	146 225	98.2	1.5	0.3	1.1	81.3
Taml	260	2607	98.1	1.5	0.4	0.9	76.4
Knda	748	10 261	98.0	1.1	0.9	1.7	41.9
Sidd	47	1157	97.9	2.1	0.0	1.0	92.4
Plrd	46	1158	97.8	0.0	2.2	1.0	81.1
Lana	84	1685	97.6	1.2	1.2	1.1	89.5
Ethi	432	5612	97.2	0.9	1.9	1.2	66.0
Jpan	355	4988	97.2	1.4	1.4	1.6	12.3
Talu	31	815	96.8	0.0	3.2	1.0	88.8
Newa	26	838	96.2	3.8	0.0	1.0	84.7
Nshu	50	2173	96.0	4.0	0.0	0.9	44.2
Osge	25	496	96.0	0.0	4.0	1.0	49.1
Dupl	40	2044	95.0	5.0	0.0	0.9	53.0
Xpeo	20	713	95.0	0.0	5.0	1.0	40.6
Bamu	253	7939	94.5	2.8	2.8	1.2	53.2
Sund	18	960	94.4	0.0	5.6	13.0	2.3
Copt	195	4699	94.4	4.1	1.5	1.0	50.8
Brah	35	1004	94.3	0.0	5.7	16.9	14.0
Mtei	16	649	93.8	0.0	6.2	1.0	63.2
Cprt	14	367	92.9	0.0	7.1	1.1	56.6
Sgnw	48	2154	91.7	2.1	6.2	1.7	79.3
Java	12	915	91.7	0.0	8.3	1.6	2.0
Tibt	92	4014	91.3	3.3	5.4	2.1	61.7
Avst	11	209	90.9	0.0	9.1	1.0	40.3
Hani	1704	53 901	90.9	4.5	4.6	2.3	20.1
Cans	109	4788	90.8	4.6	4.6	1.9	20.1
Brai	42	1813	90.5	4.8	4.8	0.9	34.1
Mong	93	3060	90.3	6.5	3.2	1.4	78.9
Glag	10	644	90.0	10.0	0.0	0.3	7.8
Cpmn	40	1879	90.0	2.5	7.5	1.1	84.8
Vith	10	325	90.0	10.0	0.0	1.0	37.5
Cakm	10	342	90.0	0.0	10.0	1.0	43.2
Hira	29	1677	89.7	6.9	3.4	1.8	7.4
Hung	18	442	88.9	11.1	0.0	1.0	63.6
Syrc	65	3663	87.7	4.6	7.7	1.3	74.6
Zzzz	32	1465	87.5	3.1	9.4	11.9	3.9
Khar	8	406	87.5	12.5	0.0	1.0	18.5
Kits	68	3014	86.8	4.4	8.8	1.0	40.6
Lyci	14	408	85.7	14.3	0.0	1.0	85.6
Cher	26	1321	84.6	11.5	3.8	2.2	12.6
Mend	25	1215	84.0	12.0	4.0	1.0	45.3
Pauc	6	141	83.3	0.0	16.7	1.1	53.7
Runr	12	528	83.3	0.0	16.7	1.0	47.1
Orkh	6	537	83.3	0.0	16.7	1.1	79.0
Olck	6	90	83.3	16.7	0.0	0.8	98.7
Lepc	6	261	83.3	16.7	0.0	1.0	77.3
Linb	64	3920	82.8	9.4	7.8	6.2	13.5
Shrd	23	1179	82.6	4.3	13.0	1.0	34.7
Saur	126	7482	82.5	10.3	7.1	1.3	39.7
Phag	37	812	81.1	13.5	5.4	1.0	93.4
Kana	36	2217	80.6	11.1	8.3	0.8	35.0
Lina	72	4136	80.6	13.9	5.6	1.1	75.4
Zanb	5	383	80.0	20.0	0.0	1.0	35.9
Gonm	5	121	80.0	20.0	0.0	1.0	71.7
Osma	5	253	80.0	0.0	20.0	1.0	73.6
Cham	10	413	80.0	20.0	0.0	1.0	52.2
Limb	37	8105	78.4	8.1	13.5	1.0	11.9
Egyp	700	57 503	75.3	8.1	16.6	3.3	63.0
Mahj	4	115	75.0	0.0	25.0	1.3	43.1
Rohg	8	522	75.0	25.0	0.0	0.7	31.0
Wara	59	3278	74.6	18.6	6.8	0.8	59.8
Xsux	437	33 886	74.1	10.8	15.1	3.0	43.2
Kawi	7	165	71.4	14.3	14.3	12.9	1.9
Vaii	48	6385	70.8	12.5	16.7	2.6	67.8
Cari	17	1359	70.6	11.8	17.6	8.2	4.3
Tang	1352	115 930	70.4	11.2	18.4	3.7	51.2
Hluw	308	31 122	70.1	9.4	20.5	2.2	48.0
Hmnp	10	523	70.0	20.0	10.0	0.7	13.3
Yiii	1392	134 304	66.7	12.1	21.1	3.3	59.3
Mroo	8	630	62.5	12.5	25.0	3.8	95.4
Lisu	9	310	55.6	33.3	11.1	0.9	18.2
Sora	2	167	50.0	50.0	0.0	1.0	31.2
Ougr	2	49	50.0	50.0	0.0	0.8	0.0
Bass	12	1927	41.7	16.7	41.7	3.4	0.0
Nkoo	45	867	28.9	71.1	0.0	0.5	66.7
All	817 177	4 021 097	98.7	0.7	0.6	1.6	36.6

Table 18: LLM instruction following metrics sorted by adherence: 64-token input sequence limit.

1. Role

You are a specialist in Unicode characters with the special expertise on private-use area (PUA) characters. The PUA character is a character in one of three Unicode ranges: `U+E000`–`U+F8FF`, `U+F0000`–`U+FFFFD`, `U+100000`–`U+10FFFD`.

2. Instructions

- The sentence below contains one or more PUA characters.
- Determine the language of the sentence. Ignore *mojibake*.
- Find each PUA character and annotate it with one of the following 2 labels:
  1. **"`LETTER`"**: The character is a valid letter in a word that belongs to the language of the sentence.
  2. **"`OTHER`"**: The character belongs to numbers, punctuation, icons, list delimiters, emoji, symbols.

Your output should consist of only a list of labels, one for each PUA character in the sentence, without additional text or explanation.

3. Input

`SENTENCE: {{text}}`

4. Few-shot examples

INPUT: ⊠Убалъэм ихуар
ANALYSIS: One PUA character. This character is a list delimiter because "Убалъэм" is a valid word.
OUTPUT: `[OTHER]`

INPUT: рӯпитэгыт үҙәге⊠дәүләт
ANALYSIS: Two PUA characters. The first PUA character is a valid substring in a word "рӯпитэгыт". The second PUA character is punctuation in "үҙәге» дәүләт".
OUTPUT: `[LETTER, OTHER]`

INPUT: loo na spolupráci s⊠Michalem\nSuchánkem a⊠Richardem
ANALYSIS: Two PUA characters. The first and second PUA characters are punctuation in "s Michalem" and "a Richardem".
OUTPUT: `[OTHER, OTHER]`

INPUT: ɑιU⊠uѡʊ
ANALYSIS: One PUA character. The full word without the garbled character is "ɑιU̇uѡʊ".
OUTPUT: `[LETTER]`

INPUT: }rdf⊠lh⊠⊠hgf fj}sgrk⊠
ANALYSIS: Four PUA characters. This is not natural language and is probably a *mojibake*.
OUTPUT: `[OTHER, OTHER, OTHER, OTHER]`

INPUT: Тыныӈ омакем, с⊠рни омакем
ANALYSIS: The PUA character in "с⊠рни" is a valid letter "cyrillic o with macron".
OUTPUT: `[LETTER]`

Figure 2: Exploratory LLM prompt for classifying the PUA characters into two basic types, without explicitly specifying the number of PUA characters.

$\|S\|$	N_S	N_T	A	E_U	E_O	R_C	R_L
32K	991 441	8 063 322	64.8	16.0	19.2	31.7	13.4
512	954 470	6 537 913	67.1	16.2	16.7	10.7	43.2
256	916 663	5 526 594	68.9	15.2	15.9	8.8	45.4
128	875 300	4 785 262	70.4	14.6	14.9	6.9	45.7
64	817 177	4 021 097	72.0	14.0	14.0	4.6	42.1

Table 19: Exploratory prompt and LLM instruction following: input length limit ($|S|$), number of paragraphs N_S satisfying the limit, total number of PUA characters N_T observed, and the values of five corresponding metrics. Number of PUA characters omitted from LLM prompt.

graph length limit achieving adherence of 91% and 97%, respectively. On the other hand, the model performs very poorly on long paragraphs in Bengali (Beng) script (32K-token limit) adhering to instructions only 5.6% of the time. In this particular scenario, according to the R_C metric (the ratio of total number of generated PUA character classes to the total number of observed character classes), the model exhibits the most extreme over-generation among all the scripts (R_C=3422.4). The Bengali script adherence significantly improves for the shortest token condition (64-token limit) to 65%, with over-generation reduced drastically (R_C=1.0). Analyzing such discrepancies in LLM performance across scripts and conditions requires further investigation.

Code	N_S	N_T	A	E_{UI}	E_O	R_C	R_L	Code	N_S	N_T	A	E_{UI}	E_O	R_C	R_L
Elym	3	9	100.0	0.0	0.0	1.0	55.6	...	...	...	...	...	...	...	...
Palm	1	2	100.0	0.0	0.0	1.0	100.0	Bass	12	1927	8.3	33.3	58.3	6.1	0.0
Orya	2904	3366	90.7	0.6	8.7	1.2	78.4	Hmnp	12	2246	8.3	58.3	33.3	6.5	0.4
Marc	7	30	85.7	14.3	0.0	0.9	57.1	Runr	12	528	8.3	75.0	16.7	5.0	5.2
Medf	403	2892	79.7	1.0	19.4	1.1	1.7	Lina	75	5787	8.0	65.3	26.7	9.3	32.8
Grek	7616	44 845	75.9	13.4	10.7	11.9	29.4	Nshu	52	2280	7.7	67.3	25.0	10.3	81.6
Latn	760 539	4 844 035	73.6	12.6	13.7	13.6	25.6	Nkoo	45	867	6.7	68.9	24.4	11.1	2.8
Bali	3	37	66.7	33.3	0.0	0.9	62.5	Plrd	47	1240	6.4	66.0	27.7	14.1	98.2
Armn	396	1948	64.1	23.7	12.1	9.3	93.0	Hira	33	4035	6.1	54.5	39.4	8.8	1.5
Arab	10 674	52 799	57.7	14.7	27.6	14.5	23.8	Beng	7830	11 531	5.6	1.3	93.1	3422.4	0.7
Deva	5404	24 033	57.5	15.7	26.8	10.5	57.0	Sund	18	960	5.6	72.2	22.2	0.6	57.9
Mymr	1664	11 196	55.0	19.5	25.5	10.3	18.7	Kana	37	2421	5.4	59.5	35.1	10.5	11.9
Guru	1901	11 160	51.8	5.6	42.6	20.6	56.6	Egyp	829	182 167	4.7	65.5	29.8	8.4	26.9
Aghb	2	46	50.0	0.0	50.0	1.1	94.0	Wara	64	6591	4.7	79.7	15.6	10.1	4.5
Gran	2	17	50.0	0.0	50.0	1.1	68.4	Linb	65	4158	4.6	86.2	9.2	4.5	7.0
Kthi	2	47	50.0	50.0	0.0	0.5	72.7	Sgnw	48	2154	4.2	54.2	41.7	33.6	58.7
Perm	2	10	50.0	50.0	0.0	0.8	12.5	Mend	27	1304	3.7	66.7	29.6	28.9	0.9
Wcho	2	22	50.0	0.0	50.0	1.0	56.5	Zzzz	54	6707	3.7	11.1	85.2	31.6	1.1
Brah	35	1004	48.6	25.7	25.7	20.1	9.2	Hluw	348	52 052	3.2	71.0	25.9	16.8	33.5
Avst	11	209	45.5	27.3	27.3	1.0	43.4	Saur	131	9763	3.1	82.4	14.5	5.4	3.5
Java	12	915	41.7	41.7	16.7	3.5	1.2	Kits	70	4204	2.9	71.4	25.7	13.2	60.3
Gujr	903	6560	40.1	44.0	15.9	5.2	4.0	Limb	38	11 146	2.6	78.9	18.4	0.9	34.3
Ethi	978	20 769	38.9	29.1	32.0	22.8	22.2	Dupl	41	2476	2.4	80.5	17.1	7.9	3.2
Cyrl	100 252	910 437	37.5	26.9	35.5	25.2	15.8	Brai	43	2924	2.3	79.1	18.6	5.9	96.7
Knda	835	10 982	37.5	24.0	38.6	6.2	33.5	Tang	1588	322 750	2.1	65.1	32.8	10.6	24.4
Geor	870	4887	37.4	40.6	22.1	1.9	7.7	Yiii	1790	502 871	1.7	66.5	31.8	9.5	22.9
Cham	11	529	36.4	36.4	27.3	31.6	98.8	Armi	6	182	0.0	100.0	0.0	0.5	25.6
Bamu	257	16 334	36.2	49.0	14.8	8.2	53.4	Batk	3	24	0.0	100.0	0.0	0.5	0.0
Osge	25	496	36.0	48.0	16.0	33.6	99.1	Bhks	7	269	0.0	85.7	14.3	0.7	41.0
Hang	30 204	186 766	33.6	24.5	41.9	278.4	2.6	Bopo	31	2333	0.0	83.9	16.1	1.0	19.9
Ital	3	109	33.3	0.0	66.7	1.2	1.6	Buhd	1	7	0.0	0.0	100.0	2.4	0.0
Mero	3	76	33.3	33.3	33.3	0.8	37.3	Cakm	10	342	0.0	50.0	50.0	0.9	31.8
Pauc	6	141	33.3	33.3	33.3	1.1	59.3	Cher	29	3313	0.0	41.4	58.6	15.2	1.6
Sogd	3	39	33.3	33.3	33.3	1.1	80.5	Cpmn	40	1879	0.0	60.0	40.0	11.0	98.7
Toto	3	60	33.3	33.3	33.3	1.1	6.1	Cprt	14	367	0.0	50.0	50.0	0.9	55.4
Sinh	1296	22 935	32.3	38.9	28.9	2.8	32.9	Dogr	2	8	0.0	50.0	50.0	0.9	0.0
Mtei	16	649	31.2	56.2	12.5	25.8	98.8	Elba	3	13	0.0	66.7	33.3	3.0	71.8
Vith	10	325	30.0	40.0	30.0	51.0	98.9	Hano	1	44	0.0	100.0	0.0	0.6	0.0
Khmr	6568	26 009	29.0	7.1	63.8	14.6	46.9	Hatr	2	24	0.0	100.0	0.0	0.8	45.0
Mlym	719	4186	28.4	62.9	8.8	0.7	15.1	Khar	8	406	0.0	50.0	50.0	7.5	1.6
Jpan	359	5178	27.9	23.4	48.7	6.2	6.4	Lepc	6	261	0.0	66.7	33.3	63.2	0.3
Thai	25 533	342 758	25.2	35.5	39.3	131.9	15.4	Lyci	14	408	0.0	78.6	21.4	0.8	50.3
Gong	4	92	25.0	75.0	0.0	0.7	7.8	Mahj	4	115	0.0	50.0	50.0	1.0	34.8
Merc	12	296	25.0	50.0	25.0	55.9	12.5	Maka	3	85	0.0	66.7	33.3	0.9	87.2
Thaa	8	310	25.0	62.5	12.5	0.8	15.1	Mand	1	22	0.0	0.0	100.0	1.5	100.0
Laoo	1993	23 184	24.9	54.7	20.3	3.8	55.8	Mani	6	828	0.0	66.7	33.3	21.0	0.5
Hani	1784	55 538	24.3	34.4	41.4	19.3	34.5	Mroo	9	1258	0.0	66.7	33.3	16.7	93.2
Adlm	9	60	22.2	55.6	22.2	1.0	16.4	Nagm	2	32	0.0	50.0	50.0	1.0	65.6
Glag	14	700	21.4	42.9	35.7	6.8	2.5	Nand	2	22	0.0	0.0	100.0	2.8	93.4
Ahom	5	72	20.0	60.0	20.0	0.8	39.0	Nbat	13	318	0.0	84.6	15.4	0.7	92.8
Gonm	5	121	20.0	80.0	0.0	0.5	45.2	Orkh	7	538	0.0	57.1	42.9	1.2	92.1
Ogam	5	198	20.0	60.0	20.0	0.7	30.6	Osma	5	253	0.0	80.0	20.0	65.1	0.7
Tibt	95	4306	20.0	44.2	35.8	18.5	77.1	Ougr	2	49	0.0	100.0	0.0	0.5	20.8
Cans	111	6090	18.9	53.2	27.9	14.6	53.7	Phli	3	98	0.0	66.7	33.3	0.6	18.6
Telu	1485	4240	18.9	15.1	66.0	1066.2	3.6	Phlp	1	62	0.0	0.0	100.0	1.0	34.9
Sidd	50	1185	18.0	64.0	18.0	10.4	38.7	Phnx	3	81	0.0	100.0	0.0	0.7	46.6
Cari	17	1359	17.6	64.7	17.6	15.5	0.8	Prti	1	40	0.0	100.0	0.0	0.3	7.1
Taml	1120	4933	17.1	5.9	77.0	1205.8	1.2	Rjng	1	40	0.0	100.0	0.0	0.3	100.0
Hung	18	442	16.7	66.7	16.7	18.5	98.7	Rohg	8	522	0.0	62.5	37.5	31.8	99.5
Kali	12	160	16.7	83.3	0.0	0.7	30.8	Samr	3	122	0.0	100.0	0.0	0.3	19.0
Hebr	9410	125 431	16.3	56.4	27.3	23.3	6.7	Sarb	1	54	0.0	100.0	0.0	0.7	94.7
Dsrt	13	295	15.4	46.2	38.5	13.5	97.8	Shaw	3	29	0.0	100.0	0.0	0.4	16.7
Lana	104	3766	14.4	45.2	40.4	27.1	68.2	Shrd	24	1308	0.0	79.2	20.8	13.1	1.6
Mult	7	212	14.3	85.7	0.0	0.6	52.6	Sind	1	3	0.0	0.0	100.0	669.3	0.0
Olck	7	295	14.3	57.1	28.6	55.8	100.0	Sogo	3	71	0.0	100.0	0.0	0.8	12.3
Copt	196	4919	13.3	76.0	10.7	4.9	4.7	Sora	2	167	0.0	50.0	50.0	0.7	64.6
Tavt	8	205	12.5	75.0	12.5	0.7	71.7	Soyo	4	77	0.0	75.0	25.0	0.8	61.9
Modi	25	537	12.0	76.0	12.0	0.6	48.1	Tagb	5	79	0.0	80.0	20.0	0.8	45.0
Newa	26	838	11.5	61.5	26.9	0.8	83.4	Takr	4	265	0.0	50.0	50.0	0.6	9.1
Diak	9	157	11.1	55.6	33.3	0.8	77.6	Tfng	7	135	0.0	28.6	71.4	1.5	31.5
Phag	37	812	10.8	64.9	24.3	0.7	81.5	Tglg	2	29	0.0	100.0	0.0	0.9	61.5
Xsux	480	71 375	10.6	65.8	23.5	9.9	21.3	Tirh	3	130	0.0	100.0	0.0	0.5	60.3
Hmng	10	309	10.0	90.0	0.0	0.7	96.1	Tnsa	8	191	0.0	50.0	50.0	0.9	57.6
Kawi	10	495	10.0	40.0	50.0	7.3	1.6	Ugar	1	6	0.0	100.0	0.0	0.7	0.0
Lisu	10	714	10.0	60.0	30.0	26.3	0.1	Vaii	53	12 064	0.0	47.2	52.8	14.3	31.3
Talu	31	815	9.7	54.8	35.5	2.6	29.3	Xpeo	20	713	0.0	85.0	15.0	0.7	26.6
Mong	96	4290	9.4	50.0	40.6	7.6	17.9	Yezi	6	105	0.0	16.7	83.3	2.3	70.1
Syrc	66	10 461	9.1	81.8	9.1	3.6	62.5	Zanb	5	383	0.0	60.0	40.0	7.6	2.3
...	...	...	...	...	...	...	...	All	991 441	8 063 322	64.8	16.0	19.2	31.7	13.4

Table 20: Exploratory prompt: LLM instruction following metrics sorted by adherence: 32K-token input sequence limit. Number of PUA characters omitted from LLM prompt.

Code	N_S	N_T	A	E_U	E_O	R_C	R_L
Elym	3	9	100.0	0.0	0.0	1.0	55.6
Palm	1	2	100.0	0.0	0.0	1.0	100.0
Orya	2006	2327	96.6	0.2	3.1	1.0	83.9
Marc	7	30	85.7	14.3	0.0	0.9	57.1
Medf	403	2892	79.7	1.0	19.4	1.1	1.7
Latn	657 219	2 447 839	79.1	10.4	10.4	3.2	31.7
Grek	7209	34 234	78.5	12.7	8.9	3.3	21.0
Guru	1075	8364	68.8	5.0	26.1	4.6	3.1
Bali	3	37	66.7	33.3	0.0	0.9	62.5
Armn	366	1779	65.6	23.2	11.2	10.1	93.6
Beng	377	843	65.3	19.6	15.1	1.0	24.0
Taml	260	2607	65.0	20.8	14.2	7.0	5.0
Arab	7220	34 923	63.1	14.6	22.3	3.3	59.8
Ethi	432	5612	58.6	17.1	24.3	9.4	55.8
Deva	3411	15 501	58.4	21.5	20.1	3.0	34.9
Mymr	1570	8673	56.4	19.1	24.5	7.1	14.4
Hebr	742	9846	51.3	29.4	19.3	19.0	82.8
Telu	344	1578	50.3	39.0	10.8	0.8	27.9
Aghb	2	46	50.0	0.0	50.0	1.1	94.0
Gran	2	17	50.0	0.0	50.0	1.1	68.4
Ital	2	55	50.0	0.0	50.0	1.2	1.5
Kthi	2	47	50.0	50.0	0.0	0.5	72.7
Perm	2	10	50.0	50.0	0.0	0.8	12.5
Wcho	2	22	50.0	0.0	50.0	1.0	56.5
Hang	14 278	65 885	49.0	33.4	17.5	2.9	42.1
Brah	35	1004	48.6	25.7	25.7	20.1	9.2
Avst	11	209	45.5	27.3	27.3	1.0	43.4
Java	12	915	41.7	41.7	16.7	3.5	1.2
Cyrl	80 795	596 194	40.1	26.3	33.6	2.2	82.4
Cham	10	413	40.0	30.0	30.0	40.2	99.4
Geor	687	3492	39.9	42.6	17.5	0.9	14.8
Knda	748	10 261	38.4	25.9	35.7	5.0	44.0
Gujr	842	6173	37.8	46.2	16.0	2.9	7.1
Bamu	253	7939	36.8	49.4	13.8	12.2	61.3
Osge	25	496	36.0	48.0	16.0	33.6	99.1
Thai	18 020	146 225	34.7	41.3	24.0	12.2	72.0
Mero	3	76	33.3	33.3	33.3	0.8	37.3
Pauc	6	141	33.3	33.3	33.3	1.1	59.3
Sogd	3	39	33.3	33.3	33.3	1.1	80.5
Toto	3	60	33.3	33.3	33.3	1.1	6.1
Mtei	16	649	31.2	56.2	12.5	25.8	98.8
Glag	10	644	30.0	40.0	30.0	5.0	2.6
Vith	10	325	30.0	40.0	30.0	51.0	98.9
Sinh	1215	22 198	29.9	41.2	29.0	1.3	74.6
Khmr	6436	22 846	29.5	6.7	63.9	8.2	39.7
Mlym	707	4004	28.3	63.6	8.1	0.7	16.3
Jpan	355	4988	27.9	23.7	48.5	6.4	5.6
Laoo	1919	20 651	25.4	55.0	19.5	3.1	74.9
Gong	4	92	25.0	75.0	0.0	0.7	7.8
Merc	12	296	25.0	50.0	25.0	55.9	12.5
Thaa	8	310	25.0	62.5	12.5	0.8	15.1
Hani	1704	53 901	24.8	35.7	39.6	18.4	36.3
Adlm	9	60	22.2	55.6	22.2	1.0	16.4
Tibt	92	4014	20.7	44.6	34.8	19.7	77.1
Ahom	5	72	20.0	60.0	20.0	0.8	39.0
Gonm	5	121	20.0	80.0	0.0	0.5	45.2
Ogam	5	198	20.0	60.0	20.0	0.7	30.6
Cans	109	4788	19.3	53.2	27.5	16.3	61.0
Cari	17	1359	17.6	64.7	17.6	15.5	0.8
Hung	18	442	16.7	66.7	16.7	18.5	98.7
Kali	12	160	16.7	83.3	0.0	0.7	30.8
Lana	84	1685	16.7	51.2	32.1	4.1	43.5
Olck	6	90	16.7	66.7	16.7	0.9	96.4
Dsrt	13	295	15.4	46.2	38.5	13.5	97.8
Sidd	47	1157	14.9	68.1	17.0	0.7	90.8
Kawi	7	165	14.3	57.1	28.6	1.0	30.9
Mult	7	212	14.3	85.7	0.0	0.6	52.6
Copt	195	4699	13.3	76.4	10.3	4.1	5.7
Tavt	8	205	12.5	75.0	12.5	0.7	71.7
Modi	25	537	12.0	76.0	12.0	0.6	48.1
Xsux	437	33 886	11.7	70.5	17.8	9.8	34.9
Newa	26	838	11.5	61.5	26.9	0.8	83.4
Diak	9	157	11.1	55.6	33.3	0.8	77.6
Lisu	9	310	11.1	66.7	22.2	53.0	0.2
Phag	37	812	10.8	64.9	24.3	0.7	81.5
Hmng	10	309	10.0	90.0	0.0	0.7	96.1
Hmnp	10	523	10.0	70.0	20.0	16.3	0.8
Mong	93	3060	9.7	48.4	41.9	10.4	16.8
...	...	...	...	...	...	...	...

Code	N_S	N_T	A	E_U	E_O	R_C	R_L
...	...	...	...	...	...	...	...
Talu	31	815	9.7	54.8	35.5	2.6	29.3
Syrc	65	3663	9.2	81.5	9.2	9.4	69.4
Bass	12	1927	8.3	33.3	58.3	6.1	0.0
Lina	72	4136	8.3	66.7	25.0	5.1	6.0
Runr	12	528	8.3	75.0	16.7	5.0	5.2
Nshu	50	2173	8.0	68.0	24.0	10.7	81.9
Hira	29	1677	6.9	62.1	31.0	3.1	5.1
Nkoo	45	867	6.7	68.9	24.4	11.1	2.8
Plrd	46	1158	6.5	67.4	26.1	15.0	98.8
Egyp	700	57 503	5.6	72.7	21.7	8.6	41.5
Kana	36	2217	5.6	61.1	33.3	10.7	12.7
Sund	18	960	5.6	72.2	22.2	0.6	57.9
Wara	59	3278	5.1	83.1	11.9	5.6	6.6
Linb	64	3920	4.7	85.9	9.4	4.8	6.9
Sgnw	48	2154	4.2	54.2	41.7	33.6	58.7
Mend	25	1215	4.0	72.0	24.0	29.1	1.0
Hluw	308	31 122	3.6	76.6	19.8	13.0	38.4
Saur	126	7482	3.2	85.7	11.1	3.5	6.4
Zzzz	32	1465	3.1	12.5	84.4	62.7	2.5
Kits	68	3014	2.9	73.5	23.5	9.6	59.5
Limb	37	8105	2.7	78.4	18.9	1.2	34.3
Dupl	40	2044	2.5	82.5	15.0	1.6	19.4
Tang	1352	115 930	2.5	73.2	24.3	11.8	37.2
Brai	42	1813	2.4	81.0	16.7	0.5	42.8
Yiii	1392	134 304	2.2	79.1	18.8	11.5	34.9
Armi	6	182	0.0	100.0	0.0	0.5	25.6
Batk	3	24	0.0	100.0	0.0	0.5	0.0
Bhks	7	269	0.0	85.7	14.3	0.7	41.0
Bopo	28	1072	0.0	85.7	14.3	0.7	57.6
Buhd	1	7	0.0	0.0	100.0	2.4	0.0
Cakm	10	342	0.0	50.0	50.0	0.9	31.8
Cher	26	1321	0.0	46.2	53.8	11.5	3.7
Cpmn	40	1879	0.0	60.0	40.0	11.0	98.7
Cprt	14	367	0.0	50.0	50.0	0.9	55.4
Dogr	2	8	0.0	50.0	50.0	0.9	0.0
Elba	3	13	0.0	66.7	33.3	3.0	71.8
Hano	1	44	0.0	100.0	0.0	0.6	0.0
Hatr	2	24	0.0	100.0	0.0	0.8	45.0
Khar	8	406	0.0	50.0	50.0	7.5	1.6
Lepc	6	261	0.0	66.7	33.3	63.2	0.3
Lyci	14	408	0.0	78.6	21.4	0.8	50.3
Mahj	4	115	0.0	50.0	50.0	1.0	34.8
Maka	3	85	0.0	66.7	33.3	0.9	87.2
Mand	1	22	0.0	0.0	100.0	1.5	100.0
Mani	4	153	0.0	100.0	0.0	0.8	63.5
Mroo	8	630	0.0	75.0	25.0	7.4	69.1
Nagm	2	32	0.0	50.0	50.0	1.0	65.6
Nand	2	22	0.0	0.0	100.0	2.8	93.4
Nbat	13	318	0.0	84.6	15.4	0.7	92.8
Orkh	6	537	0.0	66.7	33.3	1.0	90.7
Osma	5	253	0.0	80.0	20.0	65.1	0.7
Ougr	2	49	0.0	100.0	0.0	0.5	20.8
Phli	3	98	0.0	66.7	33.3	0.6	18.6
Phlp	1	62	0.0	0.0	100.0	1.0	34.9
Phnx	3	81	0.0	100.0	0.0	0.7	46.6
Prti	1	40	0.0	100.0	0.0	0.3	7.1
Rjng	1	40	0.0	100.0	0.0	0.3	100.0
Rohg	8	522	0.0	62.5	37.5	31.8	99.5
Samr	3	122	0.0	100.0	0.0	0.3	19.0
Sarb	1	54	0.0	100.0	0.0	0.7	94.7
Shaw	3	29	0.0	100.0	0.0	0.4	16.7
Shrd	23	1179	0.0	78.3	21.7	14.4	1.6
Sind	1	3	0.0	0.0	100.0	669.3	0.0
Sogo	3	71	0.0	100.0	0.0	0.8	12.3
Sora	2	167	0.0	50.0	50.0	0.7	64.6
Soyo	4	77	0.0	75.0	25.0	0.8	61.9
Tagb	5	79	0.0	80.0	20.0	0.8	45.0
Takr	4	265	0.0	50.0	50.0	0.6	9.1
Tfng	6	80	0.0	33.3	66.7	1.5	50.8
Tglg	2	29	0.0	100.0	0.0	0.9	61.5
Tirh	3	130	0.0	100.0	0.0	0.5	60.3
Tnsa	8	191	0.0	50.0	50.0	0.9	57.6
Ugar	1	6	0.0	100.0	0.0	0.7	0.0
Vaii	48	6385	0.0	52.1	47.9	20.5	39.4
Xpeo	20	713	0.0	85.0	15.0	0.7	26.6
Yezi	6	105	0.0	16.7	83.3	2.3	70.1
Zanb	5	383	0.0	60.0	40.0	7.6	2.3
All	817 177	4 021 097	72.0	14.0	14.0	4.6	42.1

Table 21: Exploratory prompt: LLM instruction following metrics sorted by adherence: 64-token input sequence limit. Number of PUA characters omitted from LLM prompt.

HAnnoI: A Handwriting Annotation Interface to Extract Data for Linguistic Analyses of Graphetic Detail

Joshua Wieler[1,3], **Simon Petitjean**[1], **Kristian Berg**[1], **Henriette Huber**[2], **Stefan Hartmann**[2]

[1]Carl von Ossietzky University of Oldenburg, Institute for German Studies
simon.petitjean, kristian.berg@uni-oldenburg.de

[2]Heinrich Heine University Düsseldorf, Department of German Linguistics
henriette.huber, hartmast@hhu.de

[3]Ruhr University Bochum, Linguistics Department
joshua.wieler@ruhr-uni-bochum.de

Abstract

In this paper, we present *HAnnoI* – short for *Handwriting Annotation Interface* –, an open-source GUI application developed in Python that allows its users to identify and annotate so-called Regions of Interest (ROIs) within digital images. Several meta data such as their coordinates are retained for each ROI and they can be annotated on user-defined annotation layers. HAnnoI comes with a function to export all annotations to a CSV file, enabling further processing as well as quantitative analyses. HAnnoI also has a function to extract single PNG image files of all ROIs. It was developed to mark and annotate single letters in scans of handwritten (alphabetic) texts for linguistic analyses, yet it is not limited to this particular use case. In this paper, we first provide information on HAnnoI's conception and technical details as well as an overview of alternative applications. We then showcase HAnnoI's capabilities in a letter annotation task, where five annotators marked instances of lower case <s> in handwritten texts. Finally, we report on an exploratory analysis of this data, showing what kinds of investigations are enabled by using HAnnoI. The tool is available for free use at https://github.com/pywielR/HAnnoI.

Keywords: Tools, Systems, Applications, Handwritten, Typewritten Document Recognition, Corpus (Creation, Annotation, etc.)

1. Introduction

This paper presents *HAnnoI*, the *Handwriting Annotation Interface*, a lightweight, platform-independent, easy-to-use GUI application developed for identifying and annotating single letters in handwritten data. Originally designed for working with scans of texts written by hand in an alphabetic writing system, the tool can be used to process just about any kind of visually available, digitized (language) data. If one thinks of an image file as a map, HAnnoI provides a toolkit that allows users to chart its geography by adding rectangular frames modifiable in size, color and position. Each frame marks a so-called Region of Interest (ROI), whose measures and coordinates are being tracked by the program, and that can be enriched with a variable amount of user-defined annotations. HAnnoI also allows for exporting the ROIs as individual PNG image files and the annotations as a CSV spreadsheet, where each row represents one ROI with all of its metadata and "geographical" information as well as its custom annotations retained in individual columns. This data may then be used for further processing and quantitative analyses. Furthermore, it can be imported back into the tool to continue working on a document.

HAnnoI's built-in function to save individual PNG images of all ROIs furthermore enables investigations of the graphetic part of language production, i.e., the concrete shape that linguistic units take in handwriting. Because each extracted image is linked to its annotations in the CSV spreadsheet via an index, and said CSV may be imported back into the program, HAnnoI makes it very easy to trace back, inspect and revise previous work, making it a powerful resource for data documentation as well.

HAnnoI was developed to handle specific problems that we encountered during the annotation of handwritten letters in an alphabetic, left-to-right directed script. This is of course reflected in the features currently implemented in the tool, which may also limit its applicability to other scripts, especially those that do not adhere to a horizontal writing direction. We still suggest that it can be useful even beyond its original purpose of annotating single letters. For example, users might choose to annotate whole words or even sentences, handwritten or in print. Any field researching such data may be supplemented using HAnnoI, for example, studies on

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 33–44
12 May, 2026.

graphetic variation (e.g., Reinken, 2023), but also Optical Character Recognition (OCR) technology (see Memon et al., 2020 for a review on handwriting specifically).

Because the tool tracks geographical information, it allows for analyses of spatial relations between annotated units. In research on handwriting, this could involve investigations of the influence that adjacent letters have on each other. Kandel and Perret (2014), for example, have observed stroke length differences in cursive realizations of the letter ‹l› when it was connected to a following ‹l› as compared to when it was connected to a following ‹e› or ‹n›, indicating anticipatory effects in handwriting, a notion they refer to as *motor anticipation*. While this particular study had children produce isolated bigrams using a digital pen and writing tablet, with HAnnoI, this could be extended to naturalistic data like scans of handwritten texts, as can be found in the GraphVar Corpus (Berg et al., 2021) that will be introduced in more detail in Section 3.

Also, suppose a researcher has collected scans of written documents and is interested in a specific linguistic phenomenon, yet the data has not been transcribed and annotated yet. In this case HAnnoI can be used to identify linguistic units of interest and annotate them on a custom set of layers, while simultaneously tracking their position. Both the units of interest as well as the annotations can be extended at a later stage of the project if necessary. All too often, researchers still use pen and paper tally sheets in these cases, which is not only cumbersome but also more error-prone – HAnnoI offers a significant improvement while at the same time ensuring reproducibility.

The remainder of this paper is structured as follows: First, we compare HAnnoI to similar applications that are available for free use. After that, in Section 2, we provide a more detailed description of the tool's functionalities. This is followed by a concrete application example in which five annotators were instructed to annotate instances of lower case ‹s› in handwritten documents (Section 3). These annotations have then been used in an analysis of ‹s› variation in handwriting, specifically, whether there are variables that can predict the usage of either cursive or block letter ‹s› (see Figure 1 for an example of this variation). Further details on the analysis will be provided in Section 4. The paper closes discussing some of HAnnoI's current limitations, as well as providing an outlook on further developments.

1.1. Comparable tools

Most tools specifically designed to annotate linguistic data focus on already digitized texts (e.g., INCEpTION, Klie et al., 2018; doccano, Nakayama et al., 2018) or video/audio recordings (ELAN, Sloetjes and Wittenburg, 2008). When it comes to undigitized texts, eScriptorium[1] can be used for transcription of handwriting. It also allows users to mark certain regions in an image, though this feature is not suited for multi-layer annotations as it only allows for one label per marking.

This limitation is also prevalent in other tools we tested, such as the web-based application makesense.ai by Skalski (2019) and PixLab Annotate[2]. These tools are more oriented towards object detection and labeling (generating training data for machine learning), not towards extensive linguistic annotation, which explains their limitations with regard to the accumulation of multivariate data.

The applications that are closest to HAnnoI in functionality are the VGG Image Annotator (VIA) by Dutta and Zisserman (2019) and IMMARKUS by De Weerdt et al. (2024). Both tools are web-based and allow their users to (1) mark ROIs of different shapes within an image (e.g., individual letters/words in scans of handwritten text), and (2) add annotations on multiple layers to each ROI. However, for VIA, we found that the marking process is less precise than in HAnnoI and it also does not allow for the extraction of the marked areas as PNG or similar image files. As for IMMARKUS, it is a powerful tool that brings many of HAnnoI's features and offers additional functionality such as smart scissors and auto transcription, making it a good choice in a wide range of situation. We encourage anyone who is interested in conducting research similar to what is described throughout this paper, to also test IMMARKUS for its applicability, as depending on what their objective is, it might suit their specific needs better.

Beyond VIA and IMMARKUS, we did not find many other tools that can be used in a similar manner. Mick O'Donnell's UAM ImageTool[3] provides some of the same functionalities, though it lacked precision when drawing ROIs and seems to be no longer maintained, with its latest version having been released in 2011. Note that in our comparison we did not consider applications that require a paid subscription such as Transkribus[4].

2. Technical details

The code for HAnnoI is written in Python, using the PyQt framework (version 6) for GUI development. The tool runs from a single Python script that loads all of its dependencies upon launch (they have to be installed once beforehand). The source code and a quick installation guide can be found here: https://github.com/pywielR/HAnnoI.

[1] https://ocr-bw.bib.uni-mannheim.de/escriptorium/
[2] https://annotate.pixlab.io/
[3] http://www.wagsoft.com/ImageTool/index.html
[4] https://www.transkribus.org/de

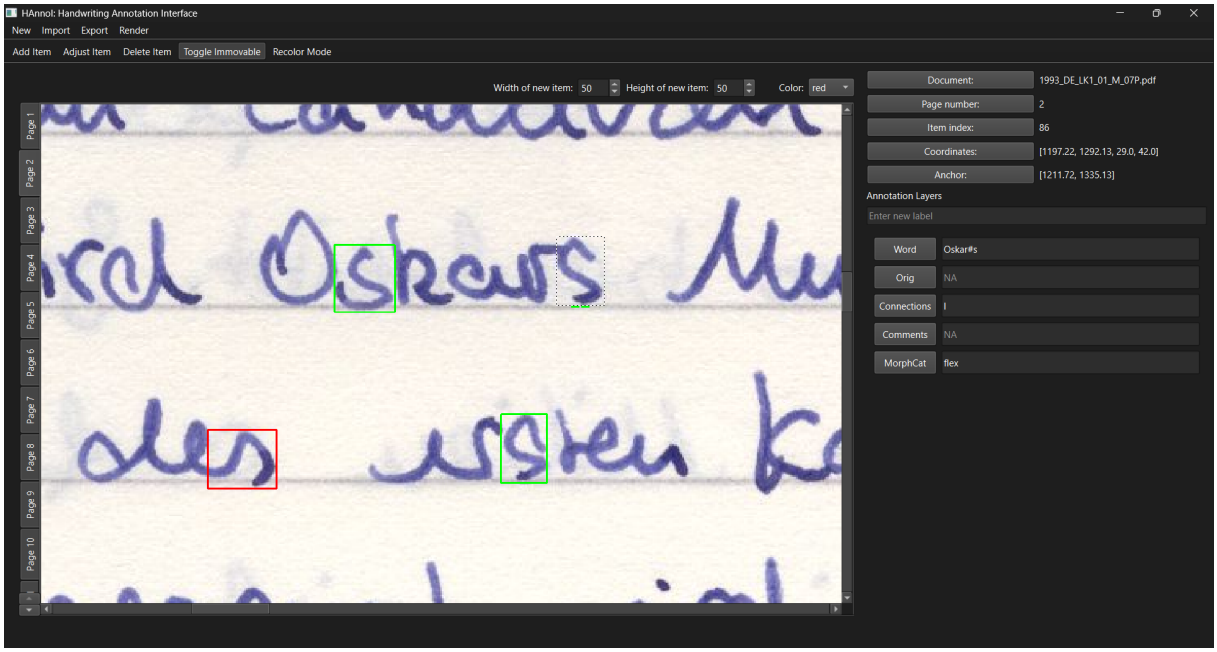


Figure 1: HAnnoI's interface displaying part of one page of a document containing handwritten text. In this zoomed-in view, one of the four visible ROIs is currently selected (indicated by the dashed lines); its corresponding annotations are shown in the panel on the right. Different colors have been used to mark either cursive instances of <s> (red) or block letters (green).

Besides the menu bar at the top, which provides options to start, save and continue with a project, HAnnoI's user interface consists of two main areas (see Figure 1): (1) the graphical view on the left, in which the user may add ROIs to imported images; and (2) the annotation panel on the right.

2.1. Graphical view and meta data

In HAnnoI, it is possible to either load in single image files or multiple images extracted from a PDF document. The user may zoom in and out of the graphical view as well as drag it around with the mouse. If an imported PDF document consists of multiple pages, each page gets an individual tab on the left-hand side of the graphical view. The header above offers buttons for adding, modifying and deleting ROIs, though some of these actions are also covered by keyboard or mouse inputs. For example, ROIs can be added by right-clicking into the displayed image and selecting *Add Item* from a context menu. An ROI can be selected and dragged around using the mouse. One can also use arrow keys to adjust its position. By holding Ctrl or Shift, arrow keys change a ROI's size in 1- or 5-pixel-increments, respectively. For a full list of HAnnoI's shortcuts, we refer to the manual in the GitHub repository.

By default, the annotation panel on the right displays the meta information for any selected ROI, such as the source document, the page number, the index as well as its coordinates and size[5]; furthermore, there is the option to set a so-called anchor for each ROI (the green dashed line just below the selected ROI in Figure 1), which in the example here was utilized to track the position of the horizontal line on the writing paper which the ROI-corresponding word was written on. Anchors are always set along the bottom edge of a selected ROI by pressing Space. A set anchor retains its position even when moving the ROI it belongs to, but can be overridden by pressing space again.

2.2. Custom annotations

Custom annotation layers can be added by entering a label in the designated field and pressing Return. Five of these layers have been added to the example shown in Figure 1 (*Word*, *Orig*, *Connections*, *Comments* and *MorphCat*). Currently, all annotation layers are text input fields. After selecting a ROI, users may type in their annotations. Additionally, it is possible to implement a categorical variable after which ROIs can be annotated with a certain feature by simply selcting it with the mouse. The user manual in the GitHub repository contains more detailed instructions on how to use this as

[5]The coordinates and size of a selected ROI are displayed together as a list in the *Coordinates* annotation layer in this order: x-coordinate, y-coordinate, width, height. The x- and y-coordinates always refer to the upper-left corner point of a rectangle.

well as other features implemented to streamline the annotation process.

2.3. Exporting data

While working with HAnnoI, meta data as well as annotations are stored in standard Python dictionaries, which are compiled into CSV format upon export. For an example of the output, see Appendix A1. Exporting annotations is also the way to save one's work, since the resulting CSV can be imported again in order to continue with the file. The corresponding single image or PDF document has to be retained by the user as a separate file, however, because the source of the image(s) itself is only included as the file's name in the CSV (in the Source column). For easy import, it is recommended to just keep the CSV and source file in the same folder, in which case HAnnoI automatically loads in the corresponding images. If the source file is not in the same folder, the user is asked to select it manually.

CSV files can be edited outside of HAnnoI and still be imported. For example, one could manually change the entries in the color column in order to change the visual appearance of ROIs inside the program. One should be cautious, however, as some changes, in particular those to the meta information or the column structure, may cause HAnnoI to fail upon trying to import a CSV file. As for the custom annotation layers, it is possible to add whole new columns at the right end of the spreadsheet outside of HAnnoI, which would then be recognized as being one of the custom annotation layers inside the tool. This allows for working on a file in and outside of the tool, offering flexibility to the annotator's work.

Finally, the function to render single PNG image files for each ROI placed in the document can be found at the right end of the upper menu bar. The file name of these extracted images contains a reference to the source as well as an index that links each image to its entry in the corresponding spreadsheet. As the CSV export function and the rendering function are independent from each other, one should keep in mind that changes to the indices in the CSV file do not carry over to the PNG files until one uses the rendering function once again. Extracted images display exactly the material that was framed, so a 29x42 pixel ROI would be saved as a 29x42 pixel PNG file. The colored frames of ROIs are hidden in the extracted images. Appendix A2 gives an example of a folder containing rendered images extracted from a multi-page document that has been annotated with HAnnoI.

The next two sections demonstrate in two complementary ways the functionalities of HAnnoI in practice. Section 3 presents a concrete annotation task in which multiple annotators gathered handwritten data using the tool, allowing us to assess its usability and the quality of the collected data. Section 4 then shows how the annotated data can be immediately used for quantitative linguistic analysis. Together, these two examples show a typical research workflow that HAnnoI is meant to support, making data collection faster, easier to reproduce, and convenient to maintain and extend over time.

3. Annotation example and assessment

HAnnoI has been tested on PCs running on Windows as well as macOS, in an annotation task involving five student assistants. After ascertaining that the system's core features are properly functioning, we allocated a set of German school-exit exams taken from the GraphVar Corpus (Berg et al., 2021) to the annotators. Their task involved framing all instances of lowercase ‹s› and ‹e› as ROIs. Each of these letters was then annotated with the (orthographically corrected[6]) word form it was a part of. Morpheme boundaries within that word annotation were marked using the special sign #. HAnnoI's Annotation Mode was used to furthermore annotate morpheme categories, for example, whether a letter was part of the word's stem or of an inflectional suffix. Figure 1 as well as Appendix A provide a snippet taken from the results of this annotation task.

Before HAnnoI was introduced to the student assistants, they had already collected data in a similar annotation task, though instead of marking and annotating ROIs using a dedicated tool, they were instructed to take individual screenshots of either ‹s› or ‹e› and catalog these screenshots in a spreadsheet, adding annotations along the way. To document a letter's position in an exam, the coders manually annotated the page and line a letter was written on. In HAnnoI, positional information is tracked automatically and with higher precision as it retains the x- and y-coordinate of any ROI. Additionally, indexing is automatized in HAnnoI as well. Before, the coders had to annotate indices themselves. While it is hard to quantify whether the implementation of HAnnoI brought an increase in efficiency, for this application example, we argue that, when compared to before, the strongest asset of the tool is the traceability it provides for the data that is produced with it, and that this necessarily culminates in data that is of higher quality. To justify this claim, we will elaborate on two examples that were taken from the outlined annotation task.

As of the time of writing, only using HAnnoI and not regarding the data that had been collected be-

[6] For spelling errors, we created a separate layer labeled *Orig*.

fore, the coders have gathered annotations for 7056 instances of <s> across twelve exams as well as for 2084 instances of <e> in a single exam. While the annotators mostly worked on different exams, we sometimes assigned the same exam to multiple coders in order to assess agreement across annotators as well as the functionality of HAnnoI.

Example I: Working on the same file

In case of the annotation of <e>, we had a first annotator frame all instances of this letter as ROIs alongside annotating the corresponding word. After that, the second annotator was provided with the resulting CSV to load into the program, effectively reproducing all of the first annotator's work in an instant. The second annotator then added information on morpheme category as well as noting whether a framed letter had a visible connection to adjacent material (such as is the case for the selected <s> in Figure 1). In HAnnoI, revisions like this are rather uncomplicated, as it brings together every annotation and its visual reference in a single interface. Before introducing the tool, this would have required the second annotator juggling the screenshots, the spreadsheet and the source file, to manually search the latter for every instance of <e> and then update the annotations, rendering this approach very slow and prone to errors, if it was at all feasible. Besides updating entire files, with HAnnoI one can also easily trace back and edit the annotations of single observations.

Example II: Comparing annotations

As for the <s> annotations, in one case we asked two student assistants to annotate two pages of the same exam. This was done so as to be able to directly compare the work of different annotators on identical data. With this, we calculated inter-annotator agreement mainly as an (early) evaluation of our annotation guidelines and method, though aspects of this can be regarded as indication of HAnnoI's viability.

First, we compared the anchors set by the annotators for each ROI. To reiterate, the instruction was to place these anchors on top of the horizontal baseline an annotated letter was written on. Over the two pages, 94 pairs of <s> annotations were collected by the student assistants. On average, the distance between the anchors was about 0.55 pixels on the y-axis (which is the axis that matters here). Given that the lowest increment to which adjustments to the anchors can be done (via keyboard inputs) is one pixel, we concluded that the annotators aligned their anchors quite nicely, demonstrating that HAnnoI allows for gauging such precise information.

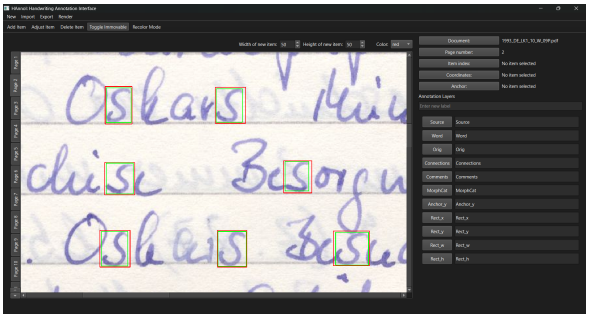


Figure 2: ROIs added by separate annotators during an early stage of the annotation process; red frames were added by the first annotator and green frames by the second annotator.

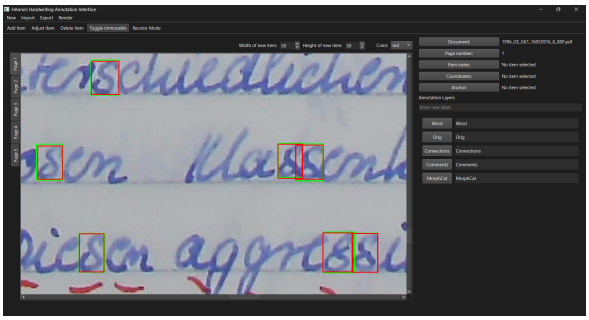


Figure 3: ROIs added by the same annotators from Figure 2, this time during a later stage of the annotation process and after receiving feedback on their earlier framing of ROIs; red frames were again added by the first annotator and green frames by the second annotator.

Furthermore, we used the <s> annotations provided by both annotators to assess how closely they drew ROI frames around identical letters. To achieve this, we copied and pasted the annotations from the second annotator into the CSV file of the first annotator, which, after loading the combined data into HAnnoI, places the frames from both annotators on top of each other. In the spreadsheet, we manually changed the color of the ROIs of the second annotator from red to green, so that after importing, the contribution of both annotators would be distinguishable. With this, we were able to immediately get an impression of the differences between the two annotators.

As can be seen in Figure 2, the green frames were drawn more closely around the letters. On average, the 94 ROI frames placed by the first annotator were about 3.46 pixels wider and 3.22 pixels higher than those placed by the second annotator[7]. As tighter frames give less room for intrusions from adjacent letters, in a discussion of these findings,

[7]To calculate this difference, just as the difference between the anchor placement, we imported the combined data into R.

we concluded that going forward we wanted the frames to fit the letters as close as possible. These and other points discussed were documented for the student helpers to refer to in the continuation of the annotation task. Of course, the annotators could also revisit the annotations they had already collected and adjust their ROIs' frames directly in the tool. Without HAnnoI, such an undertaking would require the students to search the documents for instances of <s> or <e> again (as there was no visual marker in the document itself before) in order to take new screenshots, effectively meaning they would have to repeat a task that had already been done before.

At a later stage, we once again provided the two annotators from above with identical data to annotate in HAnnoI. We repeated the comparison of ROI sizes, though this time on a database comprising 438 pairs of <s>. We found that the difference between both annotators had come down by about 50%. In fact, this time the ROI frames placed by the second annotator were about 1.64 pixels wider and 1.57 pixels higher than those of the first annotator. This suggests that after getting feedback on their annotations, the first annotator paid special attention to drawing the frames as close as possible. This is supported by Figure 3, which exemplifies how the frames of both annotators are drawn rather closely around the letters. Furthermore, we take this as indication that, on the technical side, HAnnoI provides an adequate tool set to realize fine adjustments such as those of the first annotator.

Finally, we would like to point out that this assessment in itself shows that the data that HAnnoI provides can be used in qualitative as well as in quantitative research. We now turn to another example, in which we used part of the data generated during the annotation process to investigate <s> variation in handwriting.

4. Application example: exploring <s> variation in handwriting

The annotation work described in Section 3 was initiated as part of a research project investigating morphologically induced variation in handwriting. In one branch of this project, Petitjean et al. (n.d.) extracted the pen trajectories from individual realizations of <s>, for which single screenshots of these were collected. The extracted trajectories then served as the dependent variable of statistical models including morpheme category, morpheme position, syllable position as well as letter position as predictors.

One of the central findings of this study was that variation in writing trajectories can indeed be partially explained by morphological affiliation. For example, <s> realizations found in inflectional suffixes tend to be somewhat reduced in their shape (i.e., the upper part of <s> does not travel as wide of an arc) when compared to those found in stems. When <s> is part of a derivational morpheme, however, the difference to <s> in stems becomes smaller. Petitjean et al. (n.d.) argue that this supports the notion that the morpheme is a significant processing unit in (written) language production.

During the annotation work, we also noticed that in some of the exams, when writing <s>, the students switched between two types of script, namely cursive and block letters (see Figure 1). Following Meletis (2020), this could be regarded as alternation on the level of *basic shapes*, i.e., abstract forms of <s> that are visually distinct in the sense that their components such as individual strokes use different forms (e.g., straight lines vs. arcs) or connect to each other in different angles. Meletis also remarks that this kind of switching within the same text is rather unusual (p. 114), rendering the observation of this phenomenon in our data an interesting case to provide another demonstration of what kinds of analyses are enabled by using HAnnoI as an annotation tool.

4.1. Database of different <s> types

For our analysis, we only considered exams in which we found constant variation between the different types of <s>. This was the case for five exams out of the twelve mentioned in Section 3. As Table 1 shows, in these exams, we found a total of 3383 instances of <s>. The distribution of <s> types in our data was rather balanced with 1876 written in cursive script (55.5%) and 1507 written as block letters (44.5%).

We instructed the annotators to use red frames to mark what we defined as cursive <s> and green frames for block letters. Note that this binary classification is based mainly on the starting point and first stroke of <s>, where cursive <s> is defined by starting on the left, from where its first stroke travels, in some angle to the baseline, in an upwards-right direction (for comparison, see the one <s> marked in red in Figure 1). In contrast, block letters are defined by the presence (or at least suggestion) of an upper arc that starts more to the right, from where it travels to the left. In this regard, we apply a more coarse classification of basic shapes for <s>, of which Reinken (2023) has defined four in total (in his sample of the GraphVar Corpus). For example, the upper arc of the first <s> in the word *Oskars* in Figure 1 is not fully realized and the whole letter could therefore be regarded as being an instance of a third basic shape. However, as Appendix A2 indicates, even within writers there is a high amount of variation in the actual physical manifestation of a letter, so for this analysis we attributed more fine-grained differences such as the exact shape of the

upper arc of <s> to variance within basic shapes[8] rather than between them.

Our distinction between cursive and block letter <s> becomes meaningful when one considers the writing direction of German, which goes from left to right. What we defined as cursive <s> can therefore easily connect with previous letters, whereas block letters start on a movement that opposes the natural writing direction of German, likely causing a lift of the pen. This notion is supported by our annotations: Using the *Connection* annotation layer, we checked how often cursive and block letter <s> connects with adjacent material. In 1270 (67.7%) out of 1876 cases, cursive <s> connects with a previous letter. For block letter <s>, there is a connection in 312 (20.7%) out of 1507 cases. For connections to following letters, this difference is much smaller, with 318 (17%) connections to the right involving cursive <s> and 402 (26.7%) involving block letter <s>. Note that in the *Connection* annotation layer, every time <s> so much as touches an adjacent letter, this is counted as a connection, meaning they are not necessarily indicative of a fluent writing motion where one letter goes into the next without lifting the pen[9]. However, the stark contrast between cursive and block letter <s> with regard to previous letters strongly suggests that this has to do with the fact that cursive <s> can more easily connect to previous letters.

4.2. Data processing and modeling

The annotated data underwent further processing in a separate Python script where we extracted information such as the position of each <s> in the word as well as in the morpheme it belonged to. We also extracted the immediately preceding and following letter of each observation. This data was then imported into R for the statistical analysis.

From the color coding applied in HAnnoI, we implemented a binary variable, specifically, whether a certain <s> was written in cursive script (TRUE) or as a block letter (FALSE). This served as the dependent variable of a mixed effects logistic regression model (using the `glmer` function in R).

Considering that cursive <s> connects to previous material more often, we wanted to explore whether connectivity can be attributed to certain letters, for which we included random intercepts for the preceding, but also the following letter. Though excluded from our sample, there are exams in which there is no variation between <s> types at all,

Exam No.	Instances of <s> written in ...		
	cursive script	*block letters*	*total*
1	452 (47.7%)	495 (52.3%)	947
2	156 (25.3%)	461 (74.7%)	617
3	446 (69.3%)	198 (30.7%)	644
4	388 (69.5%)	170 (30.5%)	558
5	434 (70.3%)	183 (29.7%)	617
	1876 (55.5%)	1507 (44.5%)	3383

Table 1: Instances of either cursive or block letter <s> and their distribution across five handwritten exams.

which is why we assumed that this phenomenon is highly idiosyncratic. We therefore included random intercepts for the ID of each writer as well as for each target word.

With our fixed effects, we explored the possibility of (1) positional, (2) morphological and (3) geographical influences on the choice of <s>. Specifically, we included the position that <s> takes within a word as well as within a morpheme. Both were coded categorically, where <s> could occupy *initial*, *medial* or *final* position. For morpheme position, we implemented a fourth level labeled *single*, which refers to instances of <s> that make up the whole morpheme, such as is the case in the inflectional suffix marking of the genitive in the word *Lesers* ('reader-GEN'). Prior research suggests that the morpheme might be a relevant processing unit during written language production (e.g, Kandel et al., 2012; Berg et al., 2023). With our operationalization, a morphological effect could show in that the model returns a lower probability for cursive <s> to be realized in morpheme *initial* and *single* position, where a connection to a previous letter would be impeded by a morpheme boundary. Reinken (2023) reports a similar effect for syllable boundaries, where identical bigrams are less likely to connect as compared to when they are within the same syllable.

Furthermore, we added a factor signaling whether the previous <s> in the text was cursive or not. With this factor, we wanted to approximate priming effects, i.e., whether writers tend to repeat the same type of <s>. However, this variable on its own is blind to the distance between subsequent instances of <s>, meaning that a preceding <s> could be located on the same line, likely strengthening its priming effect, just as well as on a previous page. Using the geographical annotations in our data, we therefore implemented a continuous variable that measures the vertical distance (in pixels) between the anchor points of subsequent instances of <s>. We standardized the pixel data to deal with convergence issues during model calculation and then included it in the model as an interaction with previous <s> type. Because this measure cannot

[8] According to Meletis (2020) as well as Reinken (2023) this type of variation can be explained by factors such as the writing situation (here: exam), the writer's fatigue, the pen they use and more.

[9] The main idea behind this layer was to be able to identify instances of <s> without intrusive material.

be calculated for the first ‹s› on each page, we excluded these from the analysis. Our model was therefore fit on a database comprising 3300 observations.

Finally, we also included a measure of how far each ‹s› deviates from its baseline, calculated as the (standardized) vertical distance between the bottom edge of a rectangle and its corresponding anchor. As the starting point of block letter ‹s› is likely to be farther away from the baseline, maybe this results in a different spatial alignment relative to the baseline.

We did not include the height of rectangles in the model, as from a conceptual stand point, we thought that the measurements of any particular ‹s› would be a consequence of what type had been realized rather than it being the cause. We did check the relation between height and ‹s› type separately, however, and found that instances of block letter ‹s› are on average about 1 to 11 pixels higher than cursive ‹s›, depending on the writer. As for width, instances of block letter ‹s› are typically more narrow (about 6 to 12 pixels), though the left and right boundary of letters often cannot be as clearly defined as their height, especially when they connect to other letters.

4.3. Results

Our model yields a marginal R^2 of 0.03 and a conditional R^2 of 0.77 (using the theoretical method, see Nakagawa et al., 2017), so while our fixed effects only account for very little variance, our random effects exhibit a much larger impact.

We assessed the significance of our fixed effects using likelihood ratio-tests with single term deletions. For the position that ‹s› takes inside a word, we found only a marginally significant difference between our model and a model without this predictor ($\chi^2(2) = 5.317$, $p = 0.070$); the model's coefficients show a significant contrast between ‹s› in word medial and final position only, in that medial positions are more likely to be occupied by block letters (β = -1.09, SE = 0.39, z = -2.81, $p = 0.005$).

For the position that ‹s› takes within morphemes, we did find a significant effect ($\chi^2(3) = 12.883$, $p = 0.005$), with one highly significant contrast between morpheme initial and medial position, where block letter ‹s› is more likely to occur (β = -0.98, SE = 0.28, z = -3.46, $p < 0.001$). These positional results go against our expectation that we would find more realizations of cursive ‹s› in medial positions as connections to adjacent letters would not be impeded by morpheme boundaries. One fact that our model does not account for is that instances of ‹s› in word initial position always coincide with morpheme initial position (yet not vice versa), which possibly confounds this finding. However, looking into our data, we find that there are

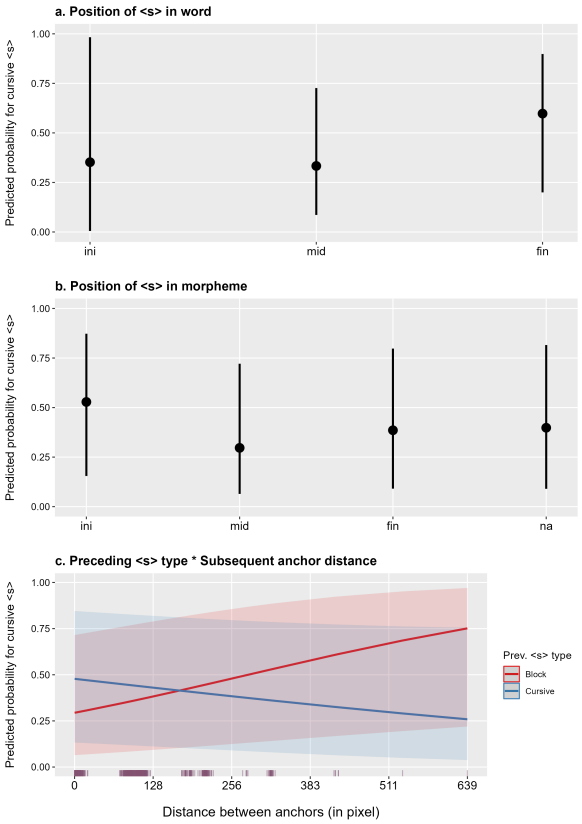


Figure 4: Visualization of the significant fixed effects in the model. For panel c. the scaled anchor distances used during model calculation were transformed back to their original scale. The rugs at the bottom of panel c. display the distribution of anchor distances in the data.

actually more cases of block letter ‹s› in word initial position (349) than there are of cursive ‹s› (299). Taking these out of the equation would therefore rather strengthen the effect of morpheme position that we found in our model.

As for the effects involving geographical information, we did find that the interaction involving previous ‹s› type and the vertical distance between subsequent anchors made a significant difference ($\chi^2(1) = 7.226$, $p = 0.007$); the vertical deviation from the baseline yielded no significant effect ($\chi^2(1) = 0.224$, $p = 0.640$). Figure 4 visualizes the significant fixed effects in our model, exhibiting highly overlapping confidence intervals in each panel, indicating that even significant effects in our model are rather small. Looking at panel c. specifically, which depicts the interaction in our model, we can see that this effect manifests as expected: at the closest vertical proximity, i.e., when it was written on the same baseline, a cursive ‹s› is more likely to follow another cursive ‹s›, yet this effect diminishes the farther apart these instances of ‹s› are from each other. Note that most subsequent anchors have distances (in pixels) that are either

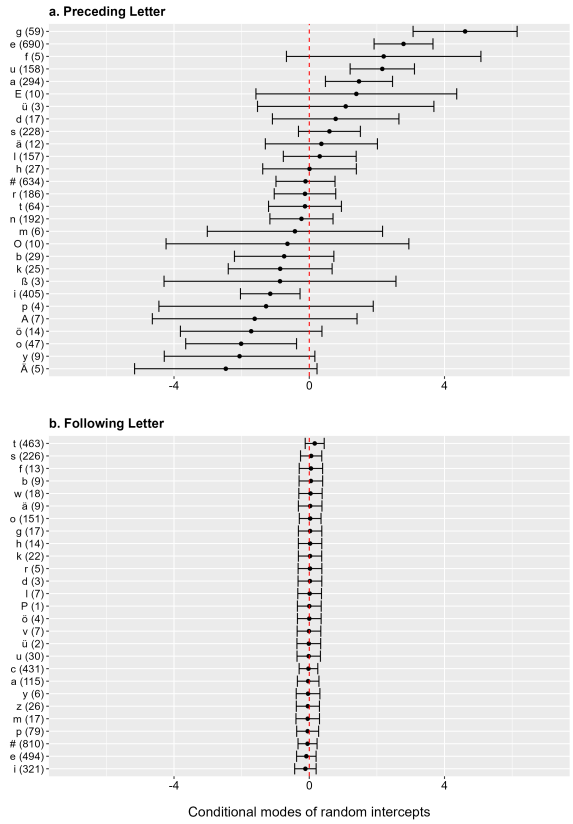


Figure 5: Conditional modes for letters that (a.) precede or (b.) follow <s>. Values that deviate to the right (i.e., positive) side of 0 indicate a higher probability for <s> to be realized in cursive script; values that deviate to the left indicate a higher probability for <s> to be realized in block letters. The number in parentheses after each letter informs of its frequency of occurrence in the data; # signifies that <s> sits at a word boundary.

close to 0 (corresponding to the same baseline) or 100 (one line apart).

For the random effects, we report the variance calculated for each random intercept: 3.058 for individual words; 2.333 for individual students; 5.045 for preceding letters; and 0.032 for following letters. Evidently, preceding letters are associated with the largest variance in the outcome, i.e., whether <s> is realized in cursive or block letter script. In contrast, following letters cause little to no difference in the outcome. Figure 5 illustrates this contrast very clearly, where some preceding letters deviate strongly from 0, whereas following letters stay close to the center of the plot, with overlapping error bars.

Panel a. of Figure 5 suggests that the influence of preceding letters may be more pronounced for instances of cursive <s> than for block letter <s>, as <g>, <e>, <u> and <a> especially deviate relatively far from 0 while also having rather narrow error bars. What is interesting about these four letters in particular is that in handwriting, they often end on an upward-right stroke which could connect to cursive <s> in one fluent motion, as its starting stroke goes in the same direction. Kandel and Spinelli (2010) have shown that complex graphemes (i.e., those consisting of multiple letters) modulate motor processes during handwritten language production, so maybe what we observe here is that some individuals anticipate the potential to seamlessly connect <g>, <e>, <u> or <a> with cursive <s>, therefore processing them as chunks much akin to complex graphemes. Note, however, that our model currently does not include information on whether there actually is a connection between a preceding letter and <s>. So while a preceding <g>, for example, is generally associated with cursive <s>, this does not mean that there necessarily is a connection between these two letters. Our reading of this effect should therefore be regarded as more of a starting point to further inquiry. Nevertheless, what we found is consistent with the idea of *motor anticipation* (Kandel and Perret, 2014) and other accounts which state that context has a role to play during motor production of handwriting (e.g., Meletis, 2020; Reinken, 2023). While in here, we investigated this by means of random intercepts, in an updated model, certain letters could be grouped based on specific features of their form, for example, whether they end on a stroke that could connect to a following letter. This could then enter the model as a fixed instead of a random effect. To retrieve (and annotate) such detailed descriptions, we could of course make use of the functionality that HAnnoI offers, extending our data to our needs.

5. Conclusion and outlook

In this paper, we presented a method to gather and annotate visual (linguistic) data using HAnnoI. We introduced in detail the tool's functionality as well as illustrated possible (and concrete) use cases. As HAnnoI's source code is publicly available, users may decide to extend its functionality to their own desire. From the perspective of the authors of this paper, HAnnoI's development is also still ongoing, with plans to address some of its current limitations (which we will outline below). Nevertheless, we already regard HAnnoI as a fully operational annotation and, more broadly, data collection tool, and we suggest its usage to scientists coming from many areas, including but not limited to research of handwriting.

Limitations

One current limitation of HAnnoI is that it only allows for placing rectangular ROI frames within an image. Other shapes are not possible at the moment. Of course, in HAnnoI, the main purpose of an

ROI is to connect an annotation with its visual reference. In contrast to some of the tools discussed in Section 1.1, HAnnoI was not designed to generate training data for machine learning tasks such as object detection, where polyline shapes would be far more beneficial. Still, even in linguistic research, being limited to rectangles might pose a serious hindrance. This would be the case, for example, when the script under investigations exhibits units that cannot be appropriately framed using rectangles.

Also, while not necessarily a limitation imposed by the rectangular shapes, the fact that the bottom line of an ROI is used to set an anchor, and said anchor is displayed by a horizontal line, utilization of this specific feature does favor situations in which vertical relations between the ROI and its anchor are of interest (such as in our case). This is not to say that anchors can only be used in this way, its positioning is completely up to the user, but it is certainly more convenient to use it in a similar manner as was shown in this paper.

HAnnoI is also limited in that the custom annotation layers are restricted to text input fields. While this is overcome somewhat by the Annotation Mode (and color coding) for categorical variables, additional options like drop down menus or check boxes are currently lacking.

Acknowledgments

We thank Lara-Celine Heikens, Melissa Jahn, Marco Sund, Jiabin Yao and CB for testing the program in the annotation task reported in Section 3, providing valuable data and feedback along the way. We are also grateful to two anonymous reviewers for helpful comments. The research reported here was funded by the German Research Foundation (DFG), project number 533131836.

Bibliographical References

Kristian Berg, Stefan Hartmann, and Daniel Claeser. 2023. Are some morphological units more prone to spelling variation than others? a case study using spontaneous handwritten data. *Morphology*, 34.

Kristian Berg, Jonas Romstadt, and Cedrek Neitzert. 2021. Graphvar–korpusaufbau und annotation version 1.0. *Online-Dokumentation: https://graphvar. uni-bonn. de/dokumentation*.

Hilde De Weerdt, Rainer Simon, Dawn Zhuang, Sunkyu Lee, Iva Stojević, Meret Meister, and Xi Wangzhi. 2024. IMMARKUS: Image annotation in X-MARKUS. Web application.

Abhishek Dutta and Andrew Zisserman. 2019. The VIA Annotation Software for Images, Audio and Video. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2276–2279, Nice France. ACM.

Sonia Kandel and Cyril Perret. 2014. How do movements to produce letters become automatic during writing acquisition ? investigating the development of motor anticipation. *International Journal of Behavioral Development*, 39:1–8.

Sonia Kandel and Elsa Spinelli. 2010. Processing complex graphemes in handwriting production. *Memory & Cognition*, 38(6):762–770.

Sonia Kandel, Elsa Spinelli, Annie Tremblay, Helena Guerassimovitch, and Carlos J. Álvarez. 2012. Processing prefixes and suffixes in handwriting production. *Acta Psychologica*, 140(3):187–195.

Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych. 2018. The inception platform: Machine-assisted and knowledge-oriented interactive annotation. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 5–9. Association for Computational Linguistics. Event Title: The 27th International Conference on Computational Linguistics (COLING 2018).

Dimitrios Meletis. 2020. *The Nature of Writing. A Theory of Grapholinguistics*, volume 3.

Jamshed Memon, Maira Sami, and Rizwan Ahmed Khan. 2020. Handwritten optical character recognition (OCR): A comprehensive systematic literature review (SLR). *CoRR*, abs/2001.00139.

Shinichi Nakagawa, Paul C. D. Johnson, and Holger Schielzeth. 2017. The coefficient of determination r2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. *Journal of The Royal Society Interface*, 14(134):20170213.

Hiroki Nakayama, Takahiro Kubo, Junya Kamura, Yasufumi Taniguchi, and Xu Liang. 2018. doccano: Text annotation tool for human. Software available from https://github.com/doccano/doccano.

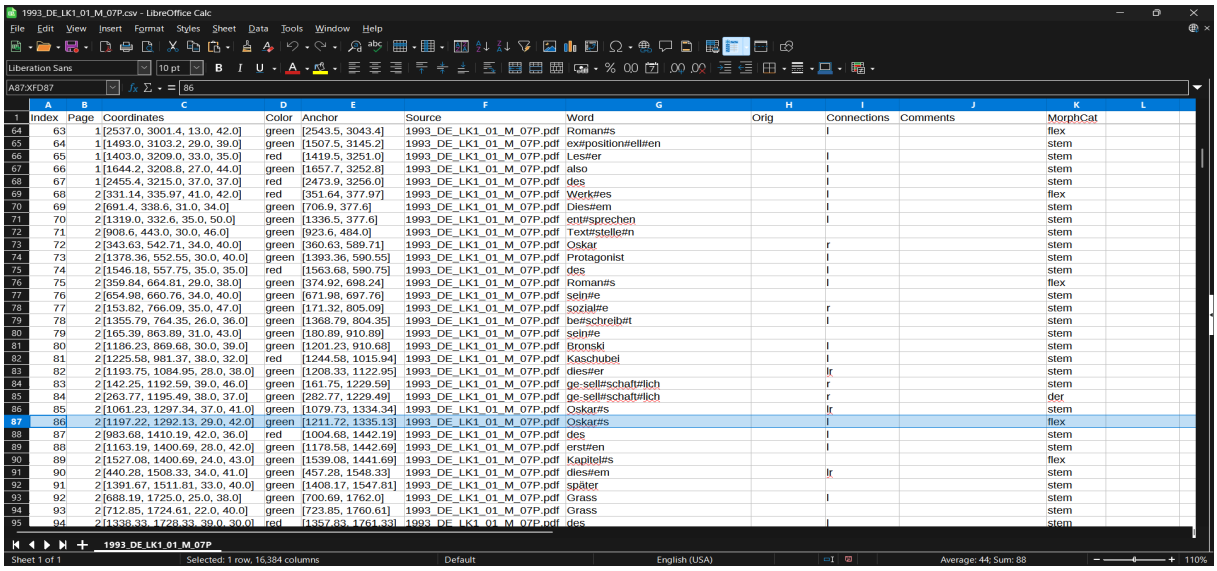
Simon Petitjean, Kristian Berg, Joshua Wieler, Henriette Huber, and Stefan Hartmann. n.d. Are handwritten letter forms shaped by morphology? Manuscript in preparation.

Niklas Reinken. 2023. *Die Grammatik der Handschriften*.

Piotr Skalski. 2019. Make Sense. https://github.com/SkalskiP/make-sense/.

Han Sloetjes and Peter Wittenburg. 2008. Annotation by category – elan and iso dcr. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, Nijmegen. Cited via ELAN software page.

A. Exemplary HAnnoI export files



Index	Page	Coordinates	Color	Anchor	Source	Word	Orig	Connections	Comments	MorphCat
63	1	[2537.0, 3001.4, 13.0, 42.0]	green	[2543.5, 3043.4]	1993_DE_LK1_01_M_07P.pdf	Roman#s		l		flex
64	1	[1493.0, 3103.2, 29.0, 39.0]	green	[1507.5, 3145.2]	1993_DE_LK1_01_M_07P.pdf	ex#position#ell#en				stem
65	1	[1403.0, 3209.0, 33.0, 35.0]	red	[1419.5, 3251.0]	1993_DE_LK1_01_M_07P.pdf	Les#er		l		stem
66	1	[1644.2, 3208.8, 27.0, 44.0]	green	[1657.7, 3252.8]	1993_DE_LK1_01_M_07P.pdf	also		l		stem
67	1	[2455.4, 3215.0, 37.0, 37.0]	red	[2473.9, 3256.0]	1993_DE_LK1_01_M_07P.pdf	des		l		stem
68	2	[331.14, 335.97, 41.0, 42.0]	red	[351.64, 377.97]	1993_DE_LK1_01_M_07P.pdf	Werk#es		l		flex
69	2	[691.4, 338.6, 31.0, 34.0]	green	[706.9, 377.6]	1993_DE_LK1_01_M_07P.pdf	Dies#em		l		stem
70	2	[1319.0, 332.6, 35.0, 50.0]	green	[1336.5, 377.6]	1993_DE_LK1_01_M_07P.pdf	ent#sprechen		l		stem
71	2	[908.6, 443.0, 30.0, 46.0]	green	[923.6, 484.0]	1993_DE_LK1_01_M_07P.pdf	Text#stelle#n				stem
72	2	[343.63, 542.71, 34.0, 40.0]	green	[360.63, 589.71]	1993_DE_LK1_01_M_07P.pdf	Oskar		r		stem
73	2	[1378.36, 552.55, 30.0, 40.0]	green	[1393.36, 590.55]	1993_DE_LK1_01_M_07P.pdf	Protagonist		l		stem
74	2	[1546.18, 557.75, 35.0, 35.0]	red	[1563.68, 590.75]	1993_DE_LK1_01_M_07P.pdf	des		l		stem
75	2	[359.84, 664.81, 29.0, 38.0]	green	[374.92, 698.24]	1993_DE_LK1_01_M_07P.pdf	Roman#s		l		flex
76	2	[654.98, 660.76, 34.0, 40.0]	green	[671.98, 697.76]	1993_DE_LK1_01_M_07P.pdf	seln#e				stem
77	2	[153.82, 766.09, 35.0, 47.0]	green	[171.32, 805.09]	1993_DE_LK1_01_M_07P.pdf	sozial#e		r		stem
78	2	[1355.79, 764.35, 26.0, 36.0]	green	[1368.79, 804.35]	1993_DE_LK1_01_M_07P.pdf	be#schreib#t		l		stem
79	2	[165.39, 863.89, 31.0, 43.0]	green	[180.89, 910.89]	1993_DE_LK1_01_M_07P.pdf	seln#e				stem
80	2	[1186.23, 869.68, 30.0, 39.0]	green	[1201.23, 910.68]	1993_DE_LK1_01_M_07P.pdf	Bronski		l		stem
81	2	[1225.58, 981.37, 38.0, 32.0]	red	[1244.58, 1015.94]	1993_DE_LK1_01_M_07P.pdf	Kaschubei		l		stem
82	2	[1193.75, 1084.95, 28.0, 38.0]	green	[1208.33, 1122.95]	1993_DE_LK1_01_M_07P.pdf	dies#er		lr		stem
83	2	[142.25, 1192.59, 39.0, 46.0]	green	[161.75, 1229.59]	1993_DE_LK1_01_M_07P.pdf	ge-sell#schaft#lich		r		stem
84	2	[263.77, 1195.49, 38.0, 37.0]	green	[282.77, 1229.49]	1993_DE_LK1_01_M_07P.pdf	ge-sell#schaft#lich		r		der
85	2	[1061.23, 1297.34, 37.0, 41.0]	green	[1079.73, 1334.34]	1993_DE_LK1_01_M_07P.pdf	Oskar#s		lr		stem
86	2	[1197.22, 1292.13, 29.0, 42.0]	green	[1211.72, 1335.13]	1993_DE_LK1_01_M_07P.pdf	Oskar#s		l		flex
87	2	[983.68, 1410.19, 42.0, 36.0]	red	[1004.68, 1442.19]	1993_DE_LK1_01_M_07P.pdf	des		l		stem
88	2	[1163.19, 1400.69, 28.0, 42.0]	green	[1178.58, 1442.69]	1993_DE_LK1_01_M_07P.pdf	erst#en		l		stem
89	2	[1527.08, 1400.69, 24.0, 43.0]	green	[1539.08, 1441.69]	1993_DE_LK1_01_M_07P.pdf	Kapitel#s				flex
90	2	[440.28, 1508.33, 34.0, 41.0]	green	[457.28, 1548.33]	1993_DE_LK1_01_M_07P.pdf	dies#em		lr		stem
91	2	[1391.67, 1511.81, 33.0, 40.0]	green	[1408.17, 1547.81]	1993_DE_LK1_01_M_07P.pdf	später				stem
92	2	[688.19, 1725.0, 25.0, 38.0]	green	[700.69, 1762.0]	1993_DE_LK1_01_M_07P.pdf	Grass		l		stem
93	2	[712.85, 1724.61, 22.0, 40.0]	green	[723.85, 1760.61]	1993_DE_LK1_01_M_07P.pdf	Grass				stem
94	2	[1338.33, 1728.33, 39.0, 30.0]	red	[1357.83, 1761.33]	1993_DE_LK1_01_M_07P.pdf	des		l		stem

Figure A1: Example of the CSV export that HAnnoI produces. The file shown here is the same that is loaded into the tool in Figure 1, highlighting the ROI selected in that figure.

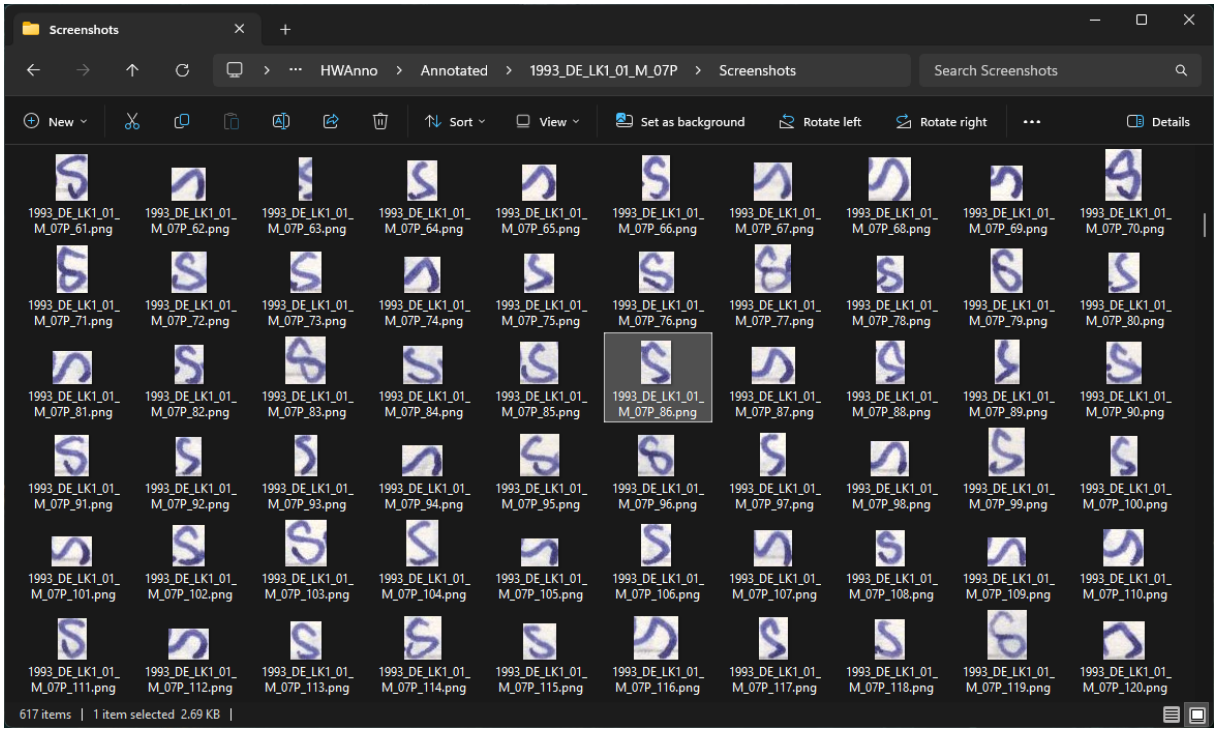


Figure A2: Folder containing PNG image files for single ROIs that have been annotated and rendered using HAnnoI. The image in selection corresponds to the ROI/row selected in Figures 1 and A1, easily identifiable via the shared index (86).

SoriGraph: A New Database of Visual Feature-Level Descriptions of Written Korean

Wednesday Bushong*, Hala Habahbeh*, Ryan Jiang†, Yoolim Kim†
*Wellesley College, Wellesley, MA, USA
{wb104, hh108}@wellesley.edu
†Carleton College, Northfield, MN, USA
{jiangr, ykim6}@carleton.edu

Abstract

Phoneticians and phonologists have developed featural systems that enable systematic description of human speech sounds. However, no such systems exist for describing the visual features of writing systems. It is critical to understand the features of writing systems given their central role in many language users' everyday experience. Just as phonetic and phonological features provide insight into speech perception, visual features can play a similar role for studying reading. In this paper, we introduce SoriGraph, a database of visual feature descriptions and IPA transcriptions for the full lexicon of Korean, drawing on a recent large-scale study of the visual features of writing systems. This database enables analysis of the visual and phonological properties of Korean and will be a critical resource for researchers. We describe the construction of the database and provide an overview of several potential uses of the database, and demonstrate one potential usage (information-theoretic analysis of lexicon structure).

Keywords: Korean, Hangul, visual features

1. Introduction

1.1. Korean and Hangul

For this project, we focus on Korean and its writing system, Hangul.[1] Typologically, Korean is an agglutinative, head-final language with rich verbal morphology. In terms of its sound system, Korean has altogether nineteen consonants and ten vowels.[2] Hangul is system wherein the individual graphemes represent units of sound in the language. The graphemes are assembled into blocks, as shown in Figure 1, which illustrates two blocks (left) and their constituent graphemes, or *jamo*, assembled in linear and non-linear fashion (right). There are 40 unique *jamo* characters.

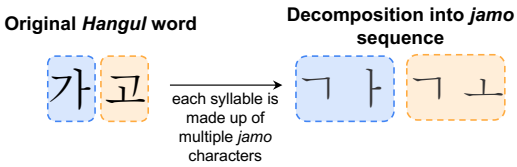


Figure 1: Each Hangul character represents a spoken syllable and is composed of *jamo* characters corresponding to individual (or some biphone sequences of) sounds. This figure shows the original Hangul form of 가고 (Romanized: *gago*) on the left, and its linearized *jamo* sequence on the right.

1.2. Describing the Visual Features of Writing Systems

Phoneticians and phonologists have developed systematic descriptions of the acoustic and articulatory features of speech. Relatively fewer analogous systems exist for describing the visual features of written language, which often focuses on either a limited subset of writing systems (e.g., Primus, 2004), or on specific visual properties (e.g., topology; Changizi and Shimoji, 2005). *Glyph* is a recent study which aimed to develop systematic visual descriptions of 43 writing systems, including Hangul, by crowdsourcing visual feature descriptions from a large number of participants via a gamified task (Kim et al., 2025). Participants' task was to divide the inventory of *jamo* characters into two sets according to a visual rule they specified. Rules were then validated by retesting participants on their own rules. After collecting many such validated rules, Kim et al., 2025 found the minimal set of visual feature rules that allows for efficient and unique identification of each *jamo* character, which resulted in 19 rules. Table 1 shows each visual feature rule, along with the *jamo* characters that satisfy the rule and the original participant description of the visual feature.

Given the crowdsourcing nature of the project, while some visual feature rules map clearly to visual properties, some are more opaque. For example, Rule 1 ("two of something") and Rule 2

[1]We acknowledge the vibrant debate on classifying Hangul as a system; here, we present Hangul generally as a system that encodes the sounds of the language, which we believe is sufficient for the purposes of our present research.

[2]The number of phonemes may vary depending on how semi-vowels and diphthongs are considered, but for our purposes, we consider the phonemic inventory to comprise twenty-nine phonemes.

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 45–49
12 May, 2026.

Rule	*jamo* characters	Original participant description
Rule 1	ㄲㄸㅃㅆㅉ	“two of something”
Rule 2	ㅏㅐㅑㅒㅓㅔㅕㅖㅘㅙㅚㅝㅞㅟㅢㅣ	“Allong”
Rule 3	ㅅㅆㅈㅉㅊ	“man symbol”
Rule 4	ㅅㅆㅇㅈㅉㅊㅎㅡㅢㅣ	“does not contain a right angle”
Rule 5	ㄲㄸㅃㅆㅎㅔㅖㅘㅙㅚㅝㅞㅟㅢ	“Buchstaben, die aus mindestens zwei voneinander getrennten Elementen bestehen”
Rule 6	ㅁㅂㅃㅇㅍㅎㅒ	“Characters with enclosed spaces.”
Rule 7	ㅗㅘㅙㅚ	“Right way up boat”
Rule 8	ㅁㅂㅃㅇㅎㅒ	“AT LEAST ONE ENCLOSED SPACE”
Rule 9	ㅑㅒㅕㅖㅛㅠㅡ	“All characters with a line and two touching tiny rectangles”
Rule 10	ㄱㄴㅈㅉㅊㅎㅗㅘㅙㅚㅛㅜㅝㅞㅟㅠㅡㅢ	“one long horizontal line”
Rule 11	ㅒㅖㅔㅖㅙㅞ	“more than one fullsized vertical line”
Rule 12	ㄱㄲㄷㄸㅂㅃㅅㅆㅈㅉ	“Lettres doubles”
Rule 13	ㅏㅑㅓㅔㅕㅖㅗㅘㅛㅜㅝㅠㅡ	“Halbe Leiter nur gerade”
Rule 14	ㄷㅋㅐㅑㅔㅕㅚㅛㅟㅠ	“Contains three straight lines”
Rule 15	ㄱㄲㅋㅓㅔㅕㅖㅙㅞ	“faces left”
Rule 16	ㄱㄴㄷㄹㅂㅡㅣ	“2 ends”
Rule 17	ㅏㅐㅑㅒㅓㅔㅕㅖㅗㅘㅙㅚㅛㅜㅝㅞㅟㅠㅡㅢ	“petit tiret”
Rule 18	ㄴㄷㄸㅍㅣ	“could pass for a roman numeral”
Rule 19	ㄱㄲㅋㅓㅔㅕㅖㅙㅞ	“a crossbar, with stuff only below it”

Table 1: Visual rules for *jamo* characters from the *Glyph* project (Kim et al., 2025).

(“Allong”) have easily described visual correlates: symmetry (Rule 1) and long vertical stroke (Rule 2). But Rule 7 (“Right way up boat”) and Rule 18 (“could pass for a roman numeral”) have no clear basic features and seem instead to draw on holistic combinations of visual features or analogy to other systems.

1.3. The Present Study

The goal of the present project was to construct a comprehensive database of the Korean lexicon that integrates orthographic, phonological, visual feature, and word frequency information into a single unified resource.

This database, which we dub SoriGraph(a combination of the word for sound in Korean ’sori’, and ’graph’), enables analyses of written Korean that were previously impossible—for example, estimating visual similarity between words beyond rough-grained measures like orthographic similarity, or investigating the relationship between the visual features of a *jamo* character and the phonological features of the sound associated with the character. We hope this database will be a useful resource for researchers interested in the visual structure of written Korean, or the relationship between spoken and written Korean. The database is available at the following link: `https://osf.io/fsb6c`. We provide the database in two file formats. One is a .RDS file readable using the programming language R (R Core Team, 2025). We chose this format because one variable in the database contains matrix entries (the visual feature descriptions of full words) which are difficult to represent in ‘flat’ comma- or tab-separated file formats. We also include the database as a comma-separated value (.csv) format excluding the visual feature matrices, along with the original visual rule-to-*jamo* mapping as another .csv file, so that researchers can compute the word feature matrices themselves.

2. Database Creation

The database includes 282,348 unique word forms. Each word contains the following information: (1) its written Hangul form; (2) its IPA transcription; (3) a decomposition of its Hangul form into a linear sequence of *jamo* characters of length n; (4) an $n \times 19$ matrix containing the visual feature encoding of each *jamo* character; and (5) word frequency information (available for a subset of the words). The sections below describe each step of this process in more detail.

2.1. Dictionary and Preprocessing

We exported the entirety of 표준국어대사전 (*Standard Korean Dictionary*), a comprehensive dictionary of standard Korean (국립국어원 [National Institute of the Korean Language]). This resulted in 436,142 entries. The dictionary includes non-word entries such as affixes, particles, interjections, and multiword expressions, which we removed. We

also removed any entries containing non-Hangul orthographic elements (e.g., *hanja*, Chinese characters). The dictionary also includes separate entries for each meaning of homonyms and polysemous words; we collapsed these distinctions so that every unique written form corresponded to only one entry. We then stripped any special characters from the word form representations (e.g., some entries contained dashes marking morpheme boundaries or parentheticals with further indexing information). This left us with 282,348 unique entries in the database.

2.2. IPA Transcriptions

For IPA transcription, we utilized the Cross-Linguistic Phonological Frequencies (XPF) corpus (Cohen Priva et al., 2021), a resource that provides standardized orthography-to-IPA transcriptions for a wide variety of languages, including Korean. Our procedure began by extracting and identifying all unique syllables (i.e., Hangul characters) present in our database and retrieving their corresponding phonemic transcriptions from the XPF corpus. To avoid issues in rendering across devices, we converted each IPA character to its Unicode value (e.g., ŋ is represented as "U+014B"). Complex IPA characters are represented as a concatenated string of their component Unicode characters (e.g., t^{h} is represented as "U+0074U+02B0"). For each word entry in the database, we use these mappings to construct a character string of Unicode IPA characters (separated by spaces). One limitation to be aware of is that the XPF corpus does not account for systematic sound changes that cross syllable boundaries, instead assuming a fixed syllable-to-IPA mapping. Thus, not all word transcriptions are an accurate reflection of their spoken realization.

2.3. Hangul-*Jamo* Decomposition

As described above (section 1.1), Hangul is an writing system in which individual phonemic units, including consonants, vowels, and certain biphone or diphthong sequences, are represented by *jamo* characters that combine to form orthographic syllable blocks (Taylor, 1980). Because the syllable block serves as the primary orthographic unit, while the information necessary for visual feature analysis resides at the *jamo* level, we decomposed each word form in our dataset into its constituent syllables and then further segmented them into a linear *jamo* character sequence using the `unicodedata` library in Python.[3] Like the IPA transcriptions, we represented each *jamo* character according to its Unicode representation (e.g., ㄱ [Romanized as *g*] corresponds to "U+1100").

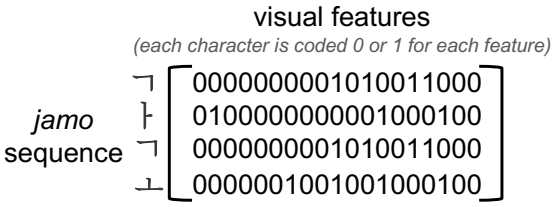


Figure 2: We describe each lexical entry using a matrix with the rows representing each *jamo* character in sequence, and the columns with the binary visual feature vector describing the *jamo* character.

2.4. Visual Feature Matrices

We represent each *jamo* character as a binary vector of length 19. These correspond to the 19 visual feature rules from the *Glyph* project of Kim et al. 2025, that allow for each *jamo* character to be uniquely identifiable.[4] A value of `0` indicates that the character does not satisfy the feature (and vice versa for a value of `1`). We repeat this process for every *jamo* character in the word, resulting in an $n \times 19$ matrix, where *n* is the word length in *jamo* characters. Figure 2 shows an example for the word 가고 (Romanized: *gago*).

2.5. Word Frequency

We obtained frequency measures from an analysis of the Sejong National Corpus (ELRA) conducted by Lee et al. 2016. While Lee et al. 2016 contains a list of only the 5,000 most frequent Korean words, the authors generously provided us with comprehensive frequency estimates of all words occurring in the corpus at least once. The Sejong National Corpus contains both spoken and written language; the frequency estimates Lee et al. 2016 provide are (1) the total number of occurrences of the word form in the corpus, and (2) the average of the spoken and written frequencies of the word form (per 100,000 words). Since our database does not include separate entries for homonyms or polysemous words, we sum the frequencies of all entries associated with each unique word form. This resulted a total of 145,454 entries (51.5%) in our database containing word frequency information.

[3] Unicode metadata includes the linear decomposition of *jamo* characters which make up each unique Hangul character.

[4] That is, we did not identify visual features from the raw data of Kim et al. 2025, but used their identified set of 19 visual feature rules for each *jamo* character.

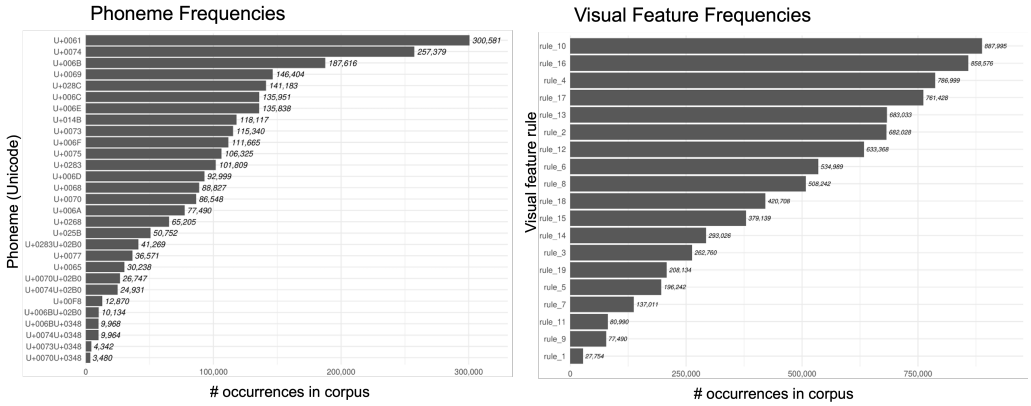


Figure 3: Raw frequency of occurrence of each phoneme (left panel) and visual feature (right panel) in the database.

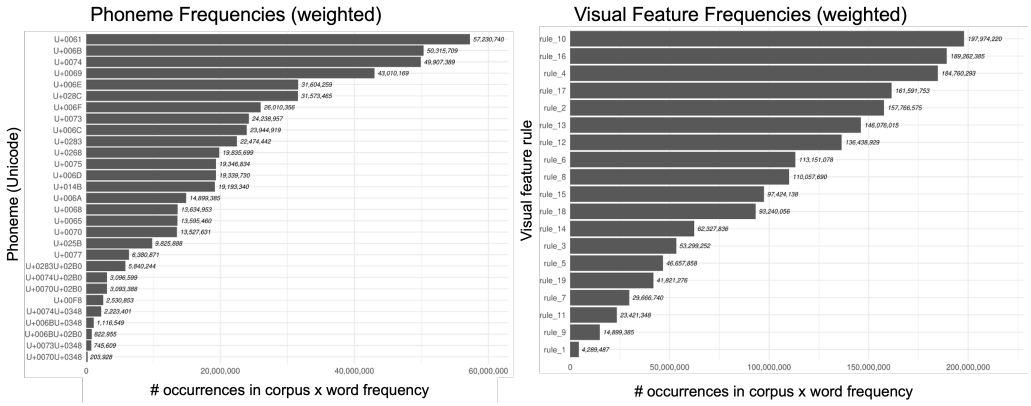


Figure 4: Frequency of occurrence, weighted by word frequency, of each phoneme (left panel) and visual feature (right panel) in the database.

3. Possible Uses of the Database

There are many ways SoriGraph can be utilized by researchers. We briefly describe two possibilities below.

3.1. Information-Theoretic Analysis of Language Structure

Cross-linguistically, the units of spoken language tend to follow similar distributions. For example, word frequencies follow a Zipfian distribution, reducing the entropy of the lexicon and allowing for efficient communication (Zipf, 1949). Phoneme frequencies also follow closely related distributions (Macklin-Cordes and Round, 2020; Tambovtsev and Martindale, 2007).

A natural question, then, is whether corresponding written language units follow a similar pattern. Korean provides a unique opportunity for testing this question, given that the explicit design intent of Hangul was to mirror the spoken language (Sohn, 1999). Is this reflected in the distribution of visual features? We conducted a simple analysis to begin to test this question. Figure 3 shows the frequency of each phoneme (left panel) and visual feature (right panel) across our database, and Figure 4 shows these values weighted by the frequency of the words they occur in. Upon first inspection, while the phonemes of the language appear to show a characteristic (sub-)Zipfian distribution, the visual features seem to be distributed differently (the relationship between frequency and frequency rank is closer to linear rather than Zipfian). This is not meant to be taken as a strong claim, but rather a demonstration that such questions can be asked using SoriGraph. Using SoriGraph, future research can investigate this relationship further and explore its potential implications for written language production and processing.

The database will also allow for other kinds of information-theoretic comparisons between the written and spoken language; for example, examination of which visual features vs. phonemes carry the greatest functional load in the lexicon, and whether these factors may influence diachronic changes.

3.2. Stimulus Design for Reading Experiments

Orthography is known to be an important factor in reading. Access to the visual feature structure of words will allow researchers to compute new measures that can be used to design future reading experiments. For example, our database can support the development of new measures of visual feature neighborhood that extend beyond relatively rough-grained measures like orthographic neighborhood density (OND).

4. Bibliographical References

Mark A. Changizi and Shinsuke Shimoji. 2005. Character complexity and redundancy in writing systems over human history. *Proceedings of the Royal Society B: Biological Sciences*, 272:267–275.

Uriel Cohen Priva, Emily Strand, Shiying Yang, William Mizgerd, Abigail Creighton, Justin Bai, Rebecca Mathew, Allison Shao, Jordan Schuster, and Daniela Wiepert. 2021. *The Cross-linguistic Phonological Frequencies (XPF) Corpus manual*. Accessible online, https://cohenpr-xpf.github.io/XPF/manual/xpf_manual.pdf.

ELRA. Sejong Corpus. http://universal.elra.info/product_info.php?cPath=42_43&products_id=1975.

Yoolim Kim, Marc Allassonnière-Tang, Helena Miton, and Olivier Morin. 2025. The phonology of letter shapes: Feature economy and informativeness in 43 writing systems. *Journal of Memory and Language*, 142:104620.

Sun-Hee Lee, Seok Bae Jang, and Sang Kyu Seo. 2016. A frequency dictionary of Korean: Core vocabulary for learners.

Jayden L. Macklin-Cordes and Erich R. Round. 2020. Re-evaluating phoneme frequencies. *Frontiers in Psychology*, 11.

Beatrice Primus. 2004. A featural analysis of the modern roman alphabet. *Written Language Literacy*, 7:235–274.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ho-min Sohn. 1999. *The Korean language*. Cambridge University Press.

Yuri Tambovtsev and Colin Martindale. 2007. Phoneme frequencies follow a Yule distribution. *SKASE Journal of Theoretical Linguistics*, 4(2):1–11.

Insup Taylor. 1980. *The Korean writing system: An alphabet? A syllabary? a logography?*, page 67–82. Springer US.

George Kingsley Zipf. 1949. Human behavior and the principle of least effort.

국립국어원 [National Institute of the Korean Language]. 표준국어대사전 [Standard Korean language dictionary]. https://stdict.korean.go.kr.

Confusable Characters as Endangered Language Markers: The Case of North Caucasus Writing Systems

Alexander Gutkin[◦] **Adrian Benton**[†] **Christo Kirov**[†] **Brian Roark**[‡]
Google Research, [◦]London, UK; [†]New York; [‡]Portland, OR
{agutkin,adbenton,ckirov,roark}@google.com

Abstract

The Abkhaz-Adyghe and Nakh-Daghestanian language families encompass 35 living languages that possess arguably the most complex modern Cyrillic orthographies due to their very sophisticated phonology. The relevant online data displays idiosyncratic patterns among which the use of confusable characters in input methods is the most prevalent. This work studies one such character—letter *palochka*—that is shared by most of the writing systems in question. We investigate whether patterns including variants of this character can act as markers of these languages in large-scale web-crawled data. We use GlotLID, a wide-coverage off-the-shelf language identification (LID) model, to label paragraph-level web text that contains a palochka confusable, and estimate the effect of confusable character normalization on the quality of GlotLID's predictions in 14 supported North Caucasian languages. According to GlotLID, the normalization significantly increases the recall (discovery of new language data) for some languages, while degrading it for others. However, manual evaluation reveals that overall, only 41% of ensuing wins and 46% of losses are accurate due to GlotLID prediction errors. We argue that, despite finding useful signals, higher precision LID approaches tailored to these long-tail languages are needed to improve the quality of mined data.

Keywords: endangered languages, language identification, homoglyphs, data mining, North Caucasus

1. Introduction

Large web-crawled text corpora in the Cyrillic script are becoming increasingly important as large natural language models are extended to cover lower-resource languages (Krsteski et al., 2025; Kuzhuget et al., 2024; Togmanov et al., 2025; Isbarov et al., 2025). For long tail languages, web-crawled sources offer a valuable alternative to traditional hand-curated datasets, which can be challenging to curate due to a scarcity of linguistic expertise and limited access to native speakers (Caswell et al., 2020; Bapna et al., 2022). Additionally, web data may play a particularly important role in the preservation and revitalization of endangered languages with Cyrillic writing systems, which often suffer from a limited online presence (Nikitina et al., 2019; Arkhangelskiy, 2019; Yankovskaya et al., 2023; Rueter et al., 2024). This last point has been emphasized even for the languages with federal republican status within the Russian Federation, such as Buryat, for which the relatively large number of native speakers does not correlate with the amount and quality of online language material (Khilkhanova, 2019; Zhanaev and Połeć, 2019). While sociolinguists often rely on social media data, general web-crawled corpora can provide a wider range of language data from more diverse sources, potentially including underrepresented Cyrillic orthographies.

To reach adequate quality levels of harvested data, aggressive filtering is typically applied, including setting thresholds on language identification (LID) confidence to retain text (Kudugunta et al., 2023; Kargaran et al., 2024). The quality and coverage of the LID models therefore become important factors, influencing the quality of mined text. For long tail languages, LID model performance is often degraded due to the paucity of training data in those languages. Additionally, when identifying minority languages in large web-crawled datasets, the quality of the data itself matters and investigations into the classes of *dataset noise* become of paramount importance (Caswell et al., 2020).

This paper investigates one type of noise in Cyrillic script text that can also potentially serve as a marker for low-resource orthographies: confusable characters. We consider the writing systems from the North Caucasus used to write over thirty languages from the Nakh-Daghestanian and Abkhaz-Adyge language families (Daniel and Lander, 2011; Chumakina, 2017), which we describe in Section 2. In this paper, we choose to focus on arguably the most salient common orthographic feature[1] of these writing systems — the Cyrillic let-

[1]Other letters and their confusable variants may also provide useful signals. For example, standalone *breve* (U+02D8) is found in the PanLex Swadesh list for the endangered Rutul language (Kamholz et al., 2014). Originally an element of Abkhaz orthography it is now obsolete but can be found in web crawls. We defer investigation of this and other such phenomena to future work.

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 50–63
12 May, 2026.

ter *palochka* (U+04C0) and its lower-case variant.[2] This letter was introduced to the Unicode standard in 2006. Prior to that, it was often represented by the capital Latin letter *I* or the digit *1* in keyboards used to type the Cyrillic orthographies of North Caucasus languages, such as Abaza (Koshevoy et al., 2023). As we show in Section 3, several additional confusable characters are actively used as proxies for *palochka*, most likely due to the scarcity of native input methods. These and other similar confusable characters have been cited as adversely affecting the quality of LID and machine translation models for long-tail languages (Caswell et al., 2020; Bapna et al., 2022).

Our experiments in Section 4 investigate the following research questions:

RQ1: How sensitive are modern, wide coverage LID models to the noise introduced by confusable characters? For this purpose, we focus on the GlotLID v3 model, which is trained to identify over 2,000 language-script labels and covers 14 languages belonging to the language families of interest (Kargaran et al., 2023).[3] The answer to this question is not obvious because within this group of 14 vulnerable or endangered languages, we expect a strong *a priori* imbalance in the amount of labeled training data available to the LID model.

RQ2: Can confusable character normalization increase the recall for languages in question? In other words, will orthographic noise reduction lead to the discovery of new data in these languages?

RQ3: How sensitive is LID model precision for endangered North Caucasian languages to confusable character normalization? Can the cases when the model stops predicting one of the languages in question post-normalization be considered a reduction in the false positive rate, or an increasing of the number of false negatives?

Since there is no hand-labeled test set available for the experiments described above, we rely on the metadata of the original web documents where possible, as well as manual curation.

2. Background

Cyrillic Script From the early days in the 1960s, the computing standards for representing Cyrillic character sets have co-evolved with the Latin script standards from initial 7-bit and later 8-bit representations. This has been attributed to the similarity and relative simplicity of the two scripts as well as a considerable degree of cooperation between various international standardization bodies at the time (Ross, 1984; Hopkinson, 1984; Clews, 1988). This gradual evolution led to the first 16-bit versions of the Unicode standard being able to encode major Cyrillic writing systems that included Bulgarian, Belarusian, Macedonian, Russian, Serbian and Ukrainian (Parry, 1991; Clews, 1991; Haralambous, 2007). Beyond these and other well-resourced languages that also include the major languages of Russian Federation and Central Asia, such as Kazakh, Kyrgyz, Mongolian, Tatar and Tajik, there is a rich and linguistically well-explored orthographic landscape of many more lesser-resourced languages utilizing Cyrillic script (Musajev, 1965). Modern estimates place the number of living languages within the Russian Federation alone at somewhere between 150 (Alpatov, 2000) and 155 (Koryakov et al., 2022), the majority of which are in various states of endangerment (Kraeva and Guermanova, 2020; Gruzdeva, 2022). At the same time, the Cyrillic portion of the Unicode standard has been continuously evolving to accommodate some of these orthographies, both living and historic (Miller, 2021; Manulov, 2022; Anderson, 2023), with some writing systems still unaddressed (Roncero, 2021).

Writing Systems of North Caucasus We investigate the languages from Abkhaz-Adyge (Northwest Caucasian, or NWC)[4] and Nakh-Daghestanian (Northeast Caucasian, or NEC)[5] language families. These two language families encompass languages with typologically idiosyncratic features of phonology, morphology and syntax that make them an important subject of linguistic studies, with some researchers calling North Caucasus "Europe's linguistically most exotic area" (Daniel and Lander, 2011). For example, according to Chirikba (2016), the phonemic inventory of Ubykh, a recently extinct NWC language, has close to eighty consonants while only possessing three vowels, while Babaliyeva (2023) mentions 44 grammatical cases in Tabasaran, an NEC language.[6]

There are presently 37 ISO 639-3 language codes allocated to the languages from these two families, shown in Table 1. Five languages are in the NWC family and the remaining languages are in the NEC family. Of these, the NWC Ubykh and NEC Aghwan are now extinct. Despite their endangered status, there is plausibly still non-negligible amounts of text in these languages present on the

[2] The official orthography of Abkhaz is the exception in not employing *palochka* (Hewitt, 2010). However, as we show in Section 4, paragraph-level LID can assign spurious Abkhaz labels to such data.

[3] https://github.com/cisnlp/GlotLID

[4] https://glottolog.org/resource/languoid/id/abkh1242

[5] https://glottolog.org/resource/languoid/id/nakh1245

[6] This traditional view of the case system in NEC languages is contested by Comrie and Polinsky (1998).

Code	Name	Family	Code	Name	Family
abk	Abkhaz	NWC	inh	Ingush	NEC
abq	Abaza	NWC	kap	Bezhta	NEC
ady	Adyghe	NWC	kbd	Kabardian	NWC
agx	Aghul	NEC	khv	Khwarshi	NEC
akv	Akhvakh	NEC	kjj	Khinalugh	NEC
ani	Andi	NEC	kpt	Karata	NEC
aqc	Archi	NEC	kry	Kryz	NEC
ava	Avar	NEC	kva	Bagvalal	NEC
bbl	Bats	NEC	lbe	Lak	NEC
bdk	Budukh	NEC	lez	Lezgian	NEC
bph	Botlikh	NEC	rut	Rutul	NEC
che	Chechen	NEC	tab	Tabasaran	NEC
cji	Chamalal	NEC	tin	Tindi	NEC
dar	Dargwa	NEC	tkr	Tsakhur	NEC
ddo	Tsez	NEC	uby	Ubykh	NWC
gdo	Godoberi	NEC	udi	Udi	NEC
gin	Hinuq	NEC	ugh	Kubachi	NEC
huz	Hunzib	NEC	xag	Aghwan	NEC
...	...	...	xdq	Kajtak	NEC

Table 1: The NEC and NWC languages of interest classified as either *vulnerable* (default), *definitely endangered* (blue), *severely endangered* (orange) or *extinct* (red) by UNESCO (Moseley, 2010).

Web (for example, in the form of lexicographic dictionaries), that either get filtered out by crawlers or misclassified by LID filters, due to poor support of NEC and NWC languages by current LID models.

The NEC and NWC alphabets with official status in the Russian Federation use the Cyrillic script. As we mentioned in the Introduction, the presence of the *palochka* letter is typically indicative of orthographies of these languages, where depending on the language, it modifies the preceding consonant, marking it as ejective or pharyngeal, or serves on its own as a glottal stop (Daniel and Lander, 2011). Since usage as a modifier letter is very common, the *palochka* letter is found in many digraph and trigraph letters of NEC and NWC writing systems that possess very complex consonantal inventories.[7] For example, in Kubachi orthography, the native word for the Kubachi language is written "ГIугъбуган" and the word for "hen" is "гIягIя", upper- and lower-case examples of the standalone digraph "гI" that includes *palochka* and represents the [ʔ] pharyngeal plosive.

The NEC family also includes unwritten languages. This situation is changing since new Cyrillic script-based orthographic proposals for previously unwritten languages have constantly emerged in Dagestan and beyond since the late 1990s (Ataev, 2015). Additional complexities arise when unwritten languages are spoken outside of the predominantly Cyrillic script area. For example, the NEC language Bats (closely related to Chechen and Ingush) is spoken across the border in Georgia and historically has been heavily influenced by the Georgian language of the unrelated Kartvelian family (Gippert, 2008). The proposed Bats writing systems therefore include ones based on Georgian, Cyrillic and even Latin alphabets. Similarly, tentative Latin and Cyrillic script-based orthographies were mentioned for the NEC language Kryz from a Lezgic branch spoken in Azerbaijan (Clifton et al., 2005). In addition to *digraphia*, the written material in such languages may borrow the orthography of a larger language spoken in the area, which further complicates mining for such data.

[7] The Kabardinian Cyrillic orthography includes a single tetragraph "кхъу" representing [qχʷ] sound, while the recently proposed orthography for Archi contains the tetragraph "ххьI" representing [χːˤ] (Chumakina et al., 2007).

Name	Unicode	Visual
CYRILLIC CAPITAL LETTER BU I	U+0406	І
CYRILLIC SMALL LETTER BU I	U+0456	і
DIGIT ONE	U+0031	1
GREEK SMALL LETTER IOTA	U+03B9	ι
LATIN CAPITAL LETTER I	U+0049	I
LATIN SMALL LETTER I	U+0069	i
LATIN SMALL LETTER IOTA	U+0269	ɩ
LATIN SMALL LETTER L	U+006C	l

Table 2: The set of confusable character Unicode code points and corresponding visual form. We abbreviate BYELORUSSIAN-UKRAINIAN as BU to preserve space.

Confusable Characters Due to the visual similarity between some characters of Latin and Cyrillic scripts, these confusable characters often appear in web-crawled data. The security threats due to these characters, or *homoglyphs*, are actively studied in the cybersecurity field (Gabrilovich and Gontmakher, 2002; Holgers et al., 2006; Deng et al., 2020; Wang et al., 2024). More recently, the effects of confusable characters on the performance of large language models (LLMs) have also been investigated in the context of preventing the misuse of AI-generated text (Boucher et al., 2022; Kirchenbauer et al., 2023; Creo and Pudasaini, 2025; Cooper et al., 2025). In the next section we explore the various confusable characters used as a proxy for *palochka* in a large web-crawled corpus, and describe our initial investigation of LID for NEC and NWC languages.

3. Preliminary Glance at the Data

For this preliminary investigation we chose to search for the NEC and NWC languages using *palochka* confusable characters as a marker in MADLAD-400, which is a general domain multilingual dataset based on CommonCrawl that spans 419 languages (Kudugunta et al., 2023). The dataset has two partitions: a 5 trillion token `noisy` dataset, which is the dataset obtained after applying document-level LID but before any filtering, and a 3 trillion token `clean` dataset, which has a wider

variety of filters applied, including some based on the document-level LID. Since we are interested in languages mostly outside the set of MADLAD-400 LID labels, we chose to explore the noisy partition for our experiment. Furthermore, in our analysis in this section we follow a manual paragraph-level LID process for NEC and NWC languages using the source document metadata, online lexicographic dictionaries and similar contextual clues, where possible.

Palochka Variants Table 2 presents the set of eight characters that in our experiments serve as a proxy for *palochka*, lower- and upper-case variants. It consists of diverse characters with varying degrees of visual confusability with the originals. The set was assembled after inspecting various data sources. Two characters in the list (the two variants of *Latin letter I*) belong to the set of genuine homoglyphs maintained by the Unicode Consortium (2025).[8] These characters along with the *Digit 1* letter are still widely found in NEC and NWC language web pages, typically but not exclusively originating from the autonomous republics of the Russian Federation where these languages are spoken. We also included rarer characters such as Greek and Latin variants of letter *Iota* that can be found in electronic lexicographic dictionaries and Swadesh lists for a multitude of NEC and NWC languages such as PanLex (Kamholz et al., 2014).[9]

Basic Method We employ a simple pipeline consisting of two steps: The first step splits MADLAD-400 documents from the noisy set into paragraphs using newline separators. The second step selects the relevant paragraphs. We denote the set of confusable variants of *palochka* described above Y. In order to retrieve paragraphs that *possibly* belong to the languages of interest recorded using input methods that employ elements of Y, we searched for paragraphs that contain at least one token consisting entirely of Cyrillic script letters combined with the elements of Y. Concretely, such tokens should contain at least one sub-string (x_1, y, x_2), $x_i \in X$ and $y \in Y$, where X is the set of all Unicode Cyrillic *lowercase* letters.[10]

Resulting Signal Splitting the noisy MADLAD-400 documents by newline characters and filtering to those with a *palochka* confusable character yields 446,356,076 paragraphs. As we show later in this section, only a small proportion of these paragraphs belong to the NEC/NWC languages. In addition to the low-quality data, such as spam and unnatural text, the filter triggers on valid text in unrelated languages whose orthographies permit tokens that pass the *palochka* filter. In particular, the Ukrainian, Belarusian, Kazakh and Rusyn Cyrillic writing systems because two of the *palochka* variants in Table 2 (the two versions of *Ukrainian-Belarusian Letter I*) are valid characters in their orthography. As we show in Section 4, applying a paragraph-level LID filter prunes out the majority of these original paragraphs resulting in a significantly smaller set of candidate paragraphs.

After removing the paragraphs that belong to the documents with MADLAD-400 language labels corresponding to the four orthographies resulting in false positives described above, we obtained a text sample that *likely* belongs to NEC and NWC languages, but still includes some unrelated languages as well. Some examples of sample sentences in 12 languages from this set are shown in Table 3, where the snippets from the resulting paragraphs with the confusable characters highlighted in red are shown along with the manual language assignment.[11] The two examples shown in the table belonging to Turkic languages—Bashkir and Southern Altai—highlight the false positives in our set. Neither language has the letter *palochka* in its orthography and the confusable characters in these examples are likely artifacts of inadequate input methods. In the Bashkir example, in addition to the actual false positive *palochka* instance (represented as digit one), there are two further non-orthographic characters (digits two and three) that indicate problems with the document encoding. The analysis for this case is provided in Table 4.

Document-level LID MADLAD-400 supports five out of the 35 living NEC and NWC languages shown in Table 1: Adyghe, Avar, Chechen, Kabardian and Tabasaran. For the snippets of NEC and NWC languages in Table 3, considering the manual language assignments in the third column as the truth, we observe that paragraphs belonging to four languages from the above set match the document-level language labels in the sixth column. For paragraphs in languages outside of the supported set, the model tended to assign the document to languages in the same family. So, for instance, Lezgin paragraph gets the Tabasaran label, which is expected as both languages reside in the Lezgic branch of NEC

[8] https://www.unicode.org/Public/security/latest/confusables.txt

[9] https://db.panlex.org/

[10] We ignore uppercase letters for left and right context in this paper. Our filter misses all the uppercase words (such as titles) from NEC and NWC text but at the same time has better precision because *palochka* variants in lowercase Cyrillic context are more likely to occur for languages of interest in general multilingual corpus.

[11] We located the source web documents and identified the languages from the context. For example, the Rutul sentence comes from bilingual Rutul-Russian newspaper https://rutnov.ru/.

Sample Sentence	Char(s)	Manual Language Assignment			Document
		Language	Code	Family	Language
УадыргIвана амартанква гьажьыта расацIла бацала йыршшуа йалагатI	U+0049	Abaza	abq	NWC	Avar
Къэбар гухэкIыбэ къэзыхьырэ уз	U+0049	Adyghe	ady	NWC	Adyghe
ХIаттарис хIуьлмат дагулурай хье арайиъ.	U+0049	Aghul	agx	NEC	Russian
Бу уредуни уренерге болушкан улус коркуш коп оны чотозо, чазын 1етпес. Ончогорго1аан быйан.	U+0031	Southern Altai	alt	Turkic	Southern Altai
Салатавиялъул диалекталъулъ туркази лъарагI рагIаби	U+0049	Avar	ava	NEC	Avar
УЗытыу тыу1ан телдә алып барылған мәктәптәр2ең 2-се класы өсөн баш3орт (дәүләт) теленән эш программаһы.	U+0031	Bashkir	bak	Turkic	Bashkir
Iин чуьрчу боданехь таьIна сайн декхнаш а гуш!	U+0049	Chechen	che	NEC	Chechen
Ва пагьмучерси дарган, мухбир хIед аррукири?	U+0049	Dargwa	dar	NEC	Avar
Берд Юрт яха оагIув укхазахьа хьожаяьй	U+0049	Ingush	inh	NEC	Chechen
ЩIэныгъэлIхэр игъащIэ лъандэрэ топсэлъыхь цIыхумрэ ар къэзыухъуреихь дунеймрэ я псэкIэ	U+0049	Kabardian	kbd	NWC	Kabardian
Гьинтнил кIиришиву махъ лирчIсса, кIинтнил замгьарду байбишин бувасса ссутнил гьантри хас бувара зула ...	U+0049	Lak	lbe	NEC	Avar
Нугъатрин, рахунрин кьет1енвилериз килигна лезги ч1ал пуд наречиедиз пай жезва	U+0031	Lezgin	lez	NEC	Tabasaran
Гьабише Аллагьахьде дилег гьаъара, джва1ршиклаа гигубыр гьаъ, хьур.	U+0031	Rutul	rut	NEC	Avar
ХанаIгьмад чоджий, ЦIаIхни хивын джегьилер.	U+0049	Tsakhur	tkr	NEC	Avar

Table 3: Sample of MADLAD-400 Cyrillic sentences in different orthographies with highlighted *palochka* confusables. The twelve NEC and NWC manually identified languages shown here are classified as either definitely endangered or vulnerable. The MADLAD-400 document language labels are shown in sixth column.

Original text	Corrected text
БАШҒОРТОСТАН РЕСПУБЛИКА*ЫНЫ4М(!АРИФ МИНИСТРЛЫ!Ы	БАШҒОРТОСТАН РЕСПУБЛИКАҺЫНЫҢ МӘҒАРИФ МИНИСТРЛЫҒЫ
МР Й(РМ(К(Й РАЙОНЫНЫ4 М(!АРИФ ИДАРАЛЫ!Ы	МР ЙӘРМӘКӘЙ РАЙОНЫНЫҢ МӘҒАРИФ ИДАРАЛЫҒЫ
ТАРҒКА2Ы АУЫЛЫ УРТА БЕЛЕМ БИРЕҮ М(КТ(БЕ	ТАРҒКАҙЫ АУЫЛЫ УРТА БЕЛЕМ БИРЕҮ МӘКТӘБЕ
Ф(ННИ- ТИКШЕРЕНЕҮЭШЕ	ФӘННИ- ТИКШЕРЕНЕҮЭШЕ
Тарка2ы хал3ы телм9ренд9 фразеологизмдар2ы 3улланыу.	Таркаҙы халҡы телмәрендә фразеологизмдарҙы ҡулланыу.
Номинация “Баш3орт теле 89м 929би9те”	Номинация “Башҡорт теле һәм әҙәбиәте”

Table 4: Snippet of badly encoded Bashkir text from an essay in Microsoft Word*. The corrupted glyphs are highlighted in red on the left-hand side of the table, the hand-corrected versions are highlighted in blue on the right-hand side. *Source: https://nsportal.ru/ap/library/drugoe/2019/01/06/ispolzovanie-frazeologizmov-v-rechi-zhiteley-s-tarkazy.

family, while Ingush gets labeled as Chechen. The behavior of the model, however, is more nuanced because the assignment to the closest language is not always followed. For example, Lak and Dargwa paragraphs get labeled as Avar – all NEC languages, but not closely related. Another example is the paragraph in Tsakhur, which is also labeled as Avar even though one might expect Tabasaran, since it is closer phylogenetically. In a more problematic scenario, some sentences get labels from different language families, e.g., the snippet in Abaza, an NWC language, is labeled as Avar from the NEC family.

Identifying the out-of-domain languages and estimating the amount of available language data is difficult because in the absence of a reliable paragraph-level LID model with enough coverage, the manual process described above is not scalable. In the next section, we investigate improvements to our pipeline involving the incorporation of a paragraph-level LID filter with wider coverage of NEC and NWC languages, and study the behavior of such a paragraph-level model in the presence of *palochka* variants.

4. Experiments and Discussion

The following analysis examines the interplay between the *palochka* variant characters and an LID model. Unlike the prior mostly manual inspection procedure to check for the presence of useful NEC and NWC language signals described in Section 3, our method here uses a wide coverage document- and paragraph-level LID model and examines a significantly bigger corpus.

The Dataset and LID Model For the main body of experiments we use DCAD-2000 (Shen et al., 2025) — a recently introduced large-scale multilingual corpus which is significantly larger than MADLAD-400, contains more recent crawls from CommonCrawl and includes the other multilingual sources such as FineWeb-2 (Penedo et al., 2025) and MaLA (Lin et al., 2024). Crucially, the dataset covers 2,282 document-level language-script LID labels produced by GlotLID (Kargaran et al., 2023),

which is a LID model using fastText architecture (Joulin et al., 2016). The subset of NEC and NWC languages supported by GlotLID consists of 14 Cyrillic writing systems. There are four NWC languages — Abaza (abq), Abkhaz (abk), Adyghe (ady), Kabardian (kbd) — and ten NEC languages — Aghul (agx), Avar (ava), Bezhta (kap), Chechen (che), Dargwa (dar), Ingush (inh), Lak (lbe), Lezgin (lez), Tabasaran (tab) and Tsakhur (tkr). This LID model has the widest NEC and NWC language coverage known to us, and thus we choose it as a default model in our experiments. We use the default GlotLID model, and set $k = 5$ for top-k label extraction with a minimum hypothesis probability of 0.01 in the model decoding hyperparameters.

Similar to experiments in Section 3 where the noisy MADLAD-400 subset was used, below we operate on the unfiltered union of keep and remove DCAD-2000 subsets. We hypothesize that the set of documents discarded by DCAD-2000 filters contains valuable data from NEC and NWC languages. Some of these documents may have been filtered by mistake, as the presence of confusable letters may have caused the removal filters to trigger, but also possibly due to poor LID.

Gauging the Amount of Useful Signal Applying the *palochka* confusables filter from Section 3 to DCAD-2000 data yields 762,639,585 paragraphs of text in various writing systems that contain at least one Cyrillic token with *palochka* variants, which is nearly double the number of paragraphs extracted using the same technique from the MADLAD-400 corpus in Section 3. This set includes 1,574,518 paragraphs where the majority script of that paragraph's constituent tokens is not Cyrillic. We filter out these paragraphs, which include text in 156 unique scripts (including the ISO 15924 script code Zzzz for *uncoded script*). We further filter the resulting set to exclude 15,129,549 short snippets consisting of less than three tokens, since such short sequences adversely affect LID.

According to paragraph-level GlotLID predictions 98.7% of the remaining 745,935,518 paragraphs belong with high confidence to Cyrillic writing systems where the *palochka* filter triggers on valid orthographic tokens. These writing systems are the relatively high-resource Belarusian, Kazakh, Ukrainian and (low-resource) Rusyn orthographies mentioned in Section 3. After excluding these paragraphs, the remaining 9,433,217 paragraphs contain some proportion of paragraphs in NEC and NWC languages that include tokens with *palochka* letter variants. In addition, this set also includes paragraphs in unrelated Cyrillic writing systems that include false positives due to typos, bad data encoding, inadequate input methods or malicious content as demonstrated by the Bashkir example in Table 4. Finally, this also includes some instances of Belarusian, Kazakh, Ukrainian and Rusyn that have been mis-classified as other languages.

The paragraph-level GlotLID predictions for NEC and NWC languages include 1,628,526 paragraphs in 14 languages as shown in the first and second columns of Table 5. Among the remaining 7,799,786 paragraphs in unrelated languages the top five high-frequency entries in the language list are Russian (rus), "Undetermined" (und), Old Russian (orv), Komi-Zyrian (kpv) and Bulgarian (bul) Cyrillic orthographies, where "Undetermined" refers to cases when GlotLID failed to assign a label. The *palochka* letter is not part of the last three writing systems in the list and does not feature in the orthography of modern Russian, but may be present in Russian paragraphs that feature NEC or NWC language code-switching. Additional signal may potentially be found among paragraphs with "Undetermined" label.

Filter Precision and Recall The application of LID after the *palochka* variant filter serves to separate the true positives (1,628,526 paragraphs) from the false positives in unrelated languages (7,799,786 paragraphs), yielding the *palochka* variant filter precision of 17.3%. Computing the corresponding filter recall requires a different pipeline, where LID is applied before the variant filter. This recall-specific pipeline yields a comparable number of true positives and significantly higher number of false negatives resulting in a low recall value of 7.4%.[12] The detailed analysis of the recall pipeline is provided in Appendix A.

Normalization of *Palochka* Variants The normalization pipeline is a straightforward extension of the *palochka* variant filters from Section 3: the input text is split into paragraphs, followed by the search for *palochka* variants in Cyrillic tokens. The additional step required for normalization involves the context-dependent rewrite rule that maps these *palochka* variants y from the set of variants Y (from Table 2) surrounded by lowercase Cyrillic letters to some canonical representation $\hat{y}$, which we set to *Cyrillic Small Letter Palochka* (U+04CF).

The differences in paragraph-level best hypothesis GlotLID performance before and after normalization of *palochka* letter variants are shown in Ta-

[12]The high number of false negatives in this scenario is not surprising. If the LID prediction is indeed correct it is not necessarily the case that the particular NEC/NWC language paragraph in question must necessarily use letter *palochka* or its variants. For example, as we mentioned earlier, there are not *supposed* to be instances of *palochka* in Abkhaz orthography, yet we observe a small number of such instances.

Lang.	Paragraphs		Change (%)	Wins	Losses	Identity	Same Group	Lengths	
	Before	After						μ	σ^2
abk	6008	6340	5.237	1118	930	4210	5078	4.25	50.71
abq	14 326	11 676	−18.498	11	854	11 646	13 472	28.2	741.25
ady	136 143	136 359	0.158	148	112	131 119	136 031	33.09	28 222.37
agx	8008	7462	−6.818	39	112	7258	7896	33.79	1473.21
ava	248 123	257 840	3.769	4261	111	247 990	248 012	33.92	1425.24
che	567 820	564 611	−0.565	539	2497	563 142	565 323	36.72	2330.68
dar	21 023	20 822	−0.956	135	153	20 551	20 870	36.2	989.31
inh	76 985	75 579	−1.826	883	2054	73 856	74 931	108.33	164 566.24
kap	534	167	−68.727	15	21	137	513	6.87	247.65
kbd	396 096	397 616	0.382	2407	736	390 027	395 360	27.51	1043.34
lbe	46 895	47 248	0.747	537	253	46 065	46 642	35.99	1629.85
lez	72 183	70 204	−2.742	304	627	69 415	71 556	41.71	2495.88
tab	33 448	33 754	0.907	255	151	32 722	33 297	40.94	1171.15
tkr	934	884	−5.353	16	21	807	913	26.98	729.09
All	1 628 526	1 630 562	0.125	10 668	8632	1 598 945	1 619 894	37.18	11 927.56

Table 5: Summary of top LID prediction differences for NEC and NWC languages in DCAD-2000 before and after the normalization of *palochka* variants.

Lang.	Size	Before				After				t-test	
	N	μ	σ^2	CI$^-$	CI$^+$	μ	σ^2	CI$^-$	CI$^+$	t-stat	p-val
abk	4210	0.487	0.093	0.478	0.496	0.521	0.093	0.512	0.530	−5.133	2.908e−7
abq	11 646	0.964	0.010	0.962	0.966	0.798	0.061	0.793	0.802	67.500	0.0
ady	131 119	0.759	0.066	0.758	0.761	0.759	0.065	0.757	0.760	0.641	0.521
agx	7258	0.878	0.056	0.872	0.883	0.863	0.064	0.857	0.869	3.571	0.0
ava	247 990	0.961	0.020	0.960	0.961	0.990	0.004	0.989	0.990	−93.649	0.0
che	563 142	0.984	0.008	0.984	0.984	0.981	0.010	0.981	0.981	18.741	2.338e−78
dar	20 551	0.857	0.045	0.854	0.859	0.851	0.046	0.848	0.854	2.491	0.013
inh	73 856	0.869	0.064	0.867	0.871	0.866	0.066	0.864	0.868	2.273	0.023
kap	137	0.544	0.095	0.492	0.596	0.489	0.080	0.441	0.537	1.542	0.124
kbd	390 027	0.935	0.032	0.934	0.935	0.940	0.031	0.940	0.941	−13.123	2.469e−39
lbe	46 065	0.851	0.047	0.849	0.853	0.857	0.044	0.856	0.859	−4.565	4.999e−6
lez	69 415	0.889	0.036	0.887	0.890	0.875	0.042	0.874	0.877	12.792	1.899e−37
tab	32 722	0.929	0.039	0.927	0.932	0.942	0.032	0.940	0.943	−8.313	9.522e−17
tkr	807	0.629	0.101	0.607	0.651	0.636	0.098	0.614	0.657	−0.419	0.675

Table 6: Differences in top LID prediction confidences before and after normalization for the cases when the same language is predicted. The second column indicates the size of the population (N) that corresponds to the seventh column named “Identity” in Table 5. The cases when normalization results in increased prediction confidences are marked in blue, the confidence degradations are marked in red.

ble 5. For each language the number of paragraphs is shown before and after applying the normalization. The percentage increase or decrease in the paragraph count is indicated in the “Change” column. The counts for the cases when the top post-normalization prediction changes from the unrelated language to the language in NEC or NWC families (e.g., from Russian to Chechen) are provided in the “Wins” column. Conversely, the counts of cases when the top pre-normalization prediction switches from an NEC or NWC language to a language from an unrelated family (e.g., from Kabardian to Karachay–Balkar) are indicated in “Losses” column. The column “Identity” provides the counts for the cases when top language prediction is not affected by normalization. The “Same Group” column provides accumulated counts for the cases when both the pre-normalization and post-normalization top hypothesis belongs to either NEC or NWC language family (e.g., Avar changes to Ingush). Finally, the “Lengths” column provides statistics on paragraph lengths.

As can be seen from Table 5, overall the effects of normalization are positive but minor resulting in just 2,036 new sentences for the 14 languages in question. These effects vary by language. According to GlotLID more paragraphs being discovered after the normalization in six languages (Abkhaz, Adyghe, Avar, Kabardian, Lak and Tabasaran), while some paragraphs are lost for another eight (Abaza, Aghul, Bezhta, Chechen, Dargwa, Ingush, Lezgin and Tsakhur). We note that very little data in Bezhta and Tsakhur make it through our filters, which reflects their limited coverage in DCAD-2000.

We next investigate the effect of normalization on prediction confidence for the languages of interest when the best prediction is not changed by normalization for which the relevant counts are shown in column "Identity" of Table 5. We compute the best prediction confidence as a difference between the best and second-best hypothesis. If normalization has positive effect on the overall LID performance one would expect an increase in confidence; conversely, a decrease in confidence points at model confusion. The differences in top prediction confidences and the relevant statistics are shown in Table 6. For each language, for a sample size of N paragraphs we compute the population mean (μ), variance (σ^2) and 95% confidence intervals ($[\mathrm{CI}^-, \mathrm{CI}^+]$ for the sets of prediction confidences before and after the normalization. To validate whether the per-language prediction confidence differences are statistically significant we perform a two-sample t-test on the two populations and provide the relevant t-stat score and p-value. The null hypothesis rejection threshold is set to 0.05. The t-test results indicating an overall improvement in confidence after the normalization are marked in blue, the cases when normalization harms the confidence are marked in red, while the rest of the cases indicate that normalization has no effect.

According to Table 6, the set of languages for which prediction confidence improvements are observed coincides with the set of languages in Table 5 for which more data is discovered post-normalization with the exception of Adyghe where normalization has no effect on same-language prediction confidence. Similarly, the set of languages where the post-normalization confidence degrades correlates with the languages with negative post-normalization effects in Table 5 with the exception of Bezhta and Tsakhur where no significant changes in prediction confidence are observed likely due to small population size, and Dargwa and Ingush, where the drop in confidence is significant but relatively small.

Inspection of the Differences We perform a manual inspection of randomly sampled 140 paragraphs (10 for each language) gained or lost during the normalization. The results are shown in Table 7, where for each "win" or "loss" category we maintain "exact" (N_e) and "lenient" (N_l) counters. The "exact" counter represents the exact matches between manually established label and the GlotLID label. Some Tsakhur paragraphs, for example, are actually in Rutul according to our inspection. In a more permissive "lenient" mode, we still count this mismatch as a match because Rutul is not supported by GlotLID but belongs to languages of interest (see Table 1). In the exact mode, we treat this mismatch as error. As

Lang.	Wins		Losses		Lang.	Wins		Losses	
	N_e	N_l	N_e	N_l		N_e	N_l	N_e	N_l
abk	0	0	0	0	inh	0	0	2	2
abq	3	4	8	8	kap	2	2	0	0
ady	8	8	9	9	kbd	6	6	7	7
agx	2	3	0	1	lbe	4	4	7	7
ava	9	9	9	9	lez	7	7	7	7
che	0	0	3	4	tab	3	4	1	1
dar	10	10	10	10	tkr	4	8	1	3
...	...	...	...	...	All	58	65	64	68

Table 7: Manual validation of GlotLID language assignments on a random 10-paragraph sample in "wins" and "losses" categories for each language. Two modes of counting are employed: exact (N_e) and lenient (N_l).

can be seen from the table, for "wins" category if counted exactly only 58 (or 41% of 140 new paragraphs) can be counted as real improvements in recall after normalization. The rest of the paragraphs are false positives due to GlotLID misclassifications. For example, we found no Abkhaz, Chechen or Ingush data in the sampled "win" category because all the "Abkhaz" paragraphs are in Ukrainian, "Chechen" paragraphs are random itemized lists of place names and Ingush data, whereas the "Ingush" label gets assigned to mostly Chechen data.

When evaluating the losses we check that pre-normalization language assigned by GlotLID is correct. If it is not, then the paragraph signal was wrong and there is nothing to declare as a loss. For example, for a sample in Table 7 normalization harms a perfectly valid Abaza sentence "Ужвы датша йыгIгIвуаш ажвакI хIвастI" which gets a Belarusian GlotLID label after normalization. For Bezhta, on the other hand, none of the source sentences are in Bezhta (most of them are in a higher-resource neighboring Avar) hence none of the data is truly lost. The inspection of the losses also reveals signals from NEC languages unsupported by GlotLID. For example, the sentences "ОъшIаъ доьвен са нишанне тастIа гъара ехне" and "Ед' ши экIра, экIра инджимишь гьеъа" in severely endangered Udi and definitely endangered Botlikh (Table 1) are labeled by GlotLID as Aghul and Tsakhur, respectively. We also note relatively high overall agreement between manual and GlotLID labels for Adyghe, Avar and Dargwa. According to Table 7 only 46% of "lost" paragraphs can be considered as losses if counted "exactly".

The only language with 100% disagreement between the automatic GlotLID and manually assigned labels is Abkhaz. Further inspection reveals that none of the paragraphs labeled as Abkhaz are in that language. Since these paragraphs include characters from *palochka* confusable set one would expect a label from another member

of NWC family where these characters are legal. However most of these paragraphs are in languages outside the NWC or NEC families, which is likely due to the abnormal (relative to other languages) distribution of paragraph lengths in "Abkhaz" shown in Table 5, which indicates very short paragraphs (4.25 tokens on average) with high consistency (low variance). These paragraphs may be too short for GlotLID to identify correctly.[13]

5. Conclusions

We have investigated the utility of string tokens with *palochka* confusable letter variants for mining multiple writing systems that share that letter. Both the manual inspection and filtering by paragraph-level LID revealed useful NEC and NWC language signals, in particular in languages currently unsupported by the off-the-shelf large coverage GlotLID model. We also explored the effects of confusable character normalization on total recall (*RQ2*) and precision (*RQ3*), which were found to be highly language-specific. We manually evaluated the differences in GlotLID assignments on a small sample of paragraphs and discovered that predictions are mostly unreliable for nine languages out of 14. The error analysis includes data quality issues and language confusions due to phylogenetic similarities leading us to conclude that GlotLID model is sensitive to confusable character noise for this set of languages (*RQ1*). This evaluation revealed further useful signals from out-of-domain languages, such as Udi and Botlikh, in the resulting data.

6. Limitations and Future Work

This exploratory work has several important limitations. Because none of the authors are native speakers of North Caucasian languages, the manual validation of candidate paragraphs and their LID assignments was both time-consuming and brittle. This step relied heavily on existing document metadata and online lexical resources, which are occasionally encoded in legacy orthographies. To improve upon these experiments and facilitate future research, a critical next step is the creation of a public-domain human-annotated, paragraph-level evaluation dataset. This dataset should provide broad coverage of NWC and NEC languages, capturing the orthographic variation of the kind observed in this paper. Crucially, such annotations must account for code-switching, as texts from this region naturally exhibit frequent mixing between languages. Beyond evaluation, this resource would be highly valuable for LLM-driven development, providing necessary data for few-shot prompting and model fine-tuning.

This work relied on two LID sources—document-level labels in MADLAD-400 from a proprietary model, and predictions from the wider-coverage GlotLID—to investigate whether LID can aid the discovery of useful language data that might otherwise be lost without orthographic normalization. While other state-of-the-art LID models like OpenLID (Burchell et al., 2023) exist, they currently lack coverage for NEC and NWC languages. Our reliance on these off-the-shelf models may partially explain why our observed improvements are marginal, as aptly observed by reviewers, and highly language-specific; the long-tail, low-resource languages in our study are simply not prioritized by current state-of-the-art LID systems. Consequently, improving the LID models themselves to better serve data discovery remains an important direction for future research. More specifically, a crucial direction for future work is determining whether to normalize the LID training data, retain *palochka* orthographic variants unaltered, or intentionally inject additional orthographic variance in to improve the robustness of the resulting LID models. Careful investigation of the data that goes into training of LID models and data selection strategies resulting in better model features, especially relevant to NEC and NWC languages, are also crucial. One such model-centric program is outlined in Appendix B.

Another promising research direction is investigating LID strategies tailored specifically to highly data-scarce languages. These approaches could leverage native lexicons (Selamat and Akosu, 2016; Duvenhage, 2019) or Swadesh lists, potentially synthesizing additional training data via unsupervised morphological analyzers (Smit et al., 2014; Grönroos et al., 2020).[14]

Finally, our investigation focused exclusively on letter *palochka* and its variants, which is the iconic visual hallmark of NEC and NWC writing systems marking ejectives and glottal stops. The filters described in this study can be extended in the future to include other important orthographic features specific to these writing systems that distinguish them from the rest of Cyrillic orthographies. One example of such features are the rich consonantal clusters mentioned in Section 2: due to poor coverage of base Cyrillic letters, these languages rely heavily on digraphs, trigraphs, and even tetragraphs to represent single phonemes. This has a stacking effect—a base Cyrillic letter is combined with "modifiers" (the *palochka*, the hard sign, the soft sign, or labialization markers) to represent a specific phoneme.

[13] We observe similar "Abkhaz" paragraph length distribution in data mined from MADLAD-400.

[14] It should be noted however that even unsupervised morphological analyzers require a non-trivial amount of unannotated raw text to train reliably.

7. Ethics Statement

All data and models we use are publicly available. The source web-crawled datasets that we inspect (MADLAD-400 and DCAD-2000) may include the crawled sources of unknown copyright provenance, hence we are not releasing any resulting derived data. We are committed to more accessible and inclusive NLP, and hope that this exploratory work contributes to extending computational approaches to these two families of vulnerable and endangered languages hitherto largely unexplored by the NLP community.

8. Acknowledgements

The authors thank Lawrence Wolf-Sonkin, Vitaly Nikolaev and anonymous reviewers for many useful suggestions for improving this paper.

9. Bibliographical References

Vladimir M. Alpatov. 2000. *150 yazykov i politika, 1917–2000. Sotsiolingvisticheskie problemy SSSR i postsovetskogo prostranstva (*150 languages and politics, 1970–2000. Sociolinguistic problems in the USSR and post-Soviet countries*)*, 2nd edition. Kraft+, Moscow. Institute of Oriental Studies, Russian Academy of Sciences (RAN). In Russian.

Deborah Anderson. 2023. RE: Comments on CYRILLIC CHE WITH HOOK's use in Khanty and Tofa (Tofalar). ISO/IEC 10646 JTC1/SC2/WG2 L2/23-015, Unicode Consortium.

Timofey Arkhangelskiy. 2019. Corpora of social media in minority Uralic languages. In *Proceedings of the Fifth International Workshop on Computational Linguistics for Uralic Languages*, pages 125–140, Tartu, Estonia. Association for Computational Linguistics.

Boris M. Ataev. 2015. The role of Bible translation in preserving the languages of Dagestan. In Marianne Beerle-Moor and Vitaly Voinov, editors, *Language vitality through Bible translation*, volume 95 of *Berkeley Insights in Linguistics and Semiotics*, chapter 12, pages 207–216. Peter Lang Academic Publishers.

Ayten Babaliyeva. 2023. Standard Tabasaran: short grammar sketch. *Languages of the Caucasus*, 6.

Ankur Bapna, Isaac Caswell, Julia Kreutzer, Orhan Firat, Daan van Esch, Aditya Siddhant, Mengmeng Niu, Pallavi Baljekar, Xavier Garcia, Wolfgang Macherey, Theresa Breiner, Vera Axelrod, Jason Riesa, Yuan Cao, Mia Xu Chen, Klaus Macherey, Maxim Krikun, Pidong Wang, Alexander Gutkin, Apurva Shah, Yanping Huang, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2022. Building machine translation systems for the next thousand languages. *arXiv preprint arXiv:2205.03983*.

Nicholas Boucher, Ilia Shumailov, Ross Anderson, and Nicolas Papernot. 2022. Bad characters: Imperceptible NLP attacks. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1987–2004, San Francisco, CA. IEEE.

Laurie Burchell, Alexandra Birch, Nikolay Bogoychev, and Kenneth Heafield. 2023. An open dataset and model for language identification. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 865–879, Toronto, Canada. Association for Computational Linguistics.

Isaac Caswell, Theresa Breiner, Daan van Esch, and Ankur Bapna. 2020. Language ID in the wild: Unexpected challenges on the path to a thousand-language web text corpus. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6588–6608, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Vyacheslav Chirikba. 2016. From North to North West: How North-West Caucasian evolved from North Caucasian. *Mother Tongue: Journal of the Association for the Study of Language in Prehistory*, 21:1–28.

Marina Chumakina. 2017. Caucasian languages. *Oxford Research Encyclopedia of Linguistics*.

Marina Chumakina, Dunstan Brown, Greville G. Corbett, and Harely Quilliam. 2007. A dictionary of the languages of the Archi villages, south Daghestan (Archi-Russian-English). (Online edition). University of Surray, UK.

John Clews. 1988. *Language Automation Worldwide: The Development of Character Set Standards*. British Library Research & Development Reports. SESAME Computer Projects, Harrogate, UK.

John Clews. 1991. Dealing with multiple languages in the computer industry computer character sets: their evolution and impact. In *Proceedings of Translating and the Computer 13: The theory and practice of machine translation – a marriage of convenience?*, London, UK. Aslib.

John M. Clifton, Janfer Mak, Gabriela Deckinga, Laura Lucht, and Calvin Tiessen. 2005. The sociolinguistic situation of the Kryz in Azerbaijan. Survey Report SIL Electronic Survey Reports 2005–006, SIL International, Journal of Language Survey Reports.

Bernard Comrie and Maria Polinsky. 1998. The great Daghestan case coax. In Anna Siewierska and Jae Jung Song, editors, *Case, Typology and Grammar*, volume 38 of *Typological Studies in Language*, pages 95–114. John Benjamins Publishing Company.

Portia Cooper, Eduardo Blanco, and Mihai Surdeanu. 2025. The lies characters tell: Utilizing large language models to normalize adversarial Unicode perturbations. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 18932–18944, Vienna, Austria. Association for Computational Linguistics.

Aldan Creo and Shushanta Pudasaini. 2025. SilverSpeak: Evading AI-generated text detectors using homoglyphs. In *Proceedings of the 1stWorkshop on GenAI Content Detection (GenAIDetect)*, pages 1–46, Abu Dhabi, UAE. International Conference on Computational Linguistics.

Michael Daniel and Yury Lander. 2011. The Caucasian languages. In Bernd Kortmann and Johan van der Auwera, editors, *The Languages and Linguistics of Europe: A comprehensive guide*, volume 1 of *The World of Linguistics (WOL)*, pages 125–158. De Gruyter Mouton, Berlin, Germany.

Perry Deng, Cooper Linsky, and Matthew Wright. 2020. Weaponizing Unicodes with deep learning-identifying homoglyphs with weakly labeled data. In *Proceedings of 2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6, Arlington, VA. IEEE.

Bernardt Duvenhage. 2019. Short text language identification for under resourced languages. In *Proceedings of 33rd Conference on Neural Information Processing Systems (NeurIPS 2019) Workshop on Machine Learning for the Developing World: Challenges and Risks of ML4D*, pages 1–6, Vancouver, Canada.

Evgeniy Gabrilovich and Alex Gontmakher. 2002. The homograph attack. *Communications of the ACM*, 45(2):128–129.

Jost Gippert. 2008. Endangered Caucasian languages in Georgia: Linguistic parameters of language endangerment. In K. David Harrison, David S. Rood, and Arienne Dwyer, editors, *Lessons from Documented Endangered Languages*, volume 78 of *Typological Studies in Language*, pages 159–194. De Gruyter Brill.

Rob Van Der Goot. 2025. Identifying open challenges in language identification. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18207–18227, Vienna, Austria. Association for Computational Linguistics.

Stig-Arne Grönroos, Sami Virpioja, and Mikko Kurimo. 2020. Morfessor EM+Prune: Improved subword segmentation with expectation maximization and pruning. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3944–3953, Marseille, France. European Language Resources Association.

Ekaterina Gruzdeva. 2022. Preserving the languages of Russia: Work in progress. *International Journal of Eurasian Linguistics*, 4(1):65–74.

Yannis Haralambous. 2007. *Fonts & Encodings*. O'Reilly Media, Sebastopol, CA.

George Hewitt. 2010. *Abkhaz: A Comprehensive Self-Tutor*. LINCOM Student Grammars. LINCOM Europa, Munich, Germany.

Tobias Holgers, David E. Watson, and Steven D. Gribble. 2006. Cutting through the confusion: a measurement study of homograph attacks. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 261–266, USA. USENIX Association.

Alan Hopkinson. 1984. International access to bibliographic data: MARC and MARC-related activities. *Journal of Documentation*, 40(1):13–24.

Jafar Isbarov, Arofat Akhundjanova, Mammad Hajili, Kavsar Huseynova, Dmitry Gaynullin, Anar Rzayev, Osman Tursun, Aizirek Turdubaeva, Ilshat Saetov, Rinat Kharisov, Saule Belginova, Ariana Kenbayeva, Amina Alisheva, Abdullatif Köksal, Samir Rustamov, and Duygu Ataman. 2025. TUMLU: A unified and native language understanding benchmark for Turkic languages. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22816–22838, Vienna, Austria. Association for Computational Linguistics.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

David Kamholz, Jonathan Pool, and Susan Colowick. 2014. PanLex: Building a resource for panlingual lexical translation. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 3145–3150, Reykjavik, Iceland. European Language Resources Association (ELRA).

Amir Hossein Kargaran, Ayyoob Imani, François Yvon, and Hinrich Schuetze. 2023. GlotLID: Language identification for low-resource languages. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6155–6218, Singapore. Association for Computational Linguistics.

Amir Hossein Kargaran, François Yvon, and Hinrich Schütze. 2024. GlotCC: An open broad-coverage CommonCrawl corpus and pipeline for minority languages. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, Vancouver, Canada.

Erzhen V. Khilkhanova. 2019. The Internet and minority languages of Russia: Symbolic presence or a revitalization tool? (a case study of the Buryat language). *Mongolian Studies*, 11(4):967–988.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR.

Y. B. Koryakov, T. I. Davidyuk, V. S. Kharitonov, A. P. Yevstigneyeva, and A. A. Syuryun. 2022. Spisok yazykov Rossii i statusy ikh vital'nosti (*List of languages of Russia and their vitality status*). Technical report, Institute of Linguistics, Russian Academy of Sciences (RAN), Moscow. Monograph preprint (in Russian).

Alexey Koshevoy, Anastasia Panova, and Ilya Makarchuk. 2023. Building a Universal Dependencies treebank for a polysynthetic language: the case of Abaza. In *Proceedings of the Sixth Workshop on Universal Dependencies (UDW, GURT/SyntaxFest 2023)*, pages 1–6, Washington, D.C. Association for Computational Linguistics.

Irina Kraeva and Natalia Guermanova. 2020. Language policy of the Russian Federation: Searching for balance among 150 languages. *European Journal of Language Policy*, 12(2):135–162.

Stefan Krsteski, Borjan Sazdov, Matea Tashkovska, Branislav Gerazov, and Hristijan Gjoreski. 2025. Towards open foundation language model and corpus for Macedonian: A low-resource language. In *Proceedings of the 10th Workshop on Slavic Natural Language Processing (Slavic NLP 2025)*, pages 44–57, Vienna, Austria. Association for Computational Linguistics.

Sneha Kudugunta, Isaac Caswell, Biao Zhang, Xavier Garcia, Derrick Xin, Aditya Kusupati, Romi Stella, Ankur Bapna, and Orhan Firat. 2023. MADLAD-400: A multilingual and document-level large audited dataset. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:67284–67296.

Ali Kuzhuget, Airana Mongush, and Nachyn-Enkhedorzhu Oorzhak. 2024. Enhancing tuvan language resources through the FLORES dataset. In *Proceedings of the Ninth Conference on Machine Translation*, pages 593–599, Miami, Florida, USA. Association for Computational Linguistics.

Peiqin Lin, Shaoxiong Ji, Jörg Tiedemann, André F. T. Martins, and Hinrich Schütze. 2024. MaLA-500: Massive language adaptation of large language models. *arXiv preprint arXiv:2401.13303*.

Nikita Manulov. 2022. Proposal to encode 23 Cyrillic characters for old Uslar's Caucasian Alphabets. ISO/IEC 10646 JTC1/SC2/WG2 L2/22-262, Unicode Consortium.

Kirk Miller. 2021. Unicode request for Cyrillic modifier letters. ISO/IEC 10646 JTC1/SC2/WG2 L2/21-107, Unicode Consortium.

Christopher and Moseley. 2010. *Atlas of the World's Languages in Danger*. Memory of peoples. UNESCO.

Kenesbaj Musaevič Musajev. 1965. *Alfavity jazykov narodov SSSR (*Alphabets of the languages of the peoples of the USSR*)*. USSR Academy of Sciences, Nauka, Moscow. In Russian.

E. V. Nikitina, T. N. Evgrafova, and E. I. Antonova. 2019. Russian minority languages representation on the Internet as their social status reflection. In *Proceedings of the 11th International Conference on Communicative Strategies*

of Information Society (CSIS'19, pages 339–348, Saint-Petersburg, Russia.

John Parry. 1991. Computer character sets: their evolution and impact. In *Proceedings of Translating and the Computer 13: The theory and practice of machine translation – a marriage of convenience?*, London, UK. Aslib.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. 2025. FineWeb2: One pipeline to scale them all – adapting pre-training data processing to every language. *arXiv preprint arXiv:2506.20920*.

Kristian Roncero. 2021. Gakvarian Chamalal (Dagestan and Chechnya) — Language snapshot. *Language Documentation and Description*, 20:64–74.

Hugh McGregor Ross. 1984. Handling non-Roman character sets with computers. In *Proceedings of Translating and the Computer 6: Translation and communication*, London, UK. Aslib.

Jack Rueter, Olga Erina, and Nadezhda Kabaeva. 2024. On Erzya and Moksha corpora and analyzer development, ERME-PSLA 1950s. In *Proceedings of the 9th International Workshop on Computational Linguistics for Uralic Languages*, pages 67–75, Helsinki, Finland. Association for Computational Linguistics.

Ali Selamat and Nicholas Akosu. 2016. Word-length algorithm for language identification of under-resourced languages. *Journal of King Saud University - Computer and Information Sciences*, 28(4):457–469.

Yingli Shen, Wen Lai, Shuo Wang, Xueren Zhang, Kangyang Luo, Alexander Fraser, and Maosong Sun. 2025. DCAD-2000: A multilingual dataset across 2000+ languages with data cleaning as anomaly detection. *arXiv preprint arXiv:2502.11546*.

Peter Smit, Sami Virpioja, Stig-Arne Grönroos, and Mikko Kurimo. 2014. Morfessor 2.0: Toolkit for statistical morphological segmentation. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 21–24, Gothenburg, Sweden. Association for Computational Linguistics.

Mukhammed Togmanov, Nurdaulet Mukhituly, Diana Turmakhan, Jonibek Mansurov, Maiya Goloburda, Akhmed Sakip, Zhuohan Xie, Yuxia Wang, Bekassyl Syzdykov, Nurkhan Laiyk, Alham Fikri Aji, Ekaterina Kochmar, Preslav Nakov, and Fajri Koto. 2025. KazMMLU: Evaluating language models on Kazakh, Russian, and regional knowledge of Kazakhstan. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14403–14416, Vienna, Austria. Association for Computational Linguistics.

Unicode Consortium. 2025. Unicode security mechanisms. Unicode® Technical Standard #39, Version 17.0.0, eds. Mark Davis and Michel Suignard, `https://www.unicode.org/reports/tr39/`, file `confusables.txt`.

Maoli Wang, Xiaodong Zang, Jianbo Cao, Bowen Zhang, and Shengbao Li. 2024. PhishHunter: Detecting camouflaged IDN-based phishing attacks via siamese neural network. *Computers & Security*, 138:103668.

Lisa Yankovskaya, Maali Tars, Andre Tättar, and Mark Fishel. 2023. Machine translation for low-resource Finno-Ugric languages. In *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*, pages 762–771, Tórshavn, Faroe Islands. University of Tartu Library.

Ayur Zhanaev and Wojciech Połeć. 2019. A minority language in the globalizing world: The Buryat language on the Internet. *Adeptus*, 2019(14):1–17. Slavistics Institute of the Polish Academy of Sciences.

A. *Palochka* Variant Filter Recall

To estimate the recall of the *palochka* variant filter for DCAD-2000 data discussed in Section 4 and to complement the precision estimates we modify our pipeline as follows: after converting DCAD-2000 documents to paragraphs, the first pipeline steps filter out short paragraphs, apply LID using the GlotLID model, and filter out all paragraphs in non-Cyrillic scripts. The paragraphs belonging to the four exception languages (Belarusian, Kazakh, Rusyn and Ukrainian) are then filtered out followed by the application of the *palochka* variant filter as a last step.

The paragraphs retained and discarded by the last filter are used to compute the overall and per-language values of filter recall as shown in Table 8. The total number of retained paragraphs (marked as true positives, or "TP") approximately corresponds to 1,628,526 paragraphs in NEC and NWC languages with *palochka* variants from Section 4. These numbers are not the same because

Lang.	TP	FN	Recall (%)
abk	6063	6 518 490	0.09
abq	14 329	6711	68.1
ady	136 171	24 137	84.94
agx	8009	95 943	7.7
ava	248 267	95 089	72.31
che	571 565	6 366 306	8.24
dar	21 025	15 654	57.32
inh	77 126	6 475 036	1.18
kap	539	1894	22.15
kbd	396 445	462 904	46.13
lbe	46 904	126 173	27.1
lez	72 608	198 008	26.83
tab	33 448	57 272	36.87
tkr	934	3847	19.54
All	1 633 433	20 447 464	7.4

Table 8: Numbers of paragraphs with (true positives, TP) and without (false negatives, FN) *palochka* variants and the corresponding value of recall.

the sequence of filters used to compute the recall is now different. Overall there are 20,447,464 paragraphs in relevant (according to GlotLID) languages without *palochka* variants shown as false negatives in the "FN" column of Table 8. The total value of recall is therefore 7.4%, which is significantly lower than the precision value of 17.3% computed in Section 4.

Several languages contribute to low recall. Abkhaz has negligible recall because *palochka* is not part of its orthography. Ingush and Chechen, along with Abkhaz, are the higher-resource languages among the 14 languages of interest. The low recall for Chechen and Ingush is likely due to a significantly more stable orthography and corresponding input methods, where the orthographically correct usage of *palochka* is preferred to its different variants. Finally, the relatively low recall for Aghul does not have a clear interpretation and further investigation is needed. The rest of the languages, including the lowest-resource Bezhta and Tsakhur, have reasonably high values of recall, where the best-performing languages are Abaza, Adyghe, Avar, Dargwa, and Kabardinian with a recall over 45%.

B. LID Model-centric Experiments: An Outline

Wide-coverage LID models are trained to perform well across a range of languages, but often perform poorly on under-resourced languages which have a weak prior of appearing in the wild (*e.g.*, in web text). Can we use the presence of a *palochka* confusable, the *palochka* variant filter, as a high precision signal to *repair* predictions made by a modern LID model?

To investigate this question, the following experiment can be designed: take the dataset used to train the latest GlotLID v3.1 model[15] and partition it into a held-out set, where 15% of sentences are randomly held out per language, up to a maximum of 1,000 sentences per language.[16] For the training set, randomly sample up to 10,000 examples for each language from the remainder. Goot (2025) has observed that 1,000 sentences is sufficient for strong LID performance for most languages. Retrain a GlotLID fastText model, with hyperparameters given in (Kargaran et al., 2023), on this training set and generate the predictions on the held-out set.

After inference on the held-out set, collect all examples which were: (1) predicted to be a Cyrillic orthography outside of Belarusian, Ukrainian, Kazakh, and Rusyn;[17] (2) outside the set of NWC/NEC languages covered by GlotLID; and (3) passed the length constraint (contain at least three tokens). Note those that match the *palochka* variant filter, and count how many of these examples are actually labeled as a ground truth NWC/NEC language. The resulting proportion represents the *palochka* precision for North Caucasian orthographies. This experiment can be repeated multiple times to obtain a robust statistical estimate of the significance of computed precision values.

Another possible direction is to investigate the features of the GlotLID model itself, which takes hashed character n-gram features as input. If a particular orthographic feature, such as *palochka* letter, is useful according to the model, it will be included among the highly weighted features of GlotLID for NEC and NWC languages, either in a frequent unigram, or more likely a (hashed) character n-gram. The absence of such orthographic features that are hypothesized to be *a priori* informative from the list of top-weighted model features potentially indicates issues with the training data for the languages in question.

[15] https://huggingface.co/datasets/cis-lmu/glotlid-corpus

[16] As was described in the GlotLID paper.

[17] All of these employed a *palochka* confusable in their standard orthographies.

Prompting Approaches to Abbreviation Expansion

Kyle Gorman
Google Research

Abstract

Abbreviations are an entrenched feature of the majority of the world's writing systems, and the ability to expand abbreviations in context is important for speech technologies and language understanding tasks. This study presents several experiments applying large language models, prompt engineering, and fine-tuning to the expansion of ad-hoc abbreviations in English, showing substantial improvements against noisy channel models previously used for this task.

Keywords: abbreviation expansion, prompting, text normalization

1. Introduction

Abbreviations are one of several types of entities processed during **text normalization** for speech and language processing applications. For speech applications like text-to-speech synthesis or automatic speech recognition, they are *non-standard words* (NSWs; Sproat et al. 2001), wordforms not generally pronounced according to the ordinary character-to-sound rules. Sproat et al. (2001) provide a taxonomy of abbreviations in English focusing on pronunciation. In English, for example, *NATO* is read as if it were a word, but *CIA* is read as a letter-sequence (i.e., an **initialism**), and *Blvd.* is read the same as the full word *Boulevard*. Abbreviation normalization poses difficulties that go beyond mere pronunciation, however. Unlike other types of NSWs, it is not always obvious what an abbreviation denotes, and context may be required to determine what word or phrase an abbreviation corresponds to. Thus abbreviation expansion is ultimately important both for pronunciation modeling and text understanding. Gorman et al. (2021) draw a distinction between high-frequency, *conventionalized* abbreviations—including units like *lbs* read as *pound(s)* or postal abbreviations like *QC* read as *Quebec*—and the open class of *ad-hoc* abbreviations coined on-the-fly as needed. Ad-hoc abbreviations are particularly common in contexts where brevity is important, as it is on mobile devices (Crystal 2008) or augmentative and alternative communication (AAC) devices used by people with gross motor impairments (e.g., Cai et al. 2024).

1.1. Prior work

There is a long literature on English abbreviation formation (e.g., Marchand 1969:§9, Cannon 1989). While much less is known about abbreviations in other languages, Gorman and Roark (2024) survey abbreviation strategies in 50 languages and scripts.

Expansion of abbreviations is by now an established experimental task (see Gorman et al. 2021:§1.2 for a survey up to that date). Abbreviation expansion has also been studied in highly-specific domains; e.g., Daza et al. (2022) study abbreviations occurring in a corpus of Slovenian-language biographies, and Hosseini et al. (2024) model English clinical abbreviations. However, the author is aware of only a few freely-available data sets for this task, mostly in English. Dekker and van der Goot (2020) study synthetic generation of abbreviation-rich English-language microblog posts. For evaluation, they ask human annotators to guess which of two posts is human-generated. They release the synthetic data generated by their best system. Gorman et al. (2021) release a data set created by asking human annotators to abbreviate sentences from English-language Wikipedia.

A wide variety of modeling strategies have been applied to abbreviation generation and expansion. Much prior work uses some form of the well-known noisy channel model (e.g., Aw et al. 2006). Gorman et al. (2021), for example, develop a noisy channel model in which a character-level pair n-gram model (e.g., Novak et al. 2016) is fused with a word-level language model over possible expansions, with the Viterbi algorithm used to decode. They find that imposing additional heuristics on the character-level expansions improves performance, as does replacing a conventional language model with a neural network language model. Cai et al. (2024), who study abbreviation expansion for AAC, use a large fine-tuned language model.

1.2. Contributions

In this study, I perform abbreviation expansion using the current generation of large language models (LLMs) in two different scenarios, one using data augmentation and distillation strategies for resource-constrained, low-latency applications, and one focusing on the quality of expansions generated, free of resource and latency concerns.

While abbreviation expansion is in some ways similar to other sequence-to-sequence tasks like machine translation, grapheme-to-phoneme conversion, etc., I hypothesize that modern LLMs are well-adapted to these tasks, for two reasons. First, abbreviation expansion is conceptually sim-

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 64–70
12 May, 2026.

	# sentences	# tokens
Training	21,318	332,829
Development	2,665	41,757
Testing	2,665	41,730
LM data	2,657,826	41,573,540

Table 1: Summary statistics for the data set (repeated from Gorman et al. 2021).

ilar to the masked word prediction task used during pre-training. Secondly, given the focus on the expansion of abbreviations within their sentential context, this task depends on the generation of locally-coherent text, one of the greatest apparent strengths of LLMs. Finally, I hypothesize that fine-tuning will condition models to properly attend to the input text and to conform to the constraints of abbreviation tasks.

2. Materials and methods

2.1. Data

The primary data for training, development and validation are drawn from the Gorman et al. (2021) corpus, described above, which contains sentences from English-language Wikipedia abbreviated by professional human annotators. Summary statistics are provided in Table 1, and example sentences drawn from this corpus are shown in Figure 1. Gorman et al. constrain the abbreviation task by requiring that any abbreviation be formed by deleting some—but not all—characters, so that any abbreviation is a proper non-empty subsequence of the word it abbreviates. These **deletion-based** abbreviations (Pennell and Liu 2010) are one of the most common form of ad-hoc abbreviation in English text, though as Gorman and Roark (2024) note, English also includes the aforementioned acronyms like *NATO* and *CIA*, stump compounds like *FiDi* ($<$ *Financial District*), and abbreviations that involve a mix of deletion and substitution as in *cuz* ($<$ *because*). Gorman et al. have a second team of annotators attempt to expand abbreviated sentences created by the first team to establish a human topline for the test set.

2.2. Tasks

Two different tasks were considered in this study. One potential application area for abbreviation expansion is in the generation of spoken driving directions, and this motivates a consideration of resources limitations and latency. Consider, for example, a New York driver whose trip requires them to take the next exit onto the Bear Mountain State Parkway. Highway signs for this exit may read *Bear Mtn Pkwy*, *Bear Mtn Pk*, and so on (see Figure 2 for an example). If the driving direction engine wants to give an instruction that the driver should take the next exit onto *Bear Mountain Parkway* any attempt to expand the abbreviations must be fed to the synthesis back-end before the driver reaches the off-ramp. In many cases, it is possible to offload this computation to a remote server via remote procedure call (RPC), but in others it is not possible to retrieve the RPC result (i.e., due to poor connectivity) before the driver reaches the exit. For this reason, a robust driving direction system must be "hybrid" in the sense that it is able to fall back to on-device generation with low latency and minimal degradation in quality. Similar concerns apply for traffic-sign recognition in advanced driver-assistance systems and self-driving vehicles, which read and interpret traffic signs.

In the first scenario, I consider **direct expansion** sentences using LLM prompting. In the driving direction scenario, a prompting expansion system could be run as a high-quality server-based system.

In the second, I consider the use of LLMs for **data augmentation**. That is, the LLM is prompted to generate more training data. This synthetic data is then combined with the original training data and used to train a noisy channel model, a naïve form of model distillation. While Gorman et al. did not design their noisy channel model with mobile devices in mind, that model requires several orders of magnitude fewer parameters than the LLMs used in the prompting experiments, and there are various ways one might reduce its computational footprint, such as lossy LM pruning (e.g., Stolcke 1998) or lossless compression techniques for weighted finite-state automata (e.g., Mohri et al. 2015). Similarly, inference with this model is efficient, allowing for latencies on the order of tens of milliseconds without specialized coprocessors. This simple data augmentation technique may allow the system to distill relevant knowledge into a resource-constrained, low-latency expansion system.

The downstream capabilities of language models—whether large or small, conventional or neural—are to some degree determined by decisions made during pre-processing (e.g., Gorman and Pinter 2025). Of these decisions, the choice of tokenization strategy is thought to be one of the most important (e.g., Bostrom and Durrett 2020). I hypothesize ad-hoc abbreviations might challenge the model in the sense that they might require a large number of short and/or rarely-used tokens. I also hypothesize that tokenization of ad-hoc abbreviations might have different effects in the data augmentation task, which requires generation of abbreviated text, and the direct abbreviation expansion task, which requires processing but not generation.

```
inernet is not th frst tech to rsult in time displcmnt .
they prvided asistnc to parishs wher ordrd by the chrc .
he ws dstngushd by a lngstndng intrst in pplr antiquities .
he latr bcam a sports brdcstr for the college ftbll games .
hwvr , due to their rare pblc apprnces , these rumors remn uncnfrmd .
```

Figure 1: Example human-abbreviated sentences (case-folded and tokenized) from the Gorman et al. development set.

Figure 2: A highway sign on the Palisades Parkway indicating an exit towards Perkins Memorial Drive and Bear Mountain State Parkway. Image credit: Alps' Roads (https://www.alpsroads.net); reproduced with permission of the photographer.

2.3. Models

Prompting experiments were conducted using Gemini 3.0 (size medium, instruction-tuned) with default generation and fine-tuning hyperparameters. The original pair n-gram model used by Gorman et al. is retained as a baseline and is used for data augmentation and distillation experiments.

In both tasks, prompts instructed the system not to insert words or substitute characters. In the data augmentation task, the model was also instructed not to delete words. In **few-shot** experiments, the prompt also included additional sample pairs of abbreviated and expanded sentences ("shots"), randomly sampled without replacement, from the training data, before providing the target example to either be expanded or abbreviated, depending on task. In direct expansion experiments using **fine-tuning**, the LLM was simply fine-tuned to produce the expanded sentence observed in the training data, without any shots in the prompts.

Sample prompts for both tasks are given in the appendix.

2.4. Metrics

Following Gorman et al. (2021), I use **word error rate** (WER), the percentage of incorrectly expanded words, as our primary metric. To facilitate error analysis, several additional statistics are reported. **Overexpansion rate** (OER) is the percentage of words in the hypothesis which are expanded but do not require expansion, i.e., because they are not abbreviated in the first place. **Underexpansion rate** (UER) is the percentage of abbreviated words which are not expanded in the hypothesis. **Incorrect expansion rate** (IER) is the percentage of abbreviated words which are expanded to the wrong word. I also introduce a novel metric, **length error rate** (LER). This statistic, computed at the sentence (rather than word) level, is the percentage of hypotheses whose length in words does not match that of the input sentence.

OER and IER, in particular, measure the degree to which a system is "Hippocratic" (Roark and Sproat 2014) in the sense that it does no harm to a human's ability to interpret abbreviated text. In other words, one might prefer to let human users cope with the unexpanded abbreviations rather than incorrect expansion or overexpansion.

LER can be thought of as a measure of the ability of a system to adhere to the instruction to neither insert nor delete words. It is undefined (or zero) for the Gorman et al. human topline because the annotator interface did not permit insertion and deletion, and it is similarly undefined for the Gorman et al. noisy channel model because that model is similarly constrained.

3. Results

3.1. Direct expansion

In direct expansion experiments, each prompt ends with a single abbreviated sentence and the response is a hypothesis expansion of that sentence. If the response contains a different number of words than the abbreviated sentence, a length error is recorded, and all abbreviated tokens are left as such, resulting in underexpansion errors.[1]

Results are shown in Table 2. All metrics are error rates, and thus lower values indicate better performance. General performance, as measured by WER, improved as the number of shots used was increased. With as few as 5 shots, prompt-based expansion outperformed the noisy channel baseline, and with 20 shots, it may have reached a

[1] One can imagine various heuristics for partially resolving length errors, but I consider the presence of a substantial number of length errors to be highly unexpected and undesirable, so no heuristics of this sort are employed here.

plateau. Further improvements were obtained with zero-shot fine-tuning, which produced the lowest overall WER. This best model represents a 3.62-point absolute (and an 81% relative) reduction in word-level error over the baseline, a substantial improvement. The zero-shot fine-tuning model also achieved the lowest UER and IER observed.

3.2. Data augmentation

In the data augmentation experiments, each prompt ends with a single full sentence from the Gorman et al. training set, and the response is a hypothesis abbreviation of that sentence. If the response contains a different number of words than the full sentence, a length error is recorded and the hypothesis is discarded. Each full sentence is presented 5 times with a different random sample of shots. All hypothesis abbreviations—except those containing length errors, which are discarded—are then concatenated with human-generated training data and the resulting data is used to fit the noisy channel baseline model. Two hyperparameters—the n-gram order of the pair n-gram model and the language model—are set to minimize WER on the held-out development set, and the best model is then evaluated against the test set. Results are shown in Table 3. Unexpectedly, none of the augmented models outperformed the baseline; some possible explanations are proffered in section 4.

3.3. Error analysis

LLM-based direct expansion, whether via few-shot prompting or fine-tuning, results in low OER, UER, and IER across the board. While all prompts included explicit instructions to the model instructing it to neither delete nor insert words, this was not sufficient: pilot experiments with zero-shot expansion (without fine-tuning) did not produce usable results due to excessive length errors, and nearly a quarter of sentences still contain length errors with 1-shot expansion. However, increasing the number of shots and employing fine-tuning were both effective in reducing such errors. Non-"Hippocratic" overexpansion and incorrect expansion errors are very uncommon in both tasks and all systems, though the baseline achieved a lower (i.e., perfect) OER than the prompting models.

Focusing on the fine-tuning-based direct expansion, underexpansion errors—themselves mostly due to the conventions used to handle length errors—and incorrect expansion errors are roughly equally common. Echoing a similar pattern of errors reported by Gorman et al. (2021), many incorrect expansion errors involve morphologically related words (e.g., **decreases* for *decrease*, **technology* for *technological*). As also noted by Gorman et al., the Wikipedia data consists of a mixture of both American and British spelling conventions, and while this results in incorrect expansions according to the exact match criterion used for evaluation, the resulting "errors" (e.g., American **theater* for British *theatre*, British **faecal* for American *fecal*) are mostly harmless for downstream applications.

The proposed data augmentation strategy appears to be capable of generating data that is both diverse and human-like. Table 4, for example, shows five abbreviated versions of a single, randomly-selected training sentence, generated using 20-shot prompting. Of these five, four are unique, and one is identical to the human-generated example from Gorman et al. corpus, despite the fact that the model itself was not exposed to this example.

4. Discussion

As hypothesized, direct expansion performed well, with errors decreasing as the number of shots increased; however, the best result was obtained with zero-shot fine-tuning. The one surprise was the high rate of length errors. While some prior work has studied constraining neural network generation using finite-state automata (e.g., Zhang et al. 2019, Koo et al. 2024), no available implementation was capable of expressing the conceptually simple constraints—i.e., each output word o_n must be a supersequence of the corresponding input word i_n—that characterize this task. Explicit prompt instructions and even fine-tuning were insufficient to completely remove these errors in both direct expansion or data augmentation. Future work should consider the performance of prompting in other output-constrained speech and language processing tasks, such as those which can be framed as tagging tasks. One potential direction is to consider reinforcement learning as an alternative to fine-tuning, under the hypothesis this may more effectively enforce the length constraint.

Despite the increased data diversity introduced by data augmentation, the naïve distillation technique did not improve the performance of the noisy channel model. While the reason for this failure must be speculative at present, it should be noted that the proposed technique provides synthetic data to the pair n-gram component modeling the formation of abbreviations (i.e., what characters are deleted). However, it does not meaningfully enrich the expansion LM component operating at the word level. Gorman et al. (2021) report that replacing the conventional LM with a neural network LM substantially improved performance, and it simply may be the case that there is much less headroom to improve the pair n-gram component. Data augmentation is unlikely to improve the expansion LM, since it itself is trained with naturally-occurring sen-

	WER	OER	UER	IER	LER
Human topline	3.51	2.23	0.30	4.88	(n.a.)
Noisy channel	2.90	0.00	2.13	4.08	(n.a.)
1-shot expansion	12.52	0.10	25.52	0.98	24.13
2-shot expansion	4.19	0.07	7.97	0.87	7.39
5-shot expansion	1.93	0.10	3.07	0.94	2.44
10-shot expansion	1.11	0.08	1.46	0.81	0.94
20-shot expansion	0.97	0.09	1.06	0.90	0.60
Zero-shot fine-tuning	0.57	0.08	0.52	0.59	0.49

Table 2: Direct expansion results on the Gorman et al. abbreviation data test set, with human topline and noisy channel model results from Gorman et al. 2021 for comparison. The best overall performance, as indicated by WER, are obtained using zero-shot prompts and fine-tuning. WER: word error rate; OER: overexpansion rate; UER: underexpansion rate; IER: incorrect expansion rate; LER: length error rate.

	WER	OER	UER	IER	LER	# sentences
Noisy channel	2.90	0.00	2.13	4.08	(n.a.)	21,318
1-shot augmentation	3.42	0.00	2.78	4.56	14.31	112,655
2-shot augmentation	3.42	0.00	2.83	4.52	9.46	117,828
5-shot augmentation	3.53	0.00	2.94	4.66	5.72	121,812
10-shot augmentation	3.58	0.00	2.95	4.76	4.01	123,636
20-shot augmentation	3.44	0.00	2.94	4.45	3.26	124,429

Table 3: Expansion results on the Gorman et al. abbreviation data test set with data augmentation and naïve distillation, with un-augmented results from Gorman et al. 2021 for comparison. Augmentation data that did not conform to the length specification was excluded, resulting in differing numbers of training sentences. WER: word error rate; OER: overexpansion rate; UER: underexpansion rate; IER: incorrect expansion rate; LER: length error rate; # sentences: number of training sentences.

tences of English. This data requires no additional labeling either by human or LLM.

5. Conclusions

Few-shot prompting and fine-tuning were both highly-effective methods for expanding English deletion-based abbreviations, resulting in substantial improvements over previous models and the human topline. However, data augmentation, combined with a naïve distillation strategy, was less effective, perhaps because the augmentation strategy failed to introduce diversity relevant to improving task performance. Future work should consider alternative augmentation strategies, generalize beyond deletion-based abbreviations, consider languages other than English, and make incremental improvements in prompt design, hyperparameter tuning, generation/decoding, and the like.

6. Limitations

The experiments reported here use the Gemini 3.0 size-medium instruction-tuned model, and the results may depend in part on the particular strengths or weaknesses of that model with respect to these tasks. However, pilot experiments using smaller models gave similar results, and given that the primary data is publicly available, the experiments could easily be reproduced using various public LLM endpoints and other publicly-available families of models.

The results reported here naturally also depend on hyperparameters and the wordings of the prompts, but no attempt was made to tune the generation hyperparameters, or the training hyperparameters for fine-tuning, or the structure of prompts.

Given that the models targeted here are pretrained on large amounts of web text, there is reason to suspect that these model might already have been exposed to some of the target sentences, given that these sentences were taken from Wikipedia and that the corpus was distributed via GitHub. Furthermore, many LLM products have an end-user license agreement which permits any submitted prompts to be used for future training. For example, Balloccu et al. (2024) perform a systematic review of recent NLP conference proceedings to identify data sets "contaminated" via submission to OpenAI's GPT-3.5 or GPT-4 endpoints. It is not clear how to rule out the possibility that the improved performance seen with the LLM prompting techniques reflects previous exposure to the Gorman et al. data.

```
thus , th tw powrs wer relatvly unabl to fght decisve battls .
thus , th two powrs wer relativly unabl to fght decisiv battls .
thus , th two powrs wer relatvly unabl to fght decisve battls .
thus , th two powrs wer reltivly unabl to fight dcisiv bttls .
thus , th two powrs wer relativly unabl to fght decisiv battls .
```

Table 4: Different versions of the same sentence abbreviated via 20-shot prompting, showing the natural diversity obtained by randomly varying which shots were used.

A final limitation of the experiments reported here is inherited from the Gorman et al. data, which only contains deletion-based English abbreviations. Given the considerable cross-linguistic diversity in abbreviation formation strategies (e.g., Gorman and Roark 2024) and general interest in the cross-linguistic capabilities of LLMs, future work should gather abbreviation data from other languages and scripts, and even in English, consider other types of abbreviation formation strategies. Finally, given the relevance of abbreviation expansion to traffic-sign recognition, it may make sense to consider a multimodal, vision-to-text abbreviation expansion task in future work.

7. Acknowledgments

Thanks to Adrian Benton, Christo Kirov, Shankar Kumar, and Brian Roark for technical assistance.

8. Bibliographical References

AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. 2006. A phrase-based statistical model for SMS text normalization. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 33–40.

Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondrej Dusek. 2024. Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source LLMs. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 67–93.

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624.

Shanqing Cai, Subhashini Venugopalan, Katie Seaver, Xiang Xiao, Katrin Tomanek, Sri Jalasutram, . . . , and Michael P. Brenner. 2024. Using large language models to accelerate communication for eye gaze typing users with ALS. *Nature Communications*, 15:9449.

Garland Cannon. 1989. Abbreviations and acronyms in English word-formation. *American Speech*, 64(2):99–127.

David Crystal. 2008. *Txtng: The Gr8 Db8*. Oxford University Press.

Angel Daza, Antske Fokkens, and Tomaž Erjavec. 2022. Dealing with abbreviations in the Slovenian biographical lexicon. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8715–8720.

Kelly Dekker and Rob van der Goot. 2020. Synthetic data for English lexical normalization: How close can we get to manually annotated data? In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 6300–6309.

Gemini Team Google. 2023. Gemini: A family of highly capable multimodal models. ArXiv preprint arXiv:2312.11805. URL: https://arxiv.org/abs/2312.11805.

Kyle Gorman, Christo Kirov, Brian Roark, and Richard Sproat. 2021. Structured abbreviation expansion in context. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 995–1005.

Kyle Gorman and Yuval Pinter. 2025. Don't touch my diacritics. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 285–291.

Kyle Gorman and Brian Roark. 2024. Abbreviation across the world's languages and scripts. In *Proceedings of the Second Workshop on Computation and Written Language (CAWL) @ LREC-COLING 2024*, pages 36–42.

Manda Hosseini, Mandana Hosseini, and Reza Javidan. 2024. Leveraging large language models for clinical abbreviation disambiguation. *Journal of Medical Systems*, 48:27.

Terry Koo, Frederick Liu, and Luheng He. 2024. Automata-based constraints for language model decoding. In *First Conference on Language Modeling*.

Hans Marchand. 1969. *The Categories and Types of Present-Day English Word-Formation*, 2nd edition. Beck.

Mehryar Mohri, Michael Riley, and Ananda Theertha Suresh. 2015. Automata and graph compression. In *2015 IEEE International Symposium on Information Theory*, pages 2989–2993.

Joseph Novak, Nobuaki Minematsu, and Keikichi Hirose. 2016. Phonetisaurus: Exploring grapheme-to-phoneme conversion with joint n-grams models in the WFST framework. *Natural Language Engineering*, 22(6):907–938.

Deana Pennell and Yang Liu. 2010. Normalization of text messages for text-to-speech. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4842–4845.

Brian Roark and Richard Sproat. 2014. Hippocratic abbreviation expansion. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369.

Richard Sproat, Alan W. Black, Stanley Chen, Shankar Kumar, Mari Ostendorf, and Christopher Richards. 2001. Normalization of non-standard words. *Computer Speech & Language*, 15(3):287–333.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *Proceedings of the DARPA Broadcast News And Understanding Workshop*, pages 270–274.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

A. Sample prompts

A.1. One-shot expansion prompt

```
You are a natural language
    annotation system. The following
    text is an English sentence that
    has been abbreviated by deleting
    some of the characters. Expand it
     to a complete, pragmatically
    plausible English sentence by
    inserting alphabetic characters
    into words. Respond in all
    lowercase. Do not substitute or
    delete characters. Do not insert
    or delete whitespace. Do not
    insert or delete words. Please
    return only the expanded sentence
    .
Here is an example:
th key featurs of knwldge rpostories
     nclud comuncatn forums . | the
    key features of knowledge
    repositories include
    communication forums .
at th turnamnt teams hd to present
    smthng based n ther projct . |
```

A.2. One-shot augmentation prompt

```
You are a natural language
    annotation system. The following
    text is an English sentence.
    Abbreviate it by deleting some of
     the characters. Do not
    substitute or insert characters.
    Do not insert or delete
    whitespace. Do not insert or
    delete words. Please return only
    the abbreviated sentence.
Here is an example:
this reaction uses calcium as a
    cofactor and plays an important
    role in the intracellular
    transduction of many
    extracellular signals . | this
    reactn uses calcum as a cofctr
    and plys an imprtn rle in the
    intracelular trnsductn of many
    extracellular signals .
the game features high precision
    physics simulation , online
    multiplayer and open architecture
     . |
```

Evaluating Data Augmentation Strategies for Training Spanish Misspelling Detection Models

Manuel Castillo-Sancho[a], **Jordi Porta-Zamorano**[b], **Asunción Gómez-Pérez**[c]
[a,b]Centro de Estudios de la Real Academia Española, [c]Real Academia Española
{manuel.castillo, porta, agp}@rae.es

Abstract

This paper evaluates three data augmentation strategies for training misspelling detection models in Spanish. Using the Spanish CORRSIC corpus of naturally occurring misspellings, we compare three misspelling generation methods: random perturbations, keyboard-based errors, and a statistical model derived from empirical edit patterns encoded as weighted finite-state transducers. We also analyze two word selection strategies (random and length-based) and two augmentation configurations designed to balance data diversity and reduce spurious correlations. This study shows that the statistical model produces misspellings most similar to real data, showing the lowest Jensen–Shannon divergence (0.148 nats) with the empirical distribution. In downstream detection experiments, performance improves with training size, and differences between word selection strategies remain minimal. Overall, the results highlight the value of statistically grounded misspelling generation for realistic and effective data augmentation in spell-checking tasks in Spanish.

Keywords: misspelling detection, data augmentation, spelling correction, error generation, Spanish NLP

1. Introduction

Spell checking remains a fundamental component of natural language processing (NLP) systems, supporting a wide range of applications from text editing tools to large-scale information retrieval and language learning platforms. Despite its long history, the task continues to present challenges, particularly in adapting to diverse domains and handling the variability of real-world misspellings. A central difficulty is the scarcity of annotated corpora that capture authentic spelling errors with sufficient coverage to train modern discriminative models.

Data augmentation offers a promising avenue to address this limitation. By generating synthetic misspellings that approximate naturally occurring ones, it is possible to expand training data, improve model robustness, and reduce overfitting. However, the effectiveness of different augmentation strategies for spell checking has not been systematically assessed. Existing approaches often rely on ad hoc misspelling generation heuristics, such as random substitutions or keyboard-based noise, without a clear understanding of their impact on downstream misspelling detection performance.

This paper assesses the role of data augmentation in spell checking by evaluating multiple misspelling generation strategies in Spanish under controlled conditions. We compare random misspelling injection, keyboard-proximity perturbations, and statistical models designed to approximate empirical misspelling distributions observed in real-world corpora. Our experiments focus on misspelling detection with discriminative models, analyzing how different augmentation strategies and sampling methods influence precision, recall, and overall performance.

2. Related Work

Recent research has explored various approaches to modeling and generating spelling errors for data augmentation in spelling correction and detection tasks. The survey on grammatical error correction by Bryant et al. (2023) includes a section with references devoted to the generation of synthetic data to produce grammatical errors in text and classifies the methods as noise injection, back-translation, and round-trip translation. A particularly relevant subsequent contribution is that of Martynov et al. (2023), who investigate augmentation methods for emulating human misspellings in Russian. They introduce a multi-domain corpus of genuine spelling errors and compare two complementary generation strategies: Augmentex, a heuristic method based on random and keyboard-level perturbations, and Statistic-based Spelling Corruption (SSC), which derives probabilistic error distributions from aligned correct-incorrect word pairs. Their evaluation, using distributional similarity measures such as the Kolmogorov-Smirnov test, demonstrates that the statistical model produces synthetic errors that more closely resemble real ones, particularly in substitution-dominated domains.

Our work follows a similar line of inquiry but focuses on Spanish and emphasizes empirical evaluation of data augmentation strategies for training misspelling detection models rather than correction. Using the CORRSIC corpus of naturally occurring Spanish misspellings as reference, we compare random, keyboard-based, and statistical generation methods, along with alternative word selection

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 71–78
12 May, 2026.

and augmentation configurations. These studies converge on a key insight: statistically grounded models that reproduce empirical edit distributions generate more realistic and effective training data than purely heuristic approaches. However, unlike Martynov et al. (2023), our experiments directly quantify the downstream impact of each augmentation strategy on discriminative detection performance, thereby bridging the gap between distributional realism and task-specific utility. A more comprehensive account of the methodology, results, and implications of this work is presented in (Castillo-Sancho, 2025), a Master's thesis developed within the framework of the LEIA project[1].

3. Description of CORRSIC

CORRSIC is a manually annotated Spanish corpus of natural misspellings derived from the *Corpus de Referencia del Español Actual* (CREA)[2], carefully annotated by qualified linguists to ensure high-quality labeling. Compiled several decades ago, it consists of text segments, typically full sentences, containing misspellings together with their corresponding corrections. The spelling of the corpus has been updated to reflect the latest reform of Spanish orthography. The size of the CORRSIC corpus is summarized in Table 1.

Element	Count
Text Segments	49676
Words	1696344
Annotated Misspellings	49958
Misspelling Density	2.95%
Misspelling Types	30008

Table 1: Summary of the CORRSIC corpus.

In the annotation scheme used in CORRSIC misspellings are marked with the `<sic>` element and the attribute `@corr` providing the corrected form. An illustrative example is shown below:

```
<sic corr="Qué">Que</sic>
bueno contar con una página
de información como la suya.
A mi hijo le interesa mucho
este deporte.  Él aún es pe-
queño y me gustaría saber
<sic corr="como">cómo</sic>
lo puedo encauzar para que
no lo tome temporalmente nada
<sic corr="más">mas</sic>,
sino que le ayude en su de-
sarrollo personal más <sic
corr="adelante">adlante</sic>.
```

where *que*/*qué*, *como*/*cómo* and *mas*/*más* are pairs of words distinguished by a diacritic accent, i.e., a special use of the accent mark that distinguishes words that are spelled the same but have different meanings or grammatical functions, and *adlante* is a misspelling of *adelante*.

A 60–40% split of this corpus was used to evaluate different misspelling detection models under natural conditions. The 60% portion will be used to obtain statistics for the statistical model, while the remaining 40% will serve as a test set.

4. Misspelling Modeling

To assess whether synthetic misspelling generation strategies for data augmentation influence the performance of the misspelling detection model, and to what extent, three models were developed in the LEIA project. Specifically, we compare: (i) a model that generates misspellings at random, (ii) a model that introduces misspellings based on keyboard proximity, and (iii) a model that produces misspellings following a distribution similar to that observed in CORRSIC.

4.1. Random and Keyboard Models

4.1.1. Random Model

This model generates misspellings through random perturbations, applying a fixed number of modifications to the characters of a word. The possible operations include the insertion of a random character, the substitution of one character with another, the deletion of an existing character, and the transposition of two consecutive characters.

4.1.2. Keyboard Model

This model simulates misspellings derived from the spatial organization of the QWERTY keyboard. Although some implementations exist for simulating keyboard errors in English (Ma, 2019), they do not fully support a Spanish keyboard layout; for this reason, we developed our own method.

The starting point is a complete representation of the Spanish layout, divided into the base layer, the *Shift* layer, the *AltGr* layer, and the extended layers with accented and diacritic characters. Each printable character is assigned a position in this map, which enables the definition of its neighbors in two dimensions.

Candidate substitutions for a given character are determined by computing their topological distance on the keyboard grid. The substitution probability

[1] *Lengua Española e Inteligencia Artificial* (Spanish Language and Artificial Intelligence): https://www.rae.es/leia-lengua-espanola-e-inteligencia-artificial

[2] https://www.rae.es/banco-de-datos/crea

decreases as this distance increases, following an exponential decay function. As a result, characters located in close proximity to the original key are the most likely candidates for replacement.

In addition to this distance-based weighting, the model integrates factors associated with the nature of the keyboard layout. Substitutions within the same layer are assigned the highest weight, while transitions across related layers (e.g., base to *Shift*) receive reduced weight, and substitutions involving more distant layers (e.g., base to *AltGr* or diacritic variants) are penalized more strongly. The final probability of substitution is obtained by combining the spatial and layer-based factors, followed by normalization of the resulting distribution. A replacement character is then sampled according to this distribution.

Additionally, the model incorporates a small probability of introducing diacritic errors in vowels. Due to the complexity and variability of diacritic input mechanisms, these substitutions are not constrained by the keyboard topology and are instead applied independently of spatial considerations.

Finally, it is important to note that this model is limited to character substitutions. It does not simulate insertion or deletion errors, focusing exclusively on replacement phenomena derived from typing interaction.

4.2. Statistical Model

The statistical model seeks to approximate the empirical distribution of misspellings observed in the CORRSIC test partition. To achieve this, misspelling patterns are first extracted from the CORRSIC training partition and then used to construct a weighted finite-state transducer (WFST) (Mohri, 2009) capable of generating misspellings in an input string with a distribution similar to that observed in CORRSIC.

4.2.1. Misspelling Patterns Extraction

The extraction of misspelling patterns starts from aligned pairs of correct-incorrect words in the CORRSIC train partition. Each pair is processed using the Damerau-Levenshtein algorithm to find the minimal edit script (Damerau, 1964; Levenshtein, 1966), i.e., the sequence of atomic operations (insertion, deletion, substitution) that transforms the correct form into the erroneous one. For example, the pair (*amigo* / *amgo*) yields the single operation $i \to \epsilon$, while (*soleado* / *ssokeado*) results in the insertion $\epsilon \to s$ plus the substitution $l \to k$.

Three refinements are introduced to capture misspelling patterns more contextually and realistically. First, explicit boundary symbols are added to model edits occurring at the beginning or end of words. Second, consecutive edits are merged when they co-occur. Third, the representation is extended from unigrams to bigrams whenever an insertion or deletion takes place.

4.2.2. Misspelling Transducer Construction

Every edit operation is stored in the form of a tuple (α, β), where α is a string to be replaced in the input, and β is its replacement in the output string. After extracting the operations for each word pair in the corpus, they are aggregated over the entire dataset to obtain their empirical frequencies.

The probabilistic model of spelling errors is then derived directly from these extracted patterns. Each edit operation $\alpha \to \beta$, together with its empirical probability, is encoded as a transition in a WFST. In this framework, states represent alignment positions, and transitions correspond to possible edits (substitutions, insertions, deletions, or identity mappings when no error occurs).

The WFST is defined over the tropical semiring, where the weight of a path is calculated as the sum of the weights of its transitions. Each transition in the transducer corresponds to a possible edit operation and is weighted by the negative logarithm of its empirical probability.

As a result, a complete misspelling sequence transforming a word into a misspelled variant corresponds to a path through the transducer. The path weight is the sum of the individual edit costs, which in the tropical semiring framework is equivalent to the negative logarithm of the product of their probabilities.

By applying the inverse of the logarithmic transformation, the weight of a path can be mapped back into a probability value.

This formulation allows us to take advantage of the shortest-path algorithms available for WFSTs. Given an input word w, it is composed with the WFST, and the operation $\mathrm{ShortestPath}(w \circ \mathrm{WFST}, k)$ returns the k lowest-cost paths, i.e., the k most probable misspellings (Mohri, 2002). Because each path weight can be mapped back into a probability through the inverse log transformation, we can not only rank candidates but also sample from them according to their true probabilities. In practice, we set $k = 1000$ to balance computational time and coverage.

4.2.3. Misspelling Model Evaluation

When comparing discrete probability distributions, such as the misspelling distributions generated by the misspelling models and the empirical distribution extracted from the CORRSIC test partition, a common choice is the Kullback–Leibler (KL) divergence (D_{KL}) (Kullback and Leibler, 1951), as it directly quantifies the information loss incurred when one distribution is used to approximate another. It

is defined as follows:

$$D_{\mathrm{KL}}(Q \,||\, P) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}$$

However, D_{KL} presents well-known limitations: it is asymmetric ($D_{\mathrm{KL}}(P \,\|\, Q) \neq D_{\mathrm{KL}}(Q \,\|\, P)$), which complicates its interpretation as a measure of similarity, and it becomes infinite whenever the supports of the compared distributions, those produced by the misspelling model and those observed in the CORRSIC test partition, do not fully overlap. To address these issues, we adopt the Jensen–Shannon (JS) divergence (Lin, 1991), a symmetrized and smoothed variant of KL that is always finite and thus better suited for comparing empirical discrete distributions. JS divergence is defined as follows:

$$D_{\mathrm{JS}}(P \,||\, Q) = \tfrac{1}{2} D_{\mathrm{KL}}(P \,||\, M) + \tfrac{1}{2} D_{\mathrm{KL}}(Q \,||\, M)$$

where $M = \frac{1}{2}(P + Q)$.

The JS divergence was applied to compare the edit distributions of words misspelled by different models. For reference, the JS divergence between the CORRSIC train and test partitions is 0.006 nats. We used the same misspelled words from the corpus test partition and applied the other noise-injection methods to them. From Table 2, we observe that the statistical model (Statistic) yields the lowest JS divergence with respect to the empirical CORRSIC test partition distribution (0.148 nats), indicating that it produces misspelling patterns most similar to those found in real data. By contrast, the keyboard-based misspelling model shows the highest divergence (0.548 nats), while the random model lies in between (0.374 nats). This higher divergence in the keyboard model can be attributed, in part, to its restriction to character substitutions only, as it does not simulate insertion or deletion errors, which are present in the real data.

P	$D_{\mathrm{JS}}(P, \mathrm{CORRSIC}_{\mathrm{Test}})$
Random	0.374
Keyboard	0.548
Statistic	0.148

Table 2: Jensen-Shannon divergence between the distributions of editions in words produced by different misspelling models and the empirical distribution from CORRSIC test partition. Units are given in nats.[1]

The ability of the statistical error model to emulate other misspelling distributions was also evaluated. The distribution of keyboard-induced errors is accurately reproduced by the WFST model, with a Jensen–Shannon divergence of 0.041 nats. In contrast, the distribution of random errors is not captured as precisely, showing divergences of 0.245 nats with k=1000 and 0.222 nats with k=5000. This difference can be explained by the greater variability of the randomly generated misspellings, which makes it more difficult to model a stable statistical pattern.

4.3. Length-Based Word Selection

As previous studies have shown for English (Özbey et al., 2022; Flor et al., 2015), word length and word frequency strongly influence the production of misspellings. When words are randomly sampled from running text, the resulting word-length distribution becomes heavily biased toward shorter words, in accordance with Zipf's law, since high-frequency words tend to be shorter. As illustrated in Figure 1, this distribution differs markedly from that observed in the Spanish CORRSIC corpus. The same figure also presents the word-length distribution of words sampled according to this empirical distribution. These two word selection strategies for error injection, the Random and Length-based approaches, are used in the experiments for training the misspelling detection models.

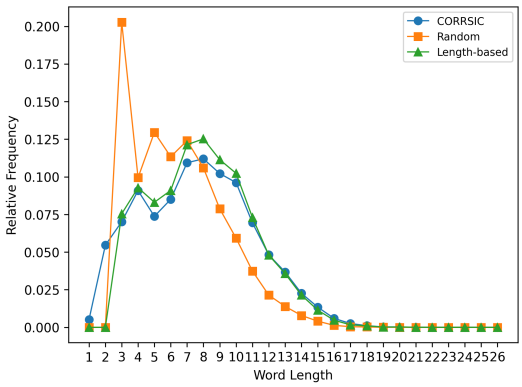


Figure 1: Comparison of word-length distributions for randomly sampled words, words from CORRSIC, and length-based samples replicating CORRSIC's distribution.

5. Experiments on Misspelling Detection

Misspelling detection is framed as a token classification problem, implemented through a discriminative model that predicts a misspelling/non-misspelling label for each token in the input sequence. We use `roberta-base-bne`[3], a monolingual Spanish variant of RoBERTa and the BIO

[1] Nats are the unit of information when logarithms are taken in base e (the natural logarithm).

[3] https://huggingface.co/PlanTL-GOB-ES/roberta-base-bne

(*Beginning–Inside–Outside*) scheme (Ramshaw and Marcus, 1995), where tokens at the beginning of a misspelling span are tagged as `B-ERR`, tokens inside the span as `I-ERR`, and all other tokens as `O`.

Table 3 reports the results of the misspelling detection task on CORRSIC, comparing different word selection strategies for noise induction, misspelling models, and alternative sampling methods for constructing the training set for fine-tuning the model. The word selection strategies are based on random sampling (RND) and on word length (LEN), designed as described in 4.3 to approximate the length distribution observed in CORRSIC. Regarding the selection of training examples, two approaches were considered: one that augments each noisy example by generating multiple misspelling combinations at varying density levels (from 25% downward) (AUG1), and another that performs random sampling over the augmented set generated by AUG1 (AUG2). AUG1 includes fewer base sentences but with a larger variety of misspelling combinations, whereas AUG2 contains a greater number of distinct base sentences but fewer combinations. This design aims to mitigate potential spurious correlations (Ye et al., 2025; Liu et al., 2024) arising from the model's memorization of specific errors or contextual patterns.

With respect to the size of the training data, we observe a general improvement in misspelling detection metrics on the CORRSIC test partition as the training set increases, suggesting that model performance scales with data availability. The full dataset contains 10 million sentences, and experiments with fewer samples were performed on subsets derived from this complete corpus.

In spell-checking tasks, however, precision is typically favored over recall, and performance is therefore often reported in terms of $F_{0.5}$ rather than F_1. As shown in Table 3, the statistical model (c) achieves higher F and recall scores than the random (a) and keyboard-based (b) models, indicating a stronger ability to capture real misspelling patterns. Regarding data-augmentation configurations, AUG2 yields higher precision but lower recall than AUG1, consistent across all models. This can be explained by the fact that AUG2 includes a wider range of distinct sentences but fewer misspelling variations per sentence, allowing the model to observe more diverse contexts and become more precise in identifying actual errors. However, the limited number of variations per context reduces its exposure to certain error structures, leading to lower recall. In contrast, AUG1 exposes the model to fewer unique contexts but with a larger set of misspelling combinations, enabling it to better generalize across diverse error patterns and achieve higher recall, albeit at the cost of precision. The mixed model (d) reaches an F score comparable to that of the statistical model, yet with slightly higher precision and somewhat lower recall, suggesting that combining generation strategies can balance complementary strengths while maintaining overall performance.

The results indicate only minimal differences between random and length-based word selection strategies in Spanish. Since the reported scores are aggregate values, no valid p-value can be computed from them alone. However, the descriptive comparison suggests that the choice of word selection method has little impact on downstream performance when using discriminative models, which calls into question the benefit of more complex selection heuristics. A possible explanation is that the models operate at the level of the tokens defined by the underlying model's tokenizer rather than whole words, which may reduce the impact of word-level selection strategies. Supporting this, the JS divergence between the token distributions of datasets with different word selection strategies is 0.00787 nats, indicating that their token distributions are nearly identical.

To evaluate how accurately a detection model trained on a dataset with misspellings produced by different generation methods can identify the source of each error, we computed the corresponding confusion matrix (Figure 4). The figure shows these results, revealing a certain degree of distinguishability among the different methods. In general, the detection model successfully recognizes most errors generated by their respective methods, with accuracies of approximately 77%, 85%, and 79% for the Random, Keyboard, and Statistical models, respectively. Confusions tend to occur mainly between the Random and Statistical categories, which present more overlapping noise patterns. By contrast, errors generated through keyboard-based perturbations are more easily identifiable, suggesting that this type of noise exhibits more distinctive and consistent features.

6. Limitations

This study presents several limitations that should be acknowledged. First, the evaluation is confined to the datasets and experimental conditions considered here, which may limit the external validity of the findings across domains and error distributions. Second, because the analysis is based on aggregate performance scores, it does not permit formal statistical significance testing, thereby constraining the strength of comparative claims. Third, the study does not differentiate between competence errors and performance errors, despite the fact that these error types may respond to distinct linguistic, cognitive, and contextual factors. Fourth, the keyboard-

Size	Word Selector	AUG1				AUG2			
		$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$	$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$
500k	RND	0.73	0.64	0.68	0.71	0.86	0.58	0.69	0.78
	LEN	0.71	0.67	0.69	0.70	0.83	0.61	0.70	0.77
1M	RND	0.77	0.62	0.68	0.73	0.84	0.60	0.70	0.78
	LEN	0.73	0.66	0.69	0.71	0.82	0.62	0.70	0.77
5M	RND	0.74	0.67	0.71	0.72	0.80	0.65	0.72	0.76
	LEN	0.75	0.68	0.71	0.73	0.77	0.66	0.71	0.75
10M	RND	0.78	0.68	0.72	0.76	0.76	0.66	0.71	0.74
	LEN	0.78	0.68	0.73	0.76	0.76	0.68	0.72	0.74

(a) Random Misspelling Model

Size	Word Selector	AUG1				AUG2			
		$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$	$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$
500k	RND	0.74	0.62	0.67	0.71	0.84	0.55	0.66	0.76
	LEN	0.76	0.60	0.67	0.72	0.85	0.56	0.67	0.77
1M	RND	0.73	0.63	0.68	0.71	0.82	0.58	0.68	0.76
	LEN	0.74	0.62	0.68	0.71	0.83	0.60	0.69	0.77
5M	RND	0.78	0.63	0.70	0.74	0.80	0.63	0.70	0.76
	LEN	0.79	0.63	0.70	0.75	0.76	0.65	0.70	0.74
10M	RND	0.78	0.63	0.70	0.74	0.75	0.67	0.71	0.73
	LEN	0.77	0.68	0.72	0.75	0.76	0.67	0.71	0.74

(b) Keyboard Misspelling Model

Size	Word Selector	AUG1				AUG2			
		$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$	$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$
500k	RND	0.67	0.79	0.73	0.69	0.83	0.74	0.78	0.81
	LEN	0.69	0.79	0.74	0.71	0.83	0.76	0.79	0.81
1M	RND	0.72	0.78	0.75	0.73	0.82	0.77	0.79	0.81
	LEN	0.72	0.80	0.76	0.73	0.83	0.77	0.80	0.82
5M	RND	0.75	0.81	0.78	0.76	0.79	0.81	0.80	0.79
	LEN	0.75	0.84	0.79	0.77	0.78	0.83	0.80	0.79
10M	RND	0.72	0.84	0.78	0.74	0.75	0.84	0.79	0.77
	LEN	0.75	0.85	0.80	0.77	0.76	0.84	0.80	0.77

(c) Statistical Misspelling Model

Size	Word Selector	AUG1				AUG2			
		$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$	$\mathbf{P}$	$\mathbf{R}$	$\mathbf{F_1}$	$\mathbf{F_{0.5}}$
500k	RND	0.75	0.69	0.72	0.74	0.84	0.69	0.75	0.81
	LEN	0.72	0.73	0.73	0.72	0.85	0.70	0.77	0.82
1M	RND	0.77	0.71	0.74	0.76	0.84	0.72	0.77	0.81
	LEN	0.77	0.72	0.75	0.76	0.85	0.72	0.78	0.82
5M	RND	0.79	0.75	0.77	0.78	0.83	0.72	0.77	0.81
	LEN	0.82	0.74	0.78	0.80	0.83	0.76	0.79	0.81
10M	RND	0.81	0.76	0.79	0.80	0.83	0.75	0.79	0.81
	LEN	0.83	0.75	0.79	0.81	0.84	0.73	0.78	0.82

(d) Mixture of Random, Keyboard and Statistical Misspelling Models

Table 3: Results of the downstream misspelling detection task on the CORRSIC test partition under different training set sizes, word selection strategies, and data augmentation methods for different misspelling generation.

based misspelling model is restricted to a single input-device setting and therefore does not capture the variability associated with other keyboards, mobile devices, or alternative input modalities. Finally, the corpus employed is not fully contemporary, which may reduce its representativeness with respect to present-day Spanish usage; diachronic change and distributional shift may affect both the prevalence and the form of misspellings, and thus limit the generalizability of the results to present-

Real/Pred	Random	Keyboard	Statistical
Random	13178 (**0.77**)	2250 (0.13)	1639 (0.10)
Keyboard	1064 (0.06)	14806 (**0.85**)	1490 (0.09)
Statistical	2093 (0.12)	1568 (0.09)	13603 (**0.79**)

Table 4: Confusion matrix for the identification of the source model that generated each misspelling. Values are reported as absolute counts, with percentages in parentheses.

day text. The results reported here remain valid for the relative comparison of models, even under the limitations described.

7. Conclusions

This work has presented a systematic evaluation, under the conditions and limitations described above, of data augmentation strategies for training misspelling detection models in Spanish. Using the CORRSIC corpus as a reference for natural errors, we compared three misspelling generation approaches—random, keyboard-based, and statistical—and analyzed their impact on downstream detection performance.

Our results show that the statistical model, which reproduces empirical misspelling distributions through a weighted finite-state transducer, most closely approximates real misspelling patterns, as confirmed by the lowest Jensen–Shannon divergence values. Although the keyboard and random models also improve robustness, their generated noise deviates more from the empirical distribution.

In the downstream detection task, performance increases consistently with training data size, while differences between random and length-based word selection remain marginal. Moreover, the two augmentation configurations (AUG1 and AUG2) exhibit complementary behaviors in terms of precision and recall, highlighting the importance of balancing data diversity and contextual variability.

Overall, the findings suggest that statistically grounded error generation constitutes an effective and realistic strategy for data augmentation in Spanish spell-checking systems. Future work will extend these experiments to multilingual settings, explore hybrid generative–statistical approaches, and assess their integration in large-scale language models for orthographic error correction.

8. Acknowledgements

The authors thank the reviewers for their valuable comments and suggestions, which contributed to improving the quality of the manuscript and to clarifying its scope and limitations. This publication results from work undertaken within the framework of the LEIA (*Lengua Española e Inteligencia Artificial*) project, promoted by the Spanish Ministry of Economic Affairs and Digital Transformation and by the Recovery, Transformation, and Resilience Plan, funded by the European Union - NextGenerationEU.

9. Bibliographical References

Christopher Bryant, Zheng Yuan, Muhammad Reza Qorib, Hannan Cao, Hwee Tou Ng, and Ted Briscoe. 2023. Grammatical error correction: A survey of the state of the art. *Computational Linguistics*, 49(3):643–701.

Manuel Castillo-Sancho. 2025. Modelado de erratas para el entrenamiento de un verificador ortográfico en español. Master's thesis, Universidad Politécnica de Madrid, Madrid, Spain.

Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176.

Michael Flor, Yoko Futagi, Melissa Lopez, and Matthew Mulholland. 2015. Patterns of misspellings in L2 and L1 English: a view from the ETS Spelling Corpus. *Bergen Language and Linguistics Studies*, 6.

S. Kullback and R. A. Leibler. 1951. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707.

Jiahua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151.

Jiawei Liu, Min Huang, and Qinghai Miao. 2024. Mitigating spurious correlations in named entity recognition models through counterfactual data augmentation. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Edward Ma. 2019. Nlp augmentation. https://github.com/makcedward/nlpaug.

Nikita Martynov, Mark Baushenko, Alexander Abramov, and Alena Fenogenova. 2023. Augmentation methods for spelling corruptions. In *Proceedings of the International Conference "Dialogue 2023"*, volume 2023.

Mehryar Mohri. 2002. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350.

Mehryar Mohri. 2009. *Weighted Automata Algorithms*, pages 213–254. Springer Berlin Heidelberg, Berlin, Heidelberg.

Lance A. Ramshaw and Mitchell P. Marcus. 1995. Text Chunking Using Transformation-Based Learning. In *Third Workshop on Very Large Corpora*, pages 82–94.

Wenqian Ye, Luyang Jiang, Eric Xie, Guangtao Zheng, Yunsheng Ma, Xu Cao, Dongliang Guo, Daiqing Qi, Zeyu He, Yijun Tian, Megan Coffee, Zhe Zeng, Sheng Li, Ting-hao, Huang, Ziran Wang, James M. Rehg, Henry Kautz, and Aidong Zhang. 2025. The clever hans mirage: A comprehensive survey on spurious correlations in machine learning.

Can Özbey, Hatice Altınok, and Mustafa Umut Demirezen. 2022. A Novel Probabilistic Framework for Modeling Spelling Errors by Term Length and Frequency. In *2022 3rd International Informatics and Software Engineering Conference (IISEC)*, pages 1–6.

Large Language Model-Based Post-OCR Correction for Low-Resource Kazakh Scripts

Henry Gagnier
Pittsford Sutherland High School
Pittsford, NY, USA
henrygagnier9@gmail.com

Abstract

Kazakh is written in the Arabic, Cyrillic, and Latin script which present unique challenges for OCR and post-OCR correction research. Despite this complexity, NLP research on Kazakh and its low-resource scripts remains extremely scarce. We analyze common OCR error patterns in all three Kazakh scripts using Tesseract and evaluate four large language models (LLMs) for post-OCR correction using minimal, confusion-aware, and few-shot prompting strategies. Our results reveal three systematic, writing-system-driven failure modes in LLM-based post-OCR correction: script switching, hallucination, and instruction-following breakdown. Arabic script post-OCR correction remains unsuccessful across all setups. In the Cyrillic script, post-OCR correction improvements are minimal due to the high baseline OCR performance on Cyrillic. For the Latin script, few-shot prompting with Gemini 2.5 Flash yields substantial improvements, reducing CER by 8.58 points and WER by 32.49 points to levels better than high-resource Kazakh Cyrillic script OCR. These findings demonstrate that LLM post-OCR correction failure modes are predictable from writing system properties such as script resource asymmetry and co-existing script dominance and demonstrate the need for typology-aware evaluation frameworks for multi-script and under-resourced languages.

Keywords: OCR, Kazakh, post-OCR correction, low-resource scripts, writing system typology

1. Introduction

Kazakh is a rare language in the sense that three different scripts are used for Kazakh writing, making optical character recognition (OCR) challenging. Arabic, Cyrillic, and Latin scripts are each used in Kazakh in different geographic regions (Honkasalo and Temirbekova, 2024). Due to this script diversity, natural language processing (NLP) tools and research have focused predominantly on Kazakh Cyrillic, leaving the Arabic and Latin scripts significantly underrepresented.

Kazakh has undergone multiple script transitions driven by political and cultural forces. Before Soviet rule, Kazakh was written in the Arabic script (Honkasalo and Temirbekova, 2024; Batyrbekkyzy et al., 2018). Under Soviet policy, in 1928, Kazakh was changed to the Latin script for the first time, and in 1940, Kazakh switched from the Latin to the Cyrillic script, which is currently the dominant script in practice for Kazakh (Batyrbekkyzy et al., 2018). Currently, in China, Kazakh is written using a modified Arabic alphabet, which is taught in schools. People also use an informal Latin script in casual contexts, differing from official Latin orthographies. In 2017, the first proposal for a new Latinized alphabet for Kazakh was created (Honkasalo and Temirbekova, 2024). Initially, a switch to the Latin script in Kazakhstan was designated for completion in 2025, but this timeline has been postponed to 2031. The majority of Kazakhstan residents believe that the Latin script is the best script for writing Kazakh, although most residents do not use the Latin script in practice (Honkasalo and Temirbekova, 2024). The sociolinguistic history of Kazakh shapes the asymmetry in script NLP resources, with current work and resources being predominantly focused on the Cyrillic script (Gagnier et al., 2026). This complex digraphic state of Kazakh must be acknowledged and further researched in the context of NLP.

Optical character recognition (OCR) is the process of digitizing a document image into its constituent characters (Bunke and Wang, 1997). Many texts resulting from OCR are noisy and need to be post-corrected. Post-correction can be done manually, using isolated-word approaches, feature-based machine learning models, sequence-to-sequence models, and language models (Nguyen et al., 2021). Prior work has shown that general-purpose OCR tools are not robust to data-scarce endangered language settings, and that post-correction methods must be tailored accordingly (Rijhwani et al., 2020). Large language models (LLMs) have recently emerged for post-OCR error correction in high-resource languages (Koynov and Doan, 2025). Danilova and Aangenendt (2025) found that German Mistral enhanced the OCR output, decreasing word error rate (WER) and character error rate (CER) remained unchanged or decreased. Kanerva et al. (2025) found that LLMs show promise for reducing CER in English, while in Finnish, a practically useful performance was not reached. LLM-based

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 79–88
12 May, 2026.

post-OCR correction is an emerging and important task, but even in well-resourced languages like Finnish, it is difficult, suggesting great challenges in under-resourced settings such as low-resource Kazakh scripts.

Work in Kazakh OCR has primarily focused on the Cyrillic script, reflecting the abundance of digitized Kazakh Cyrillic script resources compared to the Arabic and Latin scripts. Yeleussinov et al. (2023) worked on improving Kazakh OCR accuracy in the Cyrillic script using generative adversarial network (GAN) models. Nurseitov et al. (2021) and Toiganbayeva et al. (2022), respectively, created HKR and KOHTD, which are both large datasets in the Cyrillic script for handwritten text recognition. Kamshat et al. (2024) surveyed OCR, NLP, and speech recognition techniques for Kazakh as a low-resource language, reporting an OCR model achieving 85% accuracy. The Kazakh Arabic script and the Kazakh Latin script both use additional letters, modified graphemes, and different orthographic conventions, making it different from high-resource languages that use the Arabic and Latin scripts. Recently, the low-resource Latin and Arabic have been studied. (Gagnier et al., 2026) constructed KazakhOCR, a synthetic benchmark for OCR in Kazakh in all three Kazakh scripts, and found that models have high error rates in the Kazakh Arabic and Latin scripts. To our knowledge, this is the first work on post-OCR correction for Kazakh in any script, and work in OCR for low-resource Kazakh scripts is emerging.

Work on Kazakh OCR and post-OCR correction is necessary but very scarce, especially for low-resource Kazakh scripts. We present four main contributions in this paper: (1) the first study of post-OCR correction across all three Kazakh writing systems, (2) LLM-based post-OCR correction is unreliable for low-resource and multi-script settings, even with confusion-aware prompting, (3) script switching and hallucination are writing-system-driven failure modes not captured by standard OCR metrics, and (4) few-shot prompting can overcome challenges in Latin Kazakh, but does not generalize across scripts. The purpose of this paper is to (1) identify issues and challenges in Kazakh OCR and (2) evaluate LLMs for Kazakh post-OCR correction. We also aim to broaden script coverage in Kazakh NLP and encourage future work on the computational modeling of under-resourced writing systems.

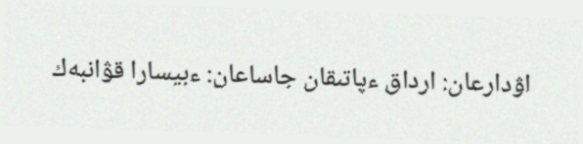


Figure 1: Example image from the KazakhOCR benchmark in the Arabic script

2. KazakhOCR

We use the KazakhOCR benchmark[1] (Gagnier et al., 2026), which consists of 7,219 synthetically created images for Kazakh OCR in all three Kazakh scripts with variations to increase authenticity and representativeness of tasks. Texts are typically medium-length, with average lengths of all texts between 700 and 750 characters in each script. In this benchmark, multimodal large language models have CERs of 0.355 to 0.725, 0.216 to 0.310, and 0.053 to 0.257 in Arabic, Latin, and Cyrillic scripts, respectively, while Tesseract has CERs of 0.150, 0.114, and 0.043 in Arabic, Latin, and Cyrillic scripts, respectively. As traditional OCR approaches perform significantly better than current multimodal large language models or MLLMs (particularly Gemma-3-12B-it, Qwen2.5-VL-7B-Instruct, and Llama-3.2-11B-Vision-Instruct) for Kazakh OCR (Gagnier et al., 2026), we choose to use Tesseract for experiments in this paper.

The three Kazakh scripts differ in visual form and typological properties, which make OCR challenging for Kazakh. The Kazakh Arabic script is an abjad script in which short vowels are often omitted and represented by diacritic marks (Daniels and Bright, 1996), creating character-level ambiguity that increases OCR errors and limits post-correction models. The Kazakh Cyrillic script is an alphabet adapted to the Kazakh phonology, including letters not present in Russian. The Kazakh Latin script is a script that encodes Kazakh phonemes through dense diacritics, creating a high density of marks that are fragile with OCR and poorly represented in NLP data. These three scripts are not equally represented in digital text, as Cyrillic Kazakh makes up the majority of text and LLM training data, while Latin and Arabic Kazakh are under-resourced. This asymmetry created LLM biases across Kazakh scripts and is an essential context for work on Kazakh in NLP and interpreting post-OCR correction results.

[1] https://huggingface.co/datasets/henrygagnier/kazakh-ocr

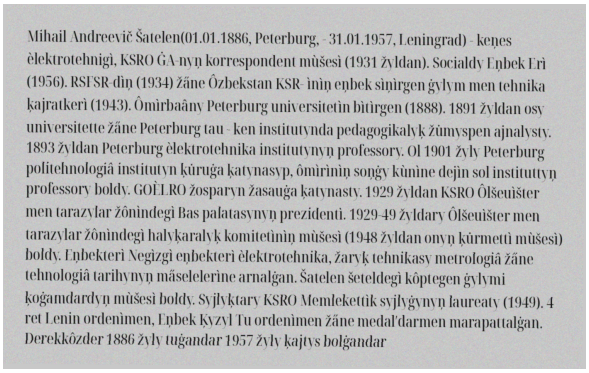


Figure 2: Example image from the KazakhOCR benchmark in the Latin script

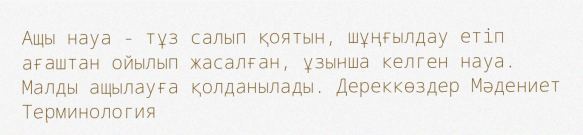


Figure 3: Example image from the KazakhOCR benchmark in the Cyrillic script

3. Methodology

We perform OCR using Tesseract, analyze script-specific OCR error patterns, and evaluate large language model (LLM)-based post-OCR correction using multiple prompting strategies.

3.1. Optical Character Recognition

We use Tesseract (Smith, 2007) for OCR in this paper. While Tesseract supports Arabic, Cyrillic, and Latin script recognition through separate trained models, it provides a Kazakh-specific language model only for the Cyrillic script, with no Kazakh-specific support for the Arabic or Latin scripts. To enable consistent comparison across all three scripts under equivalent conditions, we conduct OCR using the generic script-level configurations throughout on 200 images in each script (600 images total). Tesseract adds new lines based on the position of the text in the image, so we remove all new lines from the text before further experiments. Prior to all metric computation, we verified that all ground truth and Tesseract output strings are already in NFC form, as Unicode normalization had no effect on reported metrics. We use 150 images in each script for detecting common errors in Kazakh OCR and developing confusion-aware prompts, and we use 50 images in each script as a test set for LLM-based post-OCR correction.

3.2. Post-OCR Correction

Recently, LLMs have shown promise for post-OCR error correction in high-resource languages (Thomas et al., 2024). We evaluate four LLMs for post-OCR correction: GPT-4o-mini (OpenAI, 2024), DeepSeek v3.2 (DeepSeek-AI, 2025), Claude 3.5 Haiku (Anthropic, 2024), and Gemini 2.5 Flash (Google DeepMind, 2025). In all models, we set the `temperature` to 0.0. All 50 test images per script are submitted for post-correction regardless of OCR output quality, reflecting a realistic deployment scenario. We evaluate all models on post-OCR correction using three different prompt types. We provide the exact prompt text below.

- **Minimal Prompt:** In the minimal prompt, we provide a short instruction to correct OCR errors. We also provide the language and script of the text, and instruct the model to return the text in the original script.

- **Confusion-Aware Prompt:** Using our error analysis, we develop a prompt informing the model of common OCR errors, including deletions, substitutions, and insertions. We again provide the language and script of the text and instruct the model to return the text in the original script.

- **Few-shot Prompt:** We give the model three randomly chosen examples of an OCR output in the same script and the corresponding ground truth correction and instruct the model to correct OCR errors.

Minimal Prompt

You are correcting OCR errors in Kazakh text written in {script} script.
Original OCR output: {ocr_output}
Fix spelling and OCR recognition errors while preserving the original meaning. Return only the corrected text in the {script} script.

Arabic Script Confusion-Aware Prompt

You are correcting OCR errors in Kazakh text written in Arabic script.

Task: Correct OCR errors while preserving the original meaning, word boundaries, and punctuation usage appropriate for Kazakh.

Observed OCR error patterns in this dataset: Character substitutions: - ’،’ → ’.’, ’,’, ’»’, ’›’ - ’ى’ → ’ی’ or deleted - ’ي’ → ’ی’ - ’ك’ → ’ک’ - ’ہ’ → ’ه’ - ’پ’ → ’ب’

Frequently deleted characters: - ,’ى’ ’ ’, ,’ا’ ,’ن’ ’ه’

Frequently inserted characters: - ’ ’, ,’ه’ ,’ى’ ’ى’

Original OCR output: {ocr_output}

Return only the corrected Kazakh text in the Arabic script. Do not include explanations or metadata.

Latin Script Confusion-Aware Prompt

You are correcting OCR errors in Kazakh text written in Latin script.

Task: Correct character-level OCR errors while preserving the original meaning, word boundaries, and punctuation usage appropriate for Kazakh.

Observed OCR error patterns in this dataset: Character substitutions: - ’ì’ → ’i’ - ’k’ → ’ķ’ - ’n’ → ’ŋ’ - ’a’ → ’á’ - ’ģ’ → ’g’ - ’u’ → ’ú’, ’ů’ - Combining marks (◌̈, ◌̋, ◌̋) removed or altered

Frequently deleted characters: - ◌̈, ’ ’, ◌̋, ◌̋, ’u’

Frequently inserted characters: - ’i’, ’ ’, ’ķ’, ’ŋ’, ’g’

Original OCR output: {ocr_output}

Return only the corrected Kazakh text in the Latin script. Do not include explanations or metadata.

Cyrillic Script Confusion-Aware Prompt

You are correcting OCR errors in Kazakh text written in Cyrillic script.

Task: Correct OCR errors while preserving the original meaning, word boundaries, and punctuation usage appropriate for Kazakh.

Observed OCR error patterns in this dataset: Character substitutions: - ’ғ’ → ’г’ - ’ұ’ → ’ү’, ’у’ - ’ө’ → ’о’ - ’ң’ → ’ц’, ’н’ - ’н’ → ’п’ - ’ж’ → ’с’ - ’—’ → ’–’

Frequently deleted characters: - ’ ’, ’З’, ’—’, ’.’, ’Г’

Frequently inserted characters: - ’ ’, ’.’, ’ə’, ’с’, ’,’

Original OCR output: {ocr_output}

Return only the corrected Kazakh text in the Cyrillic script. Do not include explanations or metadata.

Few-Shot Prompt

Correct OCR errors in Kazakh {script} text. Here are examples:

{examples}

Now correct this text and return it in the {script} script: OCR: {ocr_output} Correct:

We apply the Wilcoxon signed-rank test to determine if differences between the baseline OCR text and the post-corrected text are significant, as this test is appropriate for small paired samples. We also compute the 95% bootstrap confidence interval on the mean improvement to quantify effect size and uncertainty. Each bootstrap resample draws 50 pairs with replacement from these observations and computes the mean difference, and repeating this 10,000 times yields an empirical distribution from which the 2.5th and 97.5th percentiles form the confidence interval.

4. Results

4.1. Optical Character Recognition

4.1.1. Tesseract Performance

We first look at the performance of Tesseract (Table 1), which presents the baseline OCR performance across all three Kazakh scripts. Tesseract achieves the best metrics in the Cyrillic script with a CER of 0.029 and a WER of 0.090. Performance is substantially worse in the Latin and Arabic scripts. In the Latin script, Tesseract has a CER of 0.102, and in the Arabic script, Tesseract has a CER of 0.142. These results highlight that the Arabic and Latin scripts require post-OCR correction for usability and that significant challenges remain in Kazakh OCR. All experiments use script-level rather than language-specific Tesseract configurations to enable consistent comparison across all three scripts, as no Kazakh-specific models exist for the Arabic and Latin scripts.

Script	CER	WER
Arabic	0.142	0.376
Latin	0.102	0.399
Cyrillic	0.029	0.090

Table 1: OCR Performance metrics Tesseract across different low-resource Kazakh scripts. CER = Character Error Rate, WER = Word Error Rate

4.1.2. Common OCR Errors

We now look at the most common substitutions, insertions, and deletions in Kazakh OCR by Tesseract generated from a subset of 150 images in each script (Tables 2 and 3). We use this analysis to inform models in confusion-aware prompts.

In the Kazakh Arabic script, errors are primarily from confusion between visually similar characters and punctuations, such as comma variants and substitutions involving ى (Alef Maksura, U+0649), ي, and ى. Deletions were frequent for ى (Alef Maksura, U+0649) and spaces, indicating that there is difficulty in preserving word segmentation and character identity. Deletions were also much more common than insertions.

Kazakh Cyrillic script errors involve confusion between Kazakh-specific letters and Russian letters (e.g., ғ → г, ө → о, ң → н), likely due to our use of a non-language-specific Cyrillic OCR model. While insertion and deletion counts are similar to the other two scripts, substitution counts in Cyrillic are substantially lower than both Arabic and Latin. Spaces are the most common character to be inserted and deleted in Cyrillic.

For the Latin Kazakh script, the most prominent errors involve diacritics and combining characters. Characters such as ◌̧ (combining comma below, U+0326), ◌̇ (combining dot above, U+0307), and accented vowels are frequently deleted or substituted, often collapsing distinct Kazakh letters into their unaccented forms. This leads to a high WER despite moderate CER, as diacritic loss frequently changes word identity.

4.2. Post-OCR Correction

We now look at the result of the LLM-based post-OCR correction (Table 4). We look at changes in CER and WER after applying post-OCR correction. Negative values indicate degradation or a worse OCR output, and positive values indicate reductions in errors.

4.2.1. Arabic Script

For the Arabic script, LLM-based post-OCR correction generally degrades OCR output across models and prompting strategies. Almost all setups resulted in a degradation in performance, with no setup achieving an improvement in both CER and WER.

A notable improvement of 6.79 points in WER was observed using few-shot prompting with Gemini; however, the associated change in CER was a degradation of 6.40 points, making the overall result inconsistent. Improvements in performance were very inconsistent, and despite four different models and three different prompting strategies, we were unable to reach a practically useful performance. In some cases, the resulting text was substantially worse than the input. CER degraded 73.73 points, and WER degraded 64.20 points using GPT-4o-mini with the confusion-aware prompt.

4.2.2. Cyrillic Script

Given the strong baseline OCR performance for Cyrillic Kazakh, most LLM configurations either fail to improve results or slightly degrade performance. Some models were found to have slight increases in performance.

In select cases, performance in CER and WER increased from the baseline. Using Gemini and few-shot prompting, an improvement of 0.40 in CER and 1.54 in WER was observed. In Gemini 2.5 Flash, performance improved in all three prompting setups for Cyrillic, with few-shot prompting being the most effective, and the minimal prompt being the least effective. This suggests that confusion-aware prompting may provide marginal improvements to minimal prompts. This finding also suggests that some models have much greater capabilities for Kazakh post-OCR

Arabic			Cyrillic			Latin		
#	Substitution	Count	#	Substitution	Count	#	Substitution	Count
1	، → .	474	1	ғ → г	103	1	ì → i	969
2	ى → ی	363	2	ұ → у	91	2	k → ķ	618
3	ي → ی	223	3	ө → о	83	3	n → ņ	387
4	ك → ک	211	4	— → –	54	4	a → á	233
5	ہ → ه	202	5	н → п	48	5	ģ → g	177
6	، → ,	166	6	ж → с	47	6	u → ú	130
7	، → »	110	7	ң → ц	42	7	◌̦ (U+0326) → ú	124
8	، → ›	104	8	ң → н	39	8	u → ů	101
9	ى → □	83	9	ы → Ы	31	9	□ → ’	93
10	پ → ب	78	10	ұ → у	28	10	◌̦ (U+0326) → ů	91

Table 2: Top 10 character substitutions by Tesseract OCR across Kazakh scripts. The arrow indicates the ground-truth character (left) incorrectly recognized as the OCR output (right). Combining diacritics are identified by Unicode name: ◌̦ = combining comma below (U+0326); ◌̇ = combining dot above (U+0307); ◌̋ = combining double acute accent (U+030B).

Arabic			Cyrillic			Latin		
Top 5 Deleted Characters								
#	Char	Count	#	Char	Count	#	Char	Count
1	ى (U+0649)	1,252	1	□	1,299	1	◌̦ (U+0326)	2,960
2	□	1,046	2	З	17	2	□	1,236
3	ا	541	3	—	17	3	◌̇ (U+0307)	625
4	ن	488	4	.	14	4	◌̋ (U+030B)	529
5	ه	451	5	Г	11	5	u	275
Top 5 Inserted Characters								
#	Char	Count	#	Char	Count	#	Char	Count
1	□	153	1	□	127	1	i	216
2	ه	40	2	.	24	2	□	55
3	ى (U+0649)	30	3	ə	23	3	ķ	48
4	U+200E (LRM)	18	4	c	15	4	ņ	32
5	ی	18	5	,	14	5	g	12

Table 3: Top 5 deleted and inserted characters by Tesseract OCR across Kazakh scripts. Deleted characters are missing from OCR output; inserted characters are spuriously added. Combining diacritics are identified by Unicode name: ◌̦ = combining comma below (U+0326); ◌̇ = combining dot above (U+0307); ◌̋ = combining double acute accent (U+030B).

correction, given the consistent improvement by Gemini, and the consistent degradation by GPT-4o-mini. In some cases, such as Claude with few-shot and minimal prompting, performance decreased dramatically despite Claude’s success in confusion-aware prompting. This suggests that models may be highly sensitive to different prompts.

4.2.3. Latin Script

Results for the Latin script are mixed but predominantly negative. Some setups have significant improvements in WER and CER from the baseline of 0.102 and 0.399.

Performance decreased in most models, with the decrease often being extremely substantial. Using the minimal and confusion-aware prompts, we find that CER performance degrades between 2.20 and 26.21 points, and WER performance degrades between 9.45 and 30.71 points. Conversely, using the few-shot prompts, performance increases dramatically in almost all models. Using GPT-4o-mini, Gemini 2.5 Flash, and DeepSeek v3.2, CER performance increased between 1.80 and 8.58 points, and WER performance increased between 7.39 and 32.49 points. Ultimately, the improvements of Gemini 2.5 Flash improve CER and WER to 0.016 and 0.074, to levels even lower than the initial Kazakh Cyrillic OCR. This few-shot setup makes the Latin script OCR outputs usable for downstream tasks, but is highly sensitive to the use of few-shot prompts.

Script	Model	Prompt	△CER	△WER
Arabic	GPT-4	minimal	-0.5321*	-0.4580*
Arabic	GPT-4	confusion-aware	-0.7373*	-0.6420*
Arabic	GPT-4	few-shot	-0.5182*	-0.4254*
Arabic	Claude	minimal	-0.3185*	-0.3326*
Arabic	Claude	confusion-aware	-0.1927*	-0.2098*
Arabic	Claude	few-shot	-0.4411*	-0.4185*
Arabic	Gemini	minimal	-0.0574	+0.0166
Arabic	Gemini	confusion-aware	-0.1974	-0.1052
Arabic	Gemini	few-shot	-0.0640	**+0.0679**
Arabic	DeepSeek	minimal	-0.0569*	-0.2024*
Arabic	DeepSeek	confusion-aware	**-0.0357***	-0.1602*
Arabic	DeepSeek	few-shot	-0.0499*	-0.0815*
Cyrillic	GPT-4	minimal	-0.0028	-0.0389*
Cyrillic	GPT-4	confusion-aware	-0.0020	-0.0248
Cyrillic	GPT-4	few-shot	-0.0001	-0.0179
Cyrillic	Claude	minimal	-0.1818*	-0.2341*
Cyrillic	Claude	confusion-aware	+0.0012	+0.0030
Cyrillic	Claude	few-shot	-0.6464*	-0.7371*
Cyrillic	Gemini	minimal	+0.0019	+0.0033
Cyrillic	Gemini	confusion-aware	+0.0021	+0.0054
Cyrillic	Gemini	few-shot	**+0.0040***	**+0.0152**
Cyrillic	DeepSeek	minimal	-0.0272*	-0.0748
Cyrillic	DeepSeek	confusion-aware	-0.0074	-0.0202
Cyrillic	DeepSeek	few-shot	+0.0031	+0.0074
Latin	GPT-4	minimal	-0.1083*	-0.1710*
Latin	GPT-4	confusion-aware	-0.0220*	-0.0945*
Latin	GPT-4	few-shot	+0.0303*	+0.1163*
Latin	Claude	minimal	-0.1982*	-0.2660*
Latin	Claude	confusion-aware	-0.2621*	-0.3018*
Latin	Claude	few-shot	-0.1855	-0.0514
Latin	Gemini	minimal	-0.2237*	-0.3071*
Latin	Gemini	confusion-aware	-0.2163*	-0.2898*
Latin	Gemini	few-shot	**+0.0858***	**+0.3249***
Latin	DeepSeek	minimal	-0.1179*	-0.2880*
Latin	DeepSeek	confusion-aware	-0.0804*	-0.2620*
Latin	DeepSeek	few-shot	+0.0180	+0.0739

Table 4: Change in character error rate (△CER) and word error rate (△WER) after LLM-based post-OCR correction across scripts. Negative values indicate degradation with respect to the original OCR output. *Statistically significant change (Wilcoxon signed-rank test, $p < 0.05$, with 95% bootstrap confidence intervals not crossing zero).

4.3. Error Analysis

To complement the quantitative analysis and exemplify major errors, we present a short qualitative analysis of errors made by LLMs when correcting Kazakh text (Table 5).

- **Script change:** In cases when CER and WER decreased dramatically, errors often resulted from the LLM not preserving the original script and transliterating the text to Cyrillic Kazakh. In example 1, it can be seen that the Latin OCR output is changed to Cyrillic text. In example 2, it can be seen that the Arabic script is mixed with the Cyrillic script in the LLM correction. This behavior accounts for some of the largest single-setup degradations in our results. Script switching towards the dominant Kazakh script is a predictable consequence of resource asymmetry in low-resource languages, a rare typological configuration when multiple orthographies coexist for a single language at extremely unequal levels of representation. In Kazakh, Cyrillic text dominates online corpora and LLM training data significantly, creating an association with “Kazakh” and the Cyrillic script in current language technologies. When models encounter a low-resource script with an unfamiliar orthography, it often overrides explicit instructions, producing outputs in the better-known script of the language. This is an important writing-system-specific failure that

has no direct comparison in monoscript post-OCR correction research.

- **Hallucination/overcorrection:** In some cases, LLMs made frequent incorrect corrections, even in confusion-aware settings. In examples 3 and 4, k is often corrected to q despite a q → k substitution not mentioned in the Latin script confusion-aware prompt. These rewrites may partly reflect influence from Turkish, which uses a similar Latin-based orthography and is substantially better represented in LLM training data than Kazakh Latin; models may default to Turkish orthographic patterns when Kazakh-specific knowledge is absent. More broadly, this pattern reflects the lack of orthographic knowledge that current LLMs have of Kazakh Latin specifically. Kazakh Latin uses a unique orthography with many unique diacritics differing from other high-resource languages using the Latin script. This means that models cannot use learned orthographic patterns and instead apply superficial substitutions based on high-resource co-script languages. This is a hallucination driven by the lack of script knowledge by LLMs.
- **English Text Insertion**: Surprisingly, Cyrillic OCR performance decreased dramatically in Claude. We found that Claude frequently inserted English text explaining its changes, despite the prompt asking the LLM to "return only the corrected text." We did not find this issue in other models.

Both of these errors are prevalent across the Latin and Arabic scripts and account for the dramatic decreases in CER and WER in some setups.

5. Discussion

We identify common errors in the OCR of all three Kazakh scripts and evaluate four LLMs for post-OCR error correction using three major prompt types. Our evaluation reveals that LLM-based post-OCR correction in multi-script settings fails in structured, characterizable ways that go beyond simple performance degradation. Script switching driven by resource imbalance, hallucination driven by insufficient script knowledge, and instruction-following breakdown under constrained output conditions each map onto distinct properties of the Kazakh writing system context, suggesting that post-OCR correction for multi-script languages and under-resourced writing systems requires evaluation frameworks and prompting strategies that account for script dominance hierarchies. Performance varies dramatically across scripts. In the Arabic script, post-OCR correction was unsuccessful, frequently resulting in great increases in CER and WER, demonstrating that LLM-based post-OCR correction is unreliable when the target language is severely underrepresented in LLM training data relative to other languages sharing the same script. In the Cyrillic script, LLMs provided marginal improvements and, in some cases, introduced new errors. The Latin script presents a promising case for post-OCR correction where models using few-shot prompting achieve substantial reductions in WER and CER, making their output suitable for downstream NLP tasks and usage.

These findings align with previous work showing that LLM-based post-OCR correction is less reliable in under-resourced scripts and languages compared to high-resource languages (Kanerva et al., 2025). Low-resource Kazakh scripts lack full support in LLMs, limiting model performance on Kazakh tasks. These findings conversely show promise in few-shot methods for the Kazakh Latin script and provide few-shot methods as a future direction for low-resource post-OCR correction.

We find that some models frequently fail to preserve the original script, often transliterating or partially converting text into Cyrillic Kazakh even when instructed not to do so. This behavior accounts for some of the largest degradations in CER and WER and exemplifies a limitation of current LLMs in multi-script settings. This difficulty is rare due to Kazakh's use of three scripts and must be considered in future systems using any low-resource Kazakh script. Hallucination and overcorrection played roles in the degradation of the Latin script, where models introduced corrections that were not supported by error patterns. These errors suggest that current LLMs may rely on incorrect assumptions about Kazakh. The insertion of English text by Claude when instructed not to do so was unexpected and displays the fragility of instruction-following in LLMs for text generation tasks such as post-OCR correction.

Future work should develop post-OCR correction methods to prevent script changes and overcorrection of texts. Future work should develop new post-OCR correction systems, such as hybrid approaches that combine rule-based filtering with LLM generation. Correction systems should be studied with other OCR systems other than Tesseract, as MLLMs have significantly more errors, and other models may also produce outputs with differing amounts of error from Tesseract. Future work could also include supervised or instruction-tuned post-OCR correction models trained specifically on Kazakh scripts. Work should also look into the impact of structured outputs such as JSON, which may result in more adherence to instructions because many models have been instruction-tuned

#	Ground Truth	OCR Output	LLM Correction	Error
1	Orys últy̧k-mắdeni ortalyġy, Taldykorġan k̦alasynda 1998 ž. k̦ùrylġan. Ortalyk̦tyn̦ baġdarlamasynda orys halk̦ynyn̦ saltdắstùrìn, tìlìn, mắdenietìn damytyp, sak̦tau kôzdelgen...	Orys últyk-mőűdeni ortalygy, Taldykorġan kalasynda 1998 Žž kúrylġan. Ortalyktyn baġdarlamasynda orys halkynyn̦ soltdőstùrìn, tiin, mődenietin damytyp, saktau kôzdelgen…	Орыс ұлттық-мәдени орталығы Талдықорған қаласында 1998 жылы құрылған. Орталықтың бағдарламасында орыс халқының салт-дәстүрін, тілін, мәдениетін дамытып, сақтау көзделген...	Script change (Latin → Cyrillic)
2	ايدىڭ 3-كۇنى، اتقا منگەن ساياحاتشىلار قابا اۋدانىنىڭ جايلاۋىن ساياحاتتاۋدا. قابا اۋدانىنىڭ جايلاۋىنىڭ ەكولوگيالىق ورتاسى جاقسى،...	ايدىڭ 3-كۇنى. اتقا منگەن ساياحاتشىلار قابا اۋدانىنىڭ جايلاۋىن ساياحاتتاۋدا. قابا اۋدانىنىڭ جايلاۋىنىڭ ... ەكولوگيالىق ورتاسى جاقسى. ءار	ылар ئىيىڭ 3-كۇنى. اتقا مىگگەن ساياحاتچ قابا.тап ساياحاتында نىڭ جايلاۋىданыقابا ئاۋ ئىكولوگىيالىق ۋورتاسىның جايلاۋىданынныңئاۋ جاقсы...	Partial script change (Arabic → Cyrillic)
3	..Taldykorġan k̦alasynda 1998 ž. k̦ùrylġan...	...Taldykorġan kalasynda 1998 Žž kúrylġan...	...Taldyqorghan qalasynda 1998 jyl qurylgan...	Over correction/hallucination
4	...Abaj oblysy Aâgôz audanyndaġy auyl...	...Abaj oblysy Aâgôz audanyndaġy auyl...	...Abai oblysy Aiağöz audanyndağy auyl...	Over correction/hallucination
5	N/A	N/A	(Note: I did not find any obvious OCR errors in this text that require correction.)	English text insertion

Table 5: Representative examples of LLM errors in Kazakh text correction.

for structured output. MLLMs for post-OCR correction should be explored as they may prevent overcorrection and hallucination, and their output quality may benefit from LLM-based post-OCR correction strategies such as few-shot learning.

LLM-based post-OCR correction for Kazakh is largely unreliable, but under certain setups, it can perform extremely well. Arabic-script correction remains a challenge, and Cyrillic-script correction offers limited benefits due to the strong baseline OCR performance. In the Latin script, we develop a setup to decrease CER by 8.58 points and WER by 32.49 points using a few-shot setup with Gemini. These findings display the promise and the limitations of LLMs for post-OCR correction in under-resourced and multi-script languages.

6. Conclusion

This study presents the first evaluation of LLMs for post-OCR correction in all three Kazakh scripts. We evaluate Gemini 2.5 Flash, GPT-4o-mini, DeepSeek v3.2, and Claude 3.5 Haiku using minimal, confusion-aware, and few-shot prompting.

We first identify common OCR errors and causes in Kazakh, which we use to inform prompt development. We find that LLMs are largely unsuccessful under most setups, often switching to incorrect scripts, overcorrecting, or incorrectly following instructions. In the Arabic script, we are unable to find a successful model, and in the Cyrillic script, the best models make marginal improvements due to the initial high accuracy of Kazakh Cyrillic OCR. In the Latin script, we find that few-shot prompting is highly effective, decreasing the CER by 8.58 points and WER by 32.49 points, to levels lower than the high-resource Cyrillic script, when using Gemini 2.5 Flash.

These findings display the promise and variability of LLMs for post-OCR correction in low-resource languages and scripts such as Kazakh. These findings contribute to advancing low-resource and Kazakh NLP research, building post-OCR correction models, and supporting the inclusion of low-resource writing systems in NLP.

7. Limitations

There are several limitations that should be considered in this study. First, authentic OCR data does not exist for the Latin and Arabic Kazakh scripts, so all OCR experiments rely on the synthetically generated KazakhOCR benchmark, which may not capture the full diversity of layouts, fonts, and handwriting styles found in real-world documents. Synthetic data is also more controlled, allowing for results to reflect script, model, and prompt differences rather than document-level variation. Second, the evaluation is limited to Tesseract due to its strong previous performance on the benchmark. Different OCR systems may have different error levels, which could affect post-OCR correction performance. Third, we evaluate only zero-shot and few-shot prompting without any model fine-tuning to reflect a realistic and accessible use case. This approach may not capture the potential gains achievable through supervised or instruction-tuned post-OCR correction models. Finally, the test set used consisted of 50 images per script, which is smaller than ideal for evaluation. This was constrained by API costs of running four models across three scripts and three prompting strategies. We applied two statistical tests, and we marked results that were significant in the results to mitigate incorrect findings.

8. Bibliographical References

References

Anthropic. 2024. The Claude 3 model family: Opus, Sonnet, Haiku.

G. Batyrbekkyzy, T.K. Mahanuly, M.M. Tastanbekov, L.S. Dinasheva, B.K. Issabek, and G.D.

Sugirbayeva. 2018. Latinisation of kazakh alphabet history and prospects. *European Journal of Science and Theology*, 14:125–134.

H Bunke and P S P Wang. 1997. Handbook of character recognition and document image analysis.

Peter T Daniels and William Bright. 1996. *The World's Writing Systems*. Oxford University Press.

Vera Danilova and Gijs Aangenendt. 2025. Post-OCR correction of historical German periodicals using LLMs. In *Proceedings of the Third Workshop on Resources and Representations for Under-Resourced Languages and Domains (RESOURCEFUL-2025)*, pages 120–129, Tallinn, Estonia. University of Tartu Library, Estonia.

DeepSeek-AI. 2025. DeepSeek-V3 technical report.

Google DeepMind. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities.

Sami Honkasalo and Tansulu Temirbekova. 2024. The writing reform and 'Latinization' of written Kazakh: A sociolinguistic survey. *International Journal of Eurasian Linguistics*, 6(1):48–80.

Asmaganbetova Kamshat, Ulanbek Auyeskhan, Nurzhanova Zarina, Serikbayev Alen, and Malika Yeskazina. 2024. Integration ai techniques in low-resource language: The case of kazakh language. In *2024 IEEE AITU: Digital Generation*, pages 7–13.

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.

Radoslav Koynov and Triet Ho Anh Doan. 2025. Opportunities and challenges of LLMs as post-OCR correctors. volume 45, pages 111–118. Polish Information Processing Society.

Thi Tuyet Hai Nguyen, Adam Jatowt, Mickael Coustaty, and Antoine Doucet. 2021. Survey of post-OCR processing approaches. *ACM Computing Surveys*, 54(6):1–37.

Daniyar Nurseitov, Kairat Bostanbekov, Daniyar Kurmankhojayev, Anel Alimova, Abdelrahman Abdallah, and Rassul Tolegenov. 2021. Handwritten Kazakh and Russian (HKR) database for text recognition. *Multimedia Tools and Applications*, 80(21–23):33075–33097.

OpenAI. 2024. GPT-4o system card.

Shruti Rijhwani, Antonios Anastasopoulos, and Graham Neubig. 2020. OCR Post Correction for Endangered Language Texts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5931–5942, Online. Association for Computational Linguistics.

R. Smith. 2007. An overview of the Tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 629–633.

Alan Thomas, Robert Gaizauskas, and Haiping Lu. 2024. Leveraging LLMs for post-OCR correction of historical newspapers. In *Proceedings of the Third Workshop on Language Technologies for Historical and Ancient Languages (LT4HALA) @ LREC-COLING-2024*, pages 116–121, Torino, Italia. ELRA and ICCL.

Nazgul Toiganbayeva, Mahmoud Kasem, Galymzhan Abdimanap, Kairat Bostanbekov, Abdelrahman Abdallah, Anel Alimova, and Daniyar Nurseitov. 2022. KOHTD: Kazakh offline handwritten text dataset. *Signal Processing: Image Communication*, 108:116827.

Arman Yeleussinov, Yedilkhan Amirgaliyev, and Lyailya Cherikbayeva. 2023. Improving OCR accuracy for Kazakh handwriting recognition using GAN models. *Applied Sciences*, 13(9):5677.

9. Language Resource References

Gagnier, Henry and Gagnier, Sophie and Kirubakaran, Ashwin. 2026. *KazakhOCR: A Synthetic Benchmark for Evaluating Multimodal Models in Low-Resource Kazakh Script OCR*. Association for Computational Linguistics.

G&P2P: A Multi-Source Approach to Grapheme-to-Phoneme Conversion

Chun-Yi Peng
Borough of Manhattan Community College, CUNY Graduate Center
199 Chambers St. New York, NY, 365 5th Ave. New York, NY
cpeng@bmcc.cuny.edu

Abstract

Grapheme-to-phoneme (G2P) conversion plays a central role in speech technologies. This paper introduces G&P2P, a multi-source framework that integrates multiple pronunciation dictionaries to enhance G2P modeling. We evaluate both expert-curated and crowd-sourced resources using attentive LSTM, pointer-generator LSTM, and transformer architectures. Results indicate that combining high-quality expert dictionaries yields substantial improvements, achieving an 11.26-point absolute (22% relative) reduction in word error rate. In contrast, incorporating noisy crowd-sourced resources may degrade performance. Statistical analyses further suggest that dataset quality exerts a greater influence on outcomes than the choice of fusion strategy, offering practical guidance for the design of multi-source G2P systems.

Keywords: grapheme-to-phoneme conversion, side-pronunciation, multi-source learning

1. Introduction

Grapheme-to-phoneme (G2P) conversion maps written word forms to their phonemic representations. It plays a crucial role in text-to-speech systems, automatic speech recognition, and low-resource language documentation. Despite significant progress in neural sequence modeling, G2P performance remains highly dependent on the quality and coverage of training lexicons.

Most existing systems are trained on a single pronunciation dictionary. However, multiple lexical resources often exist for a language, including expert-curated dictionaries and crowd-sourced repositories. These resources vary in coverage, transcription conventions, and annotation quality. While combining them may increase lexical coverage and improve out-of-vocabulary generalization, naive integration may introduce inconsistencies and noise.

This paper proposes G&P2P, a multi-source framework that integrates multiple pronunciation lexicons under controlled fusion strategies. The study investigates: 1) whether multi-source supervision improves G2P performance, 2) how dataset quality influences learning, 3) Whether fusion strategy significantly affects outcomes, 4) how LSTM and transformer architectures compare under multi-source training.

The results show that improvements primarily depend on dataset quality rather than fusion strategy, and that pointer-generator LSTMs outperform transformers in most settings.

2. Related Work

With the rise of deep learning, neural sequence models have since become dominant in G2P tasks (Ashby et al., 2021; Gorman et al., 2020; Kłosowski, 2022). Recurrent Neural Networks (RNNs) capture sequential dependencies (Bahdanau et al., 2015; Luong et al., 2015), and Long Short-Term Memory (LSTM) networks address vanishing gradient issues through gated memory mechanisms (Hochreiter and Schmidhuber, 1997; Eisenstein, 2019; Rao et al., 2015). Encoder–decoder LSTMs with attention further improved G2P by allowing dynamic alignment between input graphemes and output phonemes (Luong et al., 2015).

Transformer architectures (Vaswani et al., 2017) replaced recurrence with self-attention, enabling parallel computation and effective long-range modeling (Yolchuyeva et al., 2019b), though they often require larger datasets and greater computational resources (Yolchuyeva et al., 2019a,b). Evidence from the SIGMORPHON 2020 shared task suggests LSTM baselines can outperform transformers in medium-resource settings (Gorman et al., 2020). To better handle rare forms, pointer-generator architectures combine neural generation with explicit copying mechanisms (Vinyals et al., 2015; Prabhu and Kann, 2020; See et al., 2017), a design particularly well suited to G2P tasks.

However, prior work typically assumes a single lexicon source. Multi-source supervision remains underexplored in G2P research.

3. Data

3.1. Sources

The following English pronunciation resources were used for this study:

- **CELEX** (Baayen et al., 1995) transcribes English phonemes using DISC, where each

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 89–94
12 May, 2026.

phoneme is represented by exactly one ASCII character.

- **PronLex** (Kingsbury et al., 1994) provides ARPAbet transcriptions of Mainstream American English from the CALLHOME Project.
- **NETTalk** (Sejnowski and Rosenberg, 1988) transcribes English phonemes using a novel ASCII-based transcription system that assigns unique symbols or character combinations to represent different English sounds.
- **WikiPron** (Lee et al., 2020) provides crowdsourced pronunciation dictionaries from Wiktionary. This study uses separate dictionaries for British (WikiPron_UK) and American (WikiPron_US) English.

The first three are expert-curated and relatively consistent in transcription standards. WikiPron-US and WikiPron-UK offer broader coverage but contain heterogeneous and potentially noisy annotations. Table 1 shows the size of each dictionary.

Dataset	Size	Dataset	Size
CELEX	72,995	WikiPron_US	78,633
NETTalk	20,006	WikiPron_UK	79,520
PronLex	96,941		

Table 1: Size of each dictionary.

Sample transcriptions from these sources are shown in Table 2. Consonants are transcribed similarly across dictionaries, whereas the transcriptions of vowels vary. Take the word *lesson* for example, *on* is transcribed as 'H' in CELEX, 'ih0 n' in PronLex, 'N' in NETTalk, and 'ən' in Wiktionary.

Lexicon	Transcription of *lesson*
CELEX	' l E s H
PronLex	l eh1 s ih0 n
NETTalk	l E1 s N
WikiPron-UK	l ɛ s ə n
WikiPron-US	l ɛ s ə n

Table 2: Transcriptions of the word *lesson* as it appears in the five databases.

3.2. Quality Assurance

Since pronunciation data on Wiktionary is crowdsourced, transcription consistency cannot always be guaranteed. To address this, a custom English extractor was developed to standardize the transcription of the English "r." The extractor incorporates two modifications: (a) replacing the trill /r/ with /ɹ/, and (b) replacing word-final /əɹ/ with /ɚ/.

3.3. Preprocessing

To incorporate auxiliary information, primary grapheme–phoneme pairs were fused with secondary pronunciations using three types of concatenation: columns, control symbols, and subscripts. In the examples that follow, CELEX serves as the secondary pronunciation and NETTalk as the target dictionary. Table 3 shows the size of each dataset after concatenation.

In the **column-based** fusion approach, the dataset is structured into three columns: the first column contains the grapheme, the second column contains the feature (i.e., side pronunciation), and the third column contains the target for prediction.
Source:

```
a b a c k    @ ' b { k
```

Target:

```
x0 b @1 k
```

In the **control-based** fusion approach, the dataset is structured into two columns: the first column is the grapheme and side-pronunciation, each with a control symbol, and the second column is the target column for prediction.
Source:

```
{grapheme} a b a c k {CELEX} @ ' b { k
```

Target:

```
x0 b @1 k
```

In the **subscript-based** fusion approach, the dataset is also structured into two columns: the first column is the grapheme and side-pronunciation, with a subscript after each letter. The second column is the target column for prediction, also with a subscript after each letter. The token suffixes (e.g., /grapheme, /CELEX, /NETTalk) are informally referred to as "subscripts." Their purpose is to keep the grapheme and phoneme vocabularies mutually disjoint, ensuring that the model can distinguish between the same surface character appearing in different roles — for instance, b/grapheme, b/PronLex, and b/celex are three distinct token types despite sharing the same surface form.
Source:

```
a/grapheme b/grapheme a/grapheme
c/grapheme k/grapheme @/CELEX
'/CELEX b/CELEX {/CELEX k/CELEX
```

Target:

```
x0/NETTalk b/NETTalk @1/NETTalk
k/NETTalk
```

4. Methods

This study implemented and evaluated four architectures using Yoyodyne (Wiemerslage et al., 2025): 1) attentive LSTM, 2) pointer-generator

Dataset	Size
CELEX → NETTalk	20,459
CELEX → PronLex	98,007
CELEX → WikiPron_UK	80,220
PronLex → CELEX	79,078
PronLex → NETTalk	21,887
PronLex → WikiPron_US	83,652
WikiPron_UK → CELEX	78,507
WikiPron_UK → WikiPron_US	102,793
WikiPron_US → NETTalk	24,348
WikiPron_US → PronLex	105,872
WikiPron_US → WikiPron_UK	104,990

Table 3: Size of each dataset.

LSTM, 3) transformer, and 4) pointer-generator transformer.

Hyperparameters were first optimized via a deep sweep (200 replicates) using a fixed random seed. The fine-tuning sweeps were synced on Weights & Biases for tracking and visualization. The configuration of the sweep with the highest validation accuracy score was selected as the best set of hyperparameters for the experiments. The experiments were run with five random seeds. Each model was evaluated using word error rate (WER) as the primary metric. The median of the five WERs for each model was reported in the next section.

5. Results

5.1. Comparison Across datasets

5.1.1. G2P vs. G&P2P

Table 4 compares the WERs of different models trained on the G2P and G&P2P datasets. The right arrow "→" denotes the direction of concatenation, where the dictionary on the left serves as the side-pronunciation being merged to the dictionary on the right. The baseline pointer-generator LSTM model trained solely on CELEX yields a median WER of 26.90. When concatenated with PronLex, the WER decreases substantially to 19.81, resulting in a 7.09 absolute (26.35 relative) reduction in error. Even when combined with WikiPron_UK, a crowd-sourced dataset, the WER still declines modestly by 0.13. A similar pattern is observed for the PronLex baseline. Concatenating PronLex with CELEX achieves an 11.81 absolute (37.37 relative) reduction in error. In contrast, concatenating PronLex with WikiPron_US results in only a marginal improvement, with the WER decreasing by 0.1 points to 31.50.

A comparable trend is observed for transformer-based models. However, for pointer-generator transformer models, incorporating WikiPron_UK or WikiPron_US actually degrades performance. This pattern suggests that transformer-based models may be more sensitive to noise in the training data.

5.1.2. Expert-curated vs. Crowd-sourced Datasets

The effects of data cleanliness become more pronounced when comparing expert-curated and crowd-sourced datasets. As shown in Table 5, in the pointer-generator LSTM models, fusing two expert-curated dictionaries yields substantially better performance than combining an expert-curated dictionary with a crowd-sourced one. When NETTalk serves as the target dictionary, and CELEX is used as a side pronunciation, the WER is 20.95. A similar pattern emerges when CELEX is fused with PronLex, yielding a WER of 19.81. In contrast, performance degrades sharply when WikiPron_UK or WikiPron_UK serves as the target dataset.

5.1.3. Between Fusion Techniques

Differences in WERs across the three techniques are generally within 1 point. A Friedman test confirms that the choice of fusion strategy does not substantially affect overall performance, $\chi^2(2) = 2.40$, $p = .30$.

5.2. Comparisons Across Architectures

5.2.1. LSTM vs. Transformer

Overall, LSTM-based models consistently outperform transformer-based models on the grapheme-to-phoneme conversion task (Table 6). For instance, under the CELEX→PronLex configuration, the LSTM model achieves a WER of 19.79, compared to 20.37 for the transformer. Although the absolute difference is modest, this pattern is consistent across configurations, suggesting that LSTM architectures are better suited to G&P2P tasks under the present experimental setup. A Wilcoxon signed-rank test conducted on WERs from the pointer-generator LSTM and pointer-generator transformer confirms the results ($p < .00$).

The comparison between pointer-generator LSTMs and transformers highlights the importance of computational cost. As shown in Table 7, the LSTM model required over three and a half hours of training (3h 42m), while the transformer completed training in just over an hour (1h 1m). This nearly threefold reduction in training time underscores the transformer's advantage in parallelizability. The efficiency gains are also reflected in model size. The pointer-generator LSTM contained approximately 53.6 million trainable parameters, corresponding to an estimated parameter size of 214.51 MB. The

Dataset	PG LSTM	PG Transformer	Transformer
CELEX	26.90	36.59	34.38
PronLex → CELEX	19.81	22.78	23.12
WikiPron_UK → CELEX	26.77	38.39	31.88
PronLex	31.60	37.45	36.88
CELEX → PronLex	19.79	20.37	20.72
WikiPron_US → PronLex	30.01	42.25	35.99
WikiPron_UK	53.11	58.29	57.02
CELEX → WikiPron_UK	52.06	56.52	53.59
WikiPron_US → WikiPron_UK	45.58	43.82	43.13

Table 4: G2P vs. G&P2P Comparison.

Dataset	WER
CELEX → NETTalk	20.95
CELEX → PronLex	19.79
CELEX → WikiPron_UK	52.06
PronLex → CELEX	19.81
PronLex → NETTalk	20.36
PronLex → WikiPron_US	55.71

Table 5: WERs from pointer-generator LSTM models (median values).

pointer-generator transformer, on the other hand, contained only 3.2 million trainable parameters, with a parameter size of 12.73 MB. This dramatic reduction in model size — by more than a factor of 16 — suggests that pointer-generator transformers can achieve competitive performance with far fewer parameters, thereby reducing both memory consumption and the computational burden of optimization.

5.2.2. Attentive vs. Pointer-Generator LSTMs

Attentive and pointer-generator LSTM-based models exhibit largely comparable performance (see Table 8). A Wilcoxon signed-rank test conducted on WERs from the two model variants reveals no statistically significant difference ($p = .09$). Nevertheless, when an expert-curated dictionary is fused with a noisy crowd-sourced dictionary (e.g., CELEX→WikiPron_UK and PronLex→WikiPron_US), the attentive LSTM consistently outperforms the pointer-generator LSTM, with improvements of 14.23 points in WER for CELEX→WikiPron_UK and 11.44 points in WER for PronLex→WikiPron_US.

6. Conclusion

The results suggest that incorporating side pronunciations from external dictionaries can improve performance, particularly when the additional resource is expert-curated. However, gains appear to depend on data quality, as adding a crowd-sourced and potentially noisy dictionary may not yield consistent improvements and may, in some cases, reduce performance.

LSTM- and transformer-based models show broadly comparable performance on expert-curated datasets. LSTM-based architectures appear particularly well-suited for G&P2P tasks when a crowd-sourced dictionary is combined with an expert-curated one. Transformer-based models, though more sensitive to noise, may serve as a viable alternative when computational efficiency and parameter reduction are primary considerations.

Looking ahead, a key direction is the integration of large language models (LLMs) into TTS pipelines, where their ability to capture broader context may improve G2P and prosody prediction. Hybrid approaches that combine LLMs with pointer-generator networks, along with transfer learning, could further enhance performance while reducing reliance on curated resources. In addition, expanding pronunciation lexicons through cross-resource projection offers a promising way to increase coverage and improve robustness, particularly for out-of-vocabulary words.

7. Acknowledgements

I would like to thank Kyle Gorman and the two anonymous reviewers for their valuable feedback. Any remaining errors are my own.

8. Bibliographical References

Lucas F. E. Ashby, Travis M. Bartley, Simon Clematide, Luca Del Signore, Cameron Gibson, Kyle Gorman, Yeonju Lee-Sikka, Peter Makarov, Aidan Malanoski, Sean Miller, Omar Ortiz, Reuben Raff, Arundhati Sengupta, Bora Seo, Yulia Spektor, and Winnie Yan. 2021. Results of the second SIGMORPHON shared task on multilingual grapheme-to-phoneme conversion. In *Proceedings of the 18th SIGMORPHON Work-*

Dataset	PG LSTM	PG transformer
CELEX → NETTalk	20.95	23.09
CELEX → PronLex	19.79	20.37
CELEX → WikiPron_UK	52.06	56.52
PronLex → CELEX	19.81	22.78
PronLex → NETTalk	20.36	22.34
PronLex → WikiPron_US	55.71	59.56
WikiPron_UK → CELEX	26.77	38.39
WikiPron_UK → WikiPron_US	44.60	45.40
WikiPron_US → NETTalk	24.03	55.93
WikiPron_US → PronLex	30.01	42.25

Table 6: WERs from pointer-generator LSTM and pointer-generator transformer models.

	PG LSTM	PG Trans.
Trainable params	53.6 M	3.2 M
Total params	53.6 M	3.2 M
Total size (MB)	214.512	12.732
Training Time	3h 42m	1h 1m 53s

Table 7: Comparison between pointer-generator LSTM and pointer-generator transformer models trained on the WikiPron_UK-CELEX-column dataset.

shop on Computational Research in Phonetics, Phonology, and Morphology, pages 115–125, Online. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR 2015), Conference Track Proceedings*, San Diego, USA.

Jacob Eisenstein. 2019. *Introduction to Natural Language Processing*. MIT Press.

Kyle Gorman, Lucas F.E. Ashby, Aaron Goyzueta, Arya McCarthy, Shijie Wu, and Daniel You. 2020. The SIGMORPHON 2020 shared task on multilingual grapheme-to-phoneme conversion. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 40–50, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Piotr Kłosowski. 2022. A rule-based grapheme-to-phoneme conversion system. *Applied Sciences*, 12(5).

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Nikhil Prabhu and Katharina Kann. 2020. Making a point: Pointer-generator transformers for disjoint vocabularies. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 85–92, Suzhou, China. Association for Computational Linguistics.

Kanishka Rao, Fuchun Peng, Haşim Sak, and Françoise Beaufays. 2015. Grapheme-to-phoneme conversion using long short-term memory recurrent neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4225–4229, South Brisbane, Australia.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, Long Beach, USA. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700, Montréal, Canada. Curran Associates, Inc.

Adam Wiemerslage, Kyle Gorman, and Travis M. Bartley. 2025. Yoyodyne: Small-vocabulary neu-

Dataset	Att. LSTM	PG LSTM
CELEX → NETTalk	24.52	20.95
CELEX → PronLex	19.78	19.79
CELEX → WikiPron_UK	**37.83**	**52.06**
PronLex → CELEX	20.03	19.81
PronLex → NETTalk	19.75	20.36
PronLex → WikiPron_US	**44.27**	**55.71**
WikiPron_UK → CELEX	24.60	26.77
WikiPron_UK → WikiPron_US	48.08	44.60
WikiPron_US → NETTalk	23.53	24.03
WikiPron_US → PronLex	31.50	30.01

Table 8: WERs from attentive and pointer-generator LSTM models.

ral sequence-to-sequence models. Computer software.

Sevinj Yolchuyeva, Géza Németh, and Bálint Gyires-Tóth. 2019a. Grapheme-to-phoneme conversion with convolutional neural networks. *Applied Sciences*, 9(6).

Sevinj Yolchuyeva, Géza Németh, and Bálint Gyires-Tóth. 2019b. Transformer Based Grapheme-to-Phoneme Conversion. In *Interspeech 2019*, pages 2095–2099, Graz, Austria.

9. Language Resource References

Baayen, R. H. and Piepenbrock, R. and Gulikers, L. 1995. *CELEX2*. Linguistic Data Consortium. LDC Catalog No.: LDC96L14, ISBN: 1-58563-085-3.

Kingsbury, Paul and Strassel, Stephanie and McLemore, Cynthia and MacIntyre, Robert. 1994. *CALLHOME American English Lexicon (PRONLEX)*. Linguistic Data Consortium. LDC Catalog No.: LDC97L20, ISBN: 1-58563-110-8.

Lee, Jackson L. and Ashby, Lucas F.E. and Garza, M. Elizabeth and Lee-Sikka, Yeonju and Miller, Sean and Wong, Alan and McCarthy, Arya D. and Gorman, Kyle. 2020. *Massively Multilingual Pronunciation Modeling with WikiPron*. European Language Resources Association.

Sejnowski, Terry and Rosenberg, Charles. 1988. *Connectionist Bench (Nettalk Corpus)*. DOI: https://doi.org/10.24432/C5VP6T.

A Lightweight N-gram Approach to Abbreviation Expansion in Large Corpora

Tjaša Šoltes, Marko Bajec
Faculty of Computer and Information Science, University of Ljubljana
Večna pot 113, 1000 Ljubljana, Slovenia
tjasa.soltes@fri.uni-lj.si

Abstract

We present a lightweight, corpus-based approach to abbreviation expansion that relies solely on contextual N-gram statistics. The method models local context using two-sided and one-sided bigram and trigram counts extracted from a large domain-specific corpus. Candidate expansions are selected through linear interpolation of context-specific evidence, enhanced with reliability-based scaling to mitigate sparse data effects. The approach does not require external linguistic resources, pretrained language models, or explicit morphosyntactic analysis, making it suitable for domain-specific and resource-constrained settings. Experiments conducted on a large Slovene medical corpus demonstrate that interpolation generally outperforms strict backoff strategies, with notable improvements for medium- and low-frequency abbreviations. Despite its simplicity, the proposed framework achieves robust performance while remaining computationally efficient and scalable.

Keywords: text normalization, abbreviation expansion, N-gram model

1. Introduction

Abbreviation expansion is an essential preprocessing step in many natural language processing pipelines, particularly in applications such as text-to-speech (TTS) and automatic speech recognition (ASR), where non-standard word forms must be converted into their fully expanded equivalents. Existing approaches to abbreviation expansion often rely on predefined lexicons, rule-based systems, or semantic disambiguation methods that attempt to determine the underlying meaning of a given abbreviation.

In this paper, we propose a data-driven N-gram-based approach to abbreviation expansion in large corpora. We focused on abbreviations, which we defined as alphabetical strings ending with a dot. The method relies on contextual bigram and trigram information to select the most probable expansion candidate. Context representations are transformed into a compact binary form to enable efficient lookup and scoring, making the approach computationally lightweight. The proposed approach is fully unsupervised: all candidate expansions and contextual statistics are derived directly from raw corpus data.

Unlike many existing methods, the proposed approach does not require external linguistic resources, manually curated abbreviation dictionaries, or explicit semantic interpretation of abbreviations. It can be adapted to specific domains or languages by training directly on in-domain corpora, making it particularly suitable for low-resource or specialized settings. Furthermore, the method is computationally efficient and can be applied to large datasets without extensive preprocessing. Lightweight approaches remain valuable in clinical and domain-specific settings where computational resources, latency constraints, or data privacy concerns limit the applicability of large neural models.

For morphologically rich languages such as Slovene, abbreviation expansion presents an additional challenge: the correct expansion often depends on grammatical case and other morphosyntactic features. Traditional solutions typically require morphosyntactic analysis to determine the appropriate inflected form. [1] In contrast, our context-based N-gram approach implicitly captures such information from surface-level contextual patterns, eliminating the need for explicit morphological analysis while still producing grammatically appropriate expansions.

The contributions of this work are threefold: (1) we introduce a lightweight N-gram-based framework for context-sensitive abbreviation expansion that does not rely on external linguistic resources; (2) we propose a reliability-aware interpolation strategy that improves robustness in sparse-data conditions; and (3) we provide an evaluation on a large Slovene medical corpus with analysis across abbreviation frequency bands.

2. Related Work

Early work on abbreviation expansion was largely driven by distributional similarity and manually designed representations. Terada et al. (2004) pro-

[1] Slovene uses 6 grammatical cases and three grammatical numbers: alongside singular and plural, dual is also used. This means that a single word can have up to 18 forms.

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 95–100
12 May, 2026.

pose a method that ranks candidate expansions using cosine similarity between context vectors derived from abbreviation-rich and abbreviation-poor corpora, supplemented with character-based filtering rules that approximate human abbreviation patterns. In a similar vein, Pakhomov et al. (2005) introduces a semi-supervised approach to acronym disambiguation that automatically harvests sense-specific contexts from the Web, MEDLINE, and clinical notes. These contexts are encoded as vector-space representations, and meanings are assigned via cosine-based context similarity. Related distributional ideas also appear in text normalization work, where Cook and Stevenson (2009) propose an unsupervised method that learns mappings from non-standard SMS forms to standard words using contextual similarity in unannotated corpora, demonstrating that normalization can be addressed through corpus-based inference without labeled data or handcrafted rules.

Subsequent research increasingly framed abbreviation expansion as a structured inference problem, often within noisy-channel architectures. Roark and Sproat (2014) introduce a conservative, largely unsupervised methodology that mines billions of words of unannotated text to extract candidate abbreviation–expansion pairs and trains a joint 7-gram character model to score their plausibility. This pair model is combined with two context-sensitive components—a pruned trigram language model and an SVM classifier using contextual likelihood and token-level features—with expansions applied only when both models independently agree. Gorman et al. (2021) further formalize this paradigm by introducing a large, openly available dataset of English ad hoc abbreviations and evaluating two supervised noisy-channel models: a finite-state pipeline combining a trigram language model with a character-level pair model, and a neural pipeline pairing an LSTM language model with a subsequence-based abbreviation model. Their results show that both approaches achieve near-human accuracy, with the neural variant performing best.

More recent work has shifted toward neural architectures that integrate morphological and contextual modeling more tightly. Chopard and Spasić (2019) propose a modular neural framework that combines pattern-based abbreviation detection, a character-level RNN to distinguish abbreviations from acronyms, and a Siamese network to model morphological formation and generate candidate expansions, followed by context-based ranking without reliance on external lexical resources. At a larger scale, contemporary approaches increasingly leverage deep neural networks and transformer-based language models, enabling end-to-end contextual modeling. Privacy-preserving training strategies, such as reverse substitution on public corpora, show that high-quality clinical expansion models can be developed without direct access to protected health records (Rajkomar et al., 2022). Large language models have also been evaluated in zero-shot and fine-tuned settings for clinical acronym expansion, demonstrating strong adaptability alongside persistent domain-specific limitations (Kugic et al., 2024; Nezhad et al., 2025). In parallel, lightweight and task-oriented systems emerging from shared tasks emphasize efficiency, domain adaptation, and standardized evaluation protocols (Kugic et al., 2024).

3. Corpus and data

The experiment was conducted on a proprietary corpus of real Slovenian medical documents collected from various fields of medicine and various institutions. The corpus includes 3,893,434 sentences and 59,083,266 tokens (137,423 types).

We identified 237 abbreviations types in the corpus (with the total frequency of 87,610). We defined an abbreviation as any alphabetic string that ends with a dot. We divided the abbreviations into five frequency categories, that is: very frequent (more than 5,000 occurrences), frequent (1,000–5,000), medium (100–1,000), rare (10–100), and very rare (fewer than 10). The median value of expansions per abbreviation is 4.

4. Methodology

4.1. Data

To ensure scalability and efficient lookup, all contextual N-gram statistics were stored in compact binary files (*.bin)*. As per standard practice, each unique token was mapped to an integer identifier, allowing N-gram contexts to be represented as fixed-size integer tuples rather than string sequences. This representation significantly reduces memory usage and accelerates access during candidate retrieval and scoring. Separate binary count tables were constructed for each context type (center trigrams, left and right trigrams, and left and right bigrams), which can be observed in 1, enabling fast aggregation of contextual evidence at inference time. The use of binary storage allows the method to operate efficiently even on large corpora without requiring database systems or external indexing frameworks.

4.2. Detecting abbreviations and their possible expansions

We defined an abbreviation as any token that appears in the sentence and ends with a dot which is preceded by letters only. Due to some mistakes in

N-gram name	N-gram content
center trigram (C3)	(t-1, _, t+1)
left trigram (L3)	(t-2, t-1, _)
right trigram (R3)	(_, t+1, t+2)
left bigram (L2)	(t-1, _)
right bigram (R2)	(_, t+1)

Table 1: Representation of the five N-gram context types.

sentence tokenization, we identified 18 cases (out of 237) where a sentence was not split which led to the last word in the sentence being incorrectly identified as an abbreviation.

We compiled a list of abbreviations to be excluded from the expansion process. This decision was motivated by the observation that most conventionalized (highly frequent and generally non-field-specific) abbreviations rarely occur in their expanded form. This category primarily comprises titles (e.g., *dr.* 'doktor' (doctor), *ga.* 'gospa' (Mrs.)), as well as common non-domain abbreviations (e.g., *npr.* 'na primer' (e.g.)). Additionally, it includes selected domain-specific abbreviations that are conventionally not written out in full, such as, for example, various forms of *b. p.* 'brez posebnosti' (without specifics). The final exclusion list comprised a total of 34 abbreviations.

Candidate expansions are generated directly from contextual N-gram statistics extracted from the corpus. For each occurrence of an abbreviation, its surrounding context within a five-token window is matched against precomputed bigram and trigram tables. These tables store all tokens observed in identical context. The union of all tokens retrieved from the matching context tables forms the candidate set.

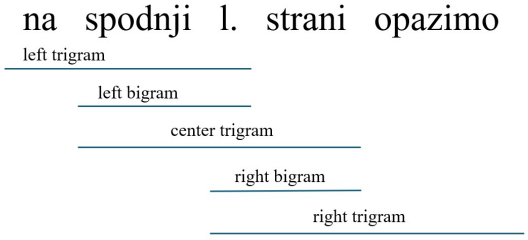


Figure 1: Representation of N-grams in an example sentence *na spodnji l. strani opazimo* 'at bottom l. side we_notice'.

As can be observed in 1s, in the example sentence: *na spodnji l. strani opazimo* (literally: 'at bottom l. side we_notice'), the target abbreviation to expand is 'l.' (to be expanded into *levi* 'left' in singular feminine locative form). The candidates are retireved from the following contexts: left trigram (*na, spodnji, _*), left bigram (*spodnji, _*), center trigram (*spodnji, _, strani*), right bigram (*_, strani*) and right trigram (*_, strani, opazimo*).

The contexts store all tokens appearing in these N-grams in the position of the target abbreviation *l.* and the frequency of the N-gram. Out of these, the candidates are selected. A candidate is always a single token. It must satisfy several constraints: they must share the prefix of the abbreviation (excluding the trailing dot), consist only of alphabetic characters, and not themselves be abbreviations. This procedure ensures that only contextually plausible and orthographically compatible expansions are considered during scoring.

4.3. Scoring Function

Once a list of candidates is generated, the candidates are scored. We compared two scoring approaches: backoff and linear interpolation.

We used a simple backoff algorithm where we used the result of a single context. We defined the order of contexts, and started from most determinative. For example: if the center trigram produces any candidates, the abbreviation is expanded into the candidate with the highest frequency (in the context of this trigram). If no candidates are produced in the center trigram, the same process is repeated with the right trigram, then left trigram, and so on.

The second approach we used for scoring is linear interpolation. First, the candidates were produced in all contexts. For each candidate w, the final score is computed using linear interpolation with predefined fixed weights across contextual models:

$$S(w) = \sum_i \lambda_i \cdot R_i \cdot \frac{c_i(w)}{\sum_{w'} c_i(w')}$$

where λ_i denotes the interpolation weight assigned to context type i, $c_i(w)$ represents the count of candidate w in that context, and R_i is a reliability factor defined as:

$$R_i = \frac{N_i}{N_i + k}$$

with N_i being the total number of observations for context i and k a smoothing constant. This reliability scaling reduces the impact of sparse contextual evidence.

It is important to note that the candidate w represents a single token (i.e., a possible expansion of the abbreviation) rather than a sequence of tokens. The proposed method ranks individual candidate expansions conditioned on observed contextual patterns.

The interpolation weights and smoothing constant were selected using grid search on a held-out validation subset.

5. Results

The test set was manually constructed to ensure balanced evaluation across abbreviation frequency bands. Abbreviations were grouped into five categories based on their corpus frequency (very frequent, frequent, medium, rare, and very rare). Sentences were randomly sampled for each abbreviation within each category. Only the selected abbreviation and its expansion were evaluated in each sentence, even if additional abbreviations appeared in the sentence. Sentences with insufficient contextual information, ambiguous expansions, or predominantly non-alphabetic content (e.g., laboratory reports) were excluded from the test set. The final test set contains 1030 instances and provides a stratified evaluation across varying levels of data sparsity. We included 151 cases of very frequent abbreviations, frequent abbreviations contribute to 252 cases, medium abbreviations appear 432 times, rare abbreviations contribute to 90 cases and very rare abbreviations are included in all 105 cases.

The abbreviations in the test set exhibited an average of 3.3 expansions, with a median of 3, a maximum of 11, and a minimum of 1.

Table 2 presents the performance of different scoring strategies across abbreviation frequency bands. The weakest configuration is the fixed-weight interpolation[2] model that includes all bigrams during candidate production, achieving an overall accuracy of 77.4%. This setting performs consistently worse across all frequency bands, with particularly low accuracy for very frequent (69.5%) and frequent abbreviations (77.0%). These results indicate that incorporating left bigram context during candidate retrieval substantially increases noise and negatively affects ranking performance.

Excluding left bigrams from candidate production while retaining fixed-weight interpolation leads to a marked improvement, raising overall accuracy to 87.1%. Gains are observed across all frequency bands, including very frequent (88.1%), frequent (90.1%), medium (89.6%), rare (78.9%), and very rare abbreviations (75.2%). This demonstrates that separating candidate generation from scoring is crucial for maintaining precision.

Introducing reliability-weighted interpolation further improves performance. With $k=5$ and left bigrams excluded from candidate production, but used in scoring, the model achieves the best overall accuracy of 89.2%. The most pronounced improvements are observed for very frequent abbreviations (95.4%–96.0%) and medium-frequency abbreviations (up to 91.9%). Rare abbreviations also benefit, improving from 78.9% under fixed interpolation to 81.1%. Performance for very rare abbreviations remains stable at approximately 75%, suggesting that extremely sparse contextual evidence continues to limit predictive accuracy regardless of scoring strategy.

Backoff strategies perform competitively, particularly for rare and very rare abbreviations, where the best backoff configuration achieves 83.3% and 76.2%, respectively. However, backoff does not surpass reliability-weighted interpolation in overall accuracy, as it underutilizes partial contextual evidence when multiple informative signals are available.

Overall, the results confirm that excluding left bigrams from candidate production is essential for stable performance, and that reliability-aware interpolation provides the most robust scoring strategy across frequency bands.

5.1. Discussion

The experimental results provide several important insights into the behavior of context-based N-gram models for abbreviation expansion. The lowest-performing configuration—fixed-weight interpolation with all bigrams included in candidate production—achieves only 77.4% overall accuracy, with particularly poor performance for very frequent abbreviations (69.5%). This indicates that incorporating left bigram context during candidate retrieval substantially increases noise. These results show that overly permissive candidate generation can harm performance.

Excluding left bigrams from candidate production raises overall accuracy to 87.1%, with consistent improvements across all frequency bands. This demonstrates the importance of separating candidate retrieval from scoring: restricting candidate generation to more informative contexts reduces noise while preserving useful contextual evidence for ranking.

Reliability-aware interpolation further improves performance, reaching up to 89.2% overall accuracy. The gains are most pronounced for very frequent and medium-frequency abbreviations, where contextual evidence is sufficient but unevenly distributed. Reliability scaling downweights sparse contexts and leads to consistent improvements for rare abbreviations as well (from 78.9% to 81.1%). Performance for very rare abbreviations remains stable at approximately 75%, suggesting that interpolation alone cannot overcome extreme sparsity.

In such sparse conditions, strict backoff performs slightly better, achieving 83.3% for rare and 76.2%

[2]The weight in all interpolation approaches used were: center trigram: 0.45, left trigram: 0.20, right trigram: 0.20, lef bigram: 0.075, right bigram: 0.075

Model	All	V. Freq.	Freq.	Medium	Rare	V. Rare
Interpolation (no reliability)						
all context for candidate production	77.4	69.5	77.0	82.2	74.4	72.4
no L2 for candidate prod.	87.1	88.1	90.1	89.6	78.9	75.2
Interpolation (reliability-weighted)						
$k = 5$	89.1	96.0	89.3	91.7	81.1	75.2
$k = 5$, no L2 for candidate prod.	89.2	95.4	89.7	91.9	81.1	75.2
Backoff						
C3 $\rightarrow$ R3 $\rightarrow$ L3 $\rightarrow$ R2 $\rightarrow$ L2	88.1	94.0	88.9	89.4	83.3	76.2
C3 $\rightarrow$ L3 $\rightarrow$ R3 $\rightarrow$ L2 $\rightarrow$ R2	87.3	94.0	88.1	88.9	80.0	75.2

Table 2: Accuracy across interpolation and backoff configurations. Columns "V. Freq.", "Freq.", "Medium", "Rare", and "V. Rare" denote performance over groups of abbreviations binned by frequency. Left context refers to its inclusion in candidate production. Reliability-weighted models use scaling parameter k. In backoff configurations, C3, L3, R3, L2, and R2 denote center, left, and right context n-gram models (trigram or bigram, respectively).

for very rare abbreviations. This reflects the difference between the two strategies: interpolation aggregates multiple signals, whereas backoff relies on a single best-available context, which may be advantageous when most signals are weak.

Overall, the findings highlight three main points: candidate generation plays a decisive role in performance, reliability-aware interpolation provides the most robust overall strategy, and backoff remains competitive under extreme sparsity. Despite its simplicity, the framework demonstrates that surface-level distributional statistics capture sufficient contextual information to resolve many morphosyntactic distinctions implicitly. However, performance remains limited for very rare abbreviations and in cases where candidate expansions are orthographically similar, making contextual discrimination inherently difficult.

5.2. Computational Efficiency

The complete set of binary N-gram tables occupies 169.8 MB of disk space. All experiments were conducted on a standard laptop equipped with an AMD Ryzen 7 5700U CPU (1.80 GHz) and 16 GB RAM. No GPU acceleration was used.

During inference, the system processes approximately 2,375 abbreviation instances per second. These results confirm the computational efficiency of the proposed approach. The model remains compact in size and achieves high throughput, making it suitable for large-scale corpus processing and real-time preprocessing pipelines.

6. Conclusion

This work presented a lightweight, corpus-based N-gram approach to abbreviation expansion that relies exclusively on contextual statistics. The method requires no external linguistic resources, pretrained models, or explicit morphosyntactic analysis, making it suitable for domain-specific and resource-constrained environments.

The experiments demonstrate that reliability-aware interpolation provides the most robust overall solution, outperforming fixed-weight interpolation and generally surpassing strict backoff strategies. The results further show that excluding left context from candidate production significantly improves accuracy, highlighting the importance of carefully separating candidate retrieval from scoring. While backoff remains competitive in extremely sparse conditions, interpolation offers a more flexible and stable framework across frequency bands.

Overall, the findings confirm that distributional N-gram modeling remains a strong and practical approach in resource-constrained settings, particularly in specialized domains and morphologically rich languages. Future work may explore hybrid approaches that integrate lightweight statistical modeling with selective linguistic constraints or domain-adaptive neural components to further improve performance on rare and ambiguous cases.

7. Acknowledgements

This paper was conducted as part of Basic Research for the Development of Spoken Language Resources and Speech Technologies for the Slovenian Language (MEZZANINE J74642), financed from the national budget by a contract between the Slovenian Research Agency and the Faculty of Computer and Information Science, University of Ljubljana.

8. Bibliographical References

Daphné Chopard and Irena Spasić. 2019. A deep learning approach to self-expansion of abbreviations based on morphology and context distance. In *Statistical Language and Speech Processing: 7th International Conference, SLSP 2019, Ljubljana, Slovenia, October 14–16, 2019, Proceedings*, page 71–82, Berlin, Heidelberg. Springer-Verlag.

Paul Cook and Suzanne Stevenson. 2009. An unsupervised model for text message normalization. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 71–78, Boulder, Colorado. Association for Computational Linguistics.

Kyle Gorman, Christo Kirov, Brian Roark, and Richard Sproat. 2021. Structured abbreviation expansion in context. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 995–1005, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Amila Kugic, Stefan Schulz, and Markus Kreuzthaler. 2024. Disambiguation of acronyms in clinical narratives with large language models. *Journal of the American Medical Informatics Association*, 31(9):2040–2046.

Nima Shafiei Rezvani Nezhad, Meysam Mansouri, Rabih Abdulkarim Zakaria, and Ruhollah Abolhasani. 2025. Medical abbreviation disambiguation with large language models: Zero- and few-shot evaluation on the medal dataset. *bioRxiv*.

Serguei V. S. Pakhomov, Ted Pedersen, and Christopher G. Chute. 2005. Abbreviation and acronym disambiguation in clinical discourse. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, pages 589–93.

Alvin Rajkomar, Eric Loreaux, Yuchen Liu, Jonas Kemp, Benny Li, Ming-Jun Chen, Yi Zhang, Afroz Mohiuddin, Juraj Gottweis, et al. 2022. Deciphering clinical abbreviations with a privacy protecting machine learning system. *Nature Communications*, 13(1):7456.

Brian Roark and Richard Sproat. 2014. Hippocratic abbreviation expansion. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369, Baltimore, Maryland. Association for Computational Linguistics.

Akira Terada, Takenobu Tokunaga, and Hozumi Tanaka. 2004. Automatic expansion of abbreviations by using context and character information. *Information Processing Management*, 40(1):31–45.

Inverse Text Normalization for Arabic Numbers in Streaming ASR

Enas Albasiri, Myungjong Kim, Nourchene Ferchichi, Oluwatobi Olabiyi
NVIDIA
Santa Clara, CA, USA
{ealbasiri, myungjongk, nferchichi, oolabiyi}@nvidia.com

Abstract

Streaming multilingual speech recognition benefits from unified systems that produce numbers in their written form '34' rather than their spoken form 'thirty-four'. By generating digits directly, these systems eliminate the post-processing latency inherent in cascaded architectures that require a separate inverse text normalization (ITN) step. Arabic presents a formidable challenge for ITN; the system must not only determine the correct numerical value but also navigate complex rules for gender, number, and case marking that are determined by the counted noun. For instance, the digit '7' (as in '47') exhibits gender polarity: it must take a masculine form if modifying a feminine noun (e.g., Halala) and a feminine form if modifying a masculine noun (e.g., Riyal). While Arabic dialects typically exhibit simplified numeral systems by omitting case and gender markers, they vary significantly in verbalization patterns. This study explores the efficacy of a unified streaming Automatic Speech Recognition (ASR) system with integrated ITN features, comparing it against a traditional cascaded approach utilizing a post-processing rule-based ITN module. We utilize a FastConformer cache-aware streaming model trained on English and a diverse Arabic corpus spanning Modern Standard (MSA), dialectal, and Classical Arabic, while maintaining diacritics where contextually appropriate. We evaluate the system using Word Error Rate (WER) for ASR accuracy and exact match for ITN capability. Our results demonstrate that integrating ITN does not degrade core ASR performance and that the unified model achieves accuracy competitive with cascaded systems across Arabic variants. However, error analysis reveals that the primary failures in ITN are rooted in diacritization, gender polarity, and orthographic variation, highlighting the challenges of Arabic's unique linguistic features in end-to-end modeling.

Keywords: Arabic Text Normalization, Inverse Text Normalization, Diacritization, Dialectal Arabic, Gender Polarity, Streaming Multilingual ASR

1. Introduction

High-quality multilingual automatic speech recognition (ASR) requires language-specific data and techniques. Streaming ASR benefits from unified systems that produce numbers in their written form (34) instead of their spoken form (thirty four), as they eliminate the post-processing latency inherent in cascaded systems where a separate inverse text normalization (ITN) step is applied to convert numbers from their spoken form to digits. English ASR has attained the quality of human transcribers not only in recognition tasks (Amodei et al., 2015; Xiong et al., 2017; Saon et al., 2017) but also in unified ASR with ITN (Nguyen et al., 2023; Tang et al., 2025). However, the pursuit of precise and efficient speech recognition models for languages with rich morphology and a substantial number of spoken variants, such as Arabic, presents significant challenges (e.g., Al-Sughaiyer and Al-Kharashi, 2004; Soudi et al., 2007; Habash, 2010; Alkuhlani and Habash, 2011; El-kholy and Habash, 2011; Inoue et al., 2022).

Unlike Modern Standard Arabic (MSA), the spoken dialects lack a standardized orthography. Much of the written data in dialectal Arabic is derived from informal communication channels, such as social messaging and social media, with inconsistent orthography. For example, **3** ‹θ.lā.θa› can have multiple spellings across MSA and dialects (1).

(1) Variations of ثلاثة **‹θ.lā.θa›** 'three'

a. ثلاثة [θala:θa]
b. ثلاثه [θala:θa]
c. تلاتة [tala:ta]
d. تلاته [tala:ta]
e. تلاتا [tala:ta]

This orthographic inconsistency poses a challenge for ASR models, as a single pronunciation may correspond to multiple valid spellings, making it difficult to learn a consistent mapping from speech to text for dialectal Arabic.

Furthermore, Arabic writing omits key information about pronunciation, which increases ambiguity. Such writing systems are sometimes referred to as defective. In Arabic, consonants and long vowels are written, while short vowels are omitted, except in educational and religious materials. Diacritics can be added to disambiguate meaning. For example, the consonantal skeleton خمس ‹x.m.s› can yield several distinct readings depending on diacritization and grammatical context (2).
An ITN system must map (2a–d) to the digit *5* and

Proceedings of the Third Workshop on Computation and Written Language (CAWL 2026) @LREC 2026, pages 101–106
12 May, 2026.

(2) خمس <x.m.s>

a.	خَمْسُ	[xamsu]	‘five’ (nom.)
b.	خَمْسَةُ	[xamsatu]	‘five’ (nom. fem.)
c.	خَمْسِ	[xamsi]	‘five’ (gen.)
d.	خَمْسَةِ	[xamsati]	‘five’ (gen. fem.)
e.	خُمْس	[xums]	‘one-fifth’ (1/5)

(2e) to the fraction *1/5*, yet without diacritics the input is ambiguous.

Defective scripts are problematic for ASR, making mapping from pronunciation to spelling more difficult (Habash, 2010; Habash et al., 2013).

For Arabic, ITN systems must not only determine the correct numerical value, but the numeral must also show the appropriate gender, number, and case marking with the counted noun. For example, 47 is structured by coordinating the digit part first ‘seven’ followed by the coordinate particle *wa* ‘and’ and the tens part ‘forty’. The digit for ‘seven’ can be either masculine or feminine depending on the noun it modifies. If it modifies a feminine noun, such as هللة ‘Halala,’ it must show the opposite gender, that is, masculine. However, if it modifies a masculine noun, ريال ‘Riyal,’ it must be feminine. This rule of agreement is known as polarity, whereby a masculine-counted noun agrees with a feminine numeral and vice versa (Alqassas, 2017).

While MSA enforces gender polarity, Arabic dialects diverge in their number verbalization. Najdi Arabic preserves gender polarity (Gadalla, 2023). Hijazi Arabic, by contrast, does not (Bardeas, 2009; Gadalla, 2000) (3b):

(3) a. بلايز أربع
balajez *ʔarbaʕ*
shirt-F.PL four.M
[polarity]
‘Four shirts’ [Najdi]

b. بلايز أربعة
balajez *ʔarbaʕ-a*
shirt.F.PL four-F
[agreement]
‘Four shirts’ [Hijazi]

This variation means that an ITN system cannot assume a single set of normalization rules.

This paper examines the interaction between these challenges: gender polarity, dialectal variation in number verbalization, and diacritization within a unified Arabic ASR and ITN pipeline. We compare unified (E2E) and cascade ITN approaches using a FastConformer cache-aware streaming model (Rekesh et al., 2023; Noroozi et al., 2024) trained on English and across Standard, dialectal, and classical Arabic. Through ASR evaluation and ITN error analysis, we show that (i) integrating ITN does not degrade ASR accuracy, (ii) the unified model achieves competitive ITN accuracy compared to the cascade system, and (iii) the dominant ITN errors stem from diacritization, gender polarity, and orthographic variation rooted in Arabic morphosyntax.

2. Related Work

Multilingual ASR has predominantly relied on rule-based ITN applied as a post-processing step after ASR decoding. However, for streaming applications, cascade ITN introduces additional latency. Gaur et al. (2023) proposed a hybrid on-device system combining a streaming transformer tagger with category-specific WFSTs, reducing model size while maintaining accuracy across 16 semiotic classes. Microsoft’s FOUR-IN-ONE system (Nguyen et al., 2023) consolidates ITN, punctuation, capitalization, and disfluency removal into a single transformer tagger, matching or outperforming task-specific models. More recently, AssemblyAI’s Universal-2-TF (Tang et al., 2025) introduced a two-stage neural model that jointly handles ITN, punctuation, and capitalization in a single architecture.

The alternative—end-to-end (E2E) ITN—trains the ASR model directly on written-form transcripts so that numbers are output as digits without post-processing. Salazar et al. (2024) compared cascade and E2E approaches for numeric expression formatting, using LLM-generated data synthesized with TTS to adapt an E2E model. Their results showed that E2E models achieve competitive accuracy with lower latency and inference cost compared to cascade systems.

Multilingual ASR with ITN remains limited, and no prior work has compared E2E and cascade ITN for Arabic. Our work addresses this gap.

3. System Architecture

3.1. ASR Model

The ASR component employs a cache-aware streaming FastConformer architecture with a hybrid CTC/RNNT decoder (Noroozi et al., 2024). This architecture adapts the FastConformer (Rekesh et al., 2023) for streaming applications by constraining both look-ahead and past contexts in the encoder and introducing an activation caching mechanism that converts the non-autoregressive encoder into an autoregressive recurrent model during inference.

A 600M-parameter multilingual model pretrained on 40 languages serves as the base model. The model employs subword encoding via Byte Pair Encoding (BPE) with a universal tokenizer supporting 40 languages (Sennrich et al., 2016). It handles Arabic script with diacritical marks as part of the output vocabulary, where diacritics are treated as separate tokens following their associated base characters, consistent with Unicode encoding order.

3.2. Arabic ITN with WFSTs

The ITN component is a rule-based normalizer developed using weighted finite-state transducers (WFSTs) with Pynini, a Python library for WFST grammar development (Gorman, 2016). The library is integrated into NVIDIA's NeMo text processing toolkit (Zhang et al., 2021).

The current WFST-based ITN system is deterministic and context-independent, meaning it does not have access to any contextual information outside the numerical token. The grammar accounts for gender agreement within numeral expressions and defaults to the nominative case and feminine gender. Additionally, it targets MSA and does not account for dialectal variation in numeral verbalization.

4. Evaluation

To evaluate unified against a cascade approach, we fine-tuned two models from the same 600M-parameter multilingual base model. The **baseline model** was fine-tuned in the transcripts in their original spoken form. The **unified model** was fine-tuned on the same data after applying the NeMo text processing ITN module for both Arabic and English. Both models use the RNNT decoder at inference with a chunk size of (0.56 s).

The training data comprises approximately 7600 hours of Arabic speech, with MSA constituting roughly 30%, dialectal Arabic approximately 20%, and classical Arabic 10%, and the remaining 40% are unclassified variants and approximately 80000 hours of English. Both models were fine-tuned on 24 A100 GPUs with noise augmentation applied to 20% of the training data at signal-to-noise ratios of 7.5 and 12.5 dB.

For ASR evaluation, both models were tested on four openly available Arabic datasets: Casablanca, Common Voice, MGB-2, and SADA from the Open Universal Arabic ASR Leaderboard (Wang et al., 2024) and three internal custom English evaluation datasets—LibriSpeech, Tedlium2, and SPGISpeech. For Arabic, WER and CER were computed using the Open Universal Arabic ASR Leaderboard evaluation protocol, which normalizes text by removing punctuation and diacritics, collapsing hamza and madda variants, and converting Eastern Arabic numerals to Western Arabic numerals before calculating edit distance. Since the unified model outputs digits while the leaderboard references are in spoken form, we apply NeMo text normalization to the unified model's predictions to convert digits back to spoken form before WER calculation, ensuring a fair comparison. For English, WER was calculated using the standard formulation after punctuation removal. For ITN evaluation, we compare two configurations: (1) **unified**, where the ITN model's output is evaluated directly, and (2) **cascade**, where the baseline model's spoken-form output is post-processed by the NeMo ITN module. For Arabic, 2,768 samples containing numerical expressions were extracted from internal ASR evaluation test sets by selecting utterances with Arabic number words, running ITN on the ground-truth transcripts, and diffing to extract (spoken form, written form). Because the written-form references are generated by the deterministic WFST grammar, which defaults to feminine gender and nominative case in MSA, the gold standard represents one conventionalized surface form rather than the uniquely correct realization. For English, the evaluation set was created from the Google text normalization corpus (Sproat and Jaitly, 2017), which was synthesized into audio using NVIDIA's MagpieTTS Multilingual model (Neekhara et al., 2024). The ITN metric used is the *exact match*, which measures whether the model's output contains the expected form for each numerical expression.

4.1. ASR Results

Table 1: ASR results: baseline vs. unified. WER / CER (%).

	Baseline		Unified	
Test Set	**WER**	**CER**	**WER**	**CER**
Arabic				
Casablanca	73.73	38.60	73.53	39.07
Common Voice	27.58	8.19	26.66	7.97
MGB-2	26.95	11.63	26.86	11.59
SADA	52.17	31.24	51.44	30.25
Average	**45.11**	**22.42**	**44.62**	**22.22**
English				
LibriSpeech	7.09	3.00	7.27	3.09
Tedlium2	5.39	3.00	5.75	3.32
SPGI	7.54	3.85	7.97	4.01
Average	**6.67**	**3.28**	**7.00**	**3.47**

Table 1 compares ASR performance between the baseline and unified models. For Arabic, the unified model achieves an average WER of 44.62% versus 45.11% for the baseline, with CER remaining nearly identical (22.22% vs. 22.42%). The unified model matches or slightly improves over the baseline on all four test sets. For English, the average WER increases from 6.67% to 7.00%, and CER rising from 3.28% to 3.47%. These results indicate that training on ITN transcripts does not significantly degrade ASR accuracy on general speech domains for either language, confirming that the unified approach can integrate ITN capability without sacrificing speech recognition quality.

4.2. ITN Results

To evaluate ITN accuracy, the accuracy of ASR ITN output to the expected written form for each numerical expression is measured using the exact match metric. We compare three configurations: (1) **baseline**, which outputs numbers in spoken form and is evaluated against the spoken-form reference; (2) **cascade**, where the baseline output is post-processed by the NeMo ITN module; and (3) **unified**, where the model is trained directly on ITN transcripts. Results are reported in Table 2.

Table 2: ITN evaluation: baseline vs. cascade vs. unified.

System	Corr.	Total	Acc.%
Arabic			
Baseline	2,144	2,768	77.46
Cascade	2,057	2,768	74.31
Unified	2,066	2,768	74.64
English			
Baseline	3,947	4,247	92.94
Cascade	3,452	4,247	81.28
Unified	3,518	4,247	82.83

For Arabic, the baseline model achieves 77.46% exact match against reference, while the cascade system, which applies the WFST-based ITN module as post-processing, achieves 74.31%. The unified model slightly outperforms cascade (74.64% vs. 74.31%)

For English, the baseline achieves 92.94% accuracy. The cascade system achieves 81.28% and the unified model 82.83%, a 1.5% advantage for the unified approach.

An error analysis of the 702 Arabic errors from the unified system reveals that in 66.7% of cases the model outputs the number in its spoken form rather than converting it to digits. Three major error types emerge:

Diacritics (27.9% of errors): the model produces the normalized number with diacritization. For example, for target *5*, the model maps it to the spoken form خمسة *xamsa*; however, when the surrounding context is fully diacritized, the model produces a diacritized variant خَمْسَةِ *xamsati* as in حَدَثَ ذَلِكَ مُنْذُ خَمْسَةِ أَيَّامٍ ‘That happened five days ago,’ and does not extend ITN to the diacritized form.

Gender mismatch (22.7% of errors): the model outputs the spoken form with the wrong gender marking. For example, for target *3*, the expected spoken form is ثلاث *θala:θ* (masculine), but the model produces ثلاثة *θala:θa* (feminine).

Orthographic variation (10.8% of errors): the model outputs the spoken form in a different orthographic form instead of the ITN digit form. For example, for target *1000*, the expected spoken form is ألف *ʔalf* with hamza, but the model produces الف *alf* without hamza, as in لو بطل شجع كمل بعد الف صوت ‘If a champion encouraged, continue after a thousand voices.’ This indicates that spelling and orthographic variation in the training data prevents the model from consistently recognizing numeral tokens for ITN conversion.

The remaining errors include partial ITN (11.1%), where only part of a compound numeral is converted (e.g., أرباع ‘quarters’ kept as text in *3/4*), which could be attributed to the streaming decoding effect; and number missing (9.1%), where the numerical content is absent from the prediction entirely.

5. Leveraging LLMs for Arabic TN/ITN

Large language models (LLMs) leverage vast amounts of text data to learn intricate linguistic patterns and generate contextually relevant outputs. Unlike rule-based models, LLMs possess the flexibility to adapt to diverse linguistic contexts and capture nuanced language features, making them promising candidates for TN/ITN tasks in Arabic and other languages. Zhang et al. (2023) reported a major success for LLMs in handling text normalization in English, achieving a 40% lower word error rate compared to rule-based systems.

LLM-based TN/ITN suffers the production of *unrecoverable errors*: outputs that are linguistically plausible but factually incorrect (Gorman and Sproat, 2016).

A hybrid strategy combining WFSTs for deterministic, high-confidence transductions with LLMs for context-dependent cases could serve as a preprocessing pipeline for generating higher-quality ITN training data, enabling the unified model to learn from more accurate and context aware

transcripts covering dialectal variants, diacritized forms, and gender-inflected numerals.

6. Conclusion

We compared unified (E2E) and cascade ITN approaches within a multilingual streaming ASR system for Arabic and English. Our results show that training on written-form transcripts does not degrade ASR performance: the unified model achieves comparable WER and CER to the baseline across both languages. For ITN accuracy, the unified model slightly outperforms the cascade for both Arabic (74.64% vs. 74.31%) and English (82.83% vs. 81.28%), while eliminating the latency of post-processing. Error analysis reveals that the dominant Arabic ITN failures stem from diacritization mismatches (27.9%), gender polarity errors (22.7%), and orthographic variation (10.8%). These findings point toward dialect-sensitive, context-aware normalization strategies as a direction for future work, such as enriching training data with dialectal variants or leveraging LLMs for context-dependent normalization.

7. Limitations

The Arabic ITN evaluation corpus was constructed by filtering utterances containing number words from existing test sets, which may not fully capture the range of naturally occurring Arabic numerical expressions. The English evaluation set was synthetically generated using TTS, introducing acoustic artifacts that may affect ASR accuracy. Additionally, the WFST-based ITN grammar covers only cardinal numbers, decimals, fractions, and money, excluding date, time, and measure expressions.

8. Bibliographical References

References

Imad Al-Sughaiyer and Ibrahim Al-Kharashi. 2004. Arabic morphological analysis techniques: A comprehensive survey. *JASIST*, 55:189–213.

Sarah Alkuhlani and Nizar Habash. 2011. A corpus for modeling morpho-syntactic agreement in Arabic: Gender, number and rationality. In *ACL-HLT 2011 - Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 357–362.

Ahmad Alqassas. 2017. Gender and number polarity in modern standard arabic numeral phrases. *Canadian Journal of Linguistics/Revue canadienne de linguistique*, 62(1):1–17.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, et al. 2015. Deep speech 2: End-to-end speech recognition in english and mandarin.

Suzanne Mahmoud Bardeas. 2009. *The Syntax of the Arabic DP*. Ph.D. thesis, University of York.

Ahmed El-kholy and Nizar Habash. 2011. Automatic error analysis for morphologically rich languages.

Hassan Gadalla. 2023. *Comparative Morphology of Standard and Egyptian Arabic*.

Hassan Abdel-Shafik Hassan Gadalla. 2000. Numerals in standard arabic and egyptian colloquial arabic: A comparative study. *Bulletin of the Faculty of Arts, Assiut University*, 4(2):149–183.

Yashesh Gaur, Nick Kibre, Jian Xue, Kangyuan Shu, Yuhui Wang, Issac Alphanso, Jinyu Li, and Yifan Gong. 2023. Streaming, fast and accurate on-device inverse text normalization for automatic speech recognition. In *IEEE Spoken Language Technology Workshop (SLT)*. IEEE.

Kyle Gorman. 2016. Pynini: A Python library for weighted finite-state grammar compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 75–80, Berlin, Germany. Association for Computational Linguistics.

Kyle Gorman and Richard Sproat. 2016. Minimally supervised number normalization. *Transactions of the Association for Computational Linguistics*, 4:507–519.

Nizar Habash, Ryan Roth, Owen Rambow, Ramy Eskander, and Nadi Tomeh. 2013. Morphological analysis and disambiguation for dialectal arabic. In *Proceedings of NAACL-HLT 2013*.

Nizar Y. Habash. 2010. *Introduction to Arabic Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool Publishers.

Go Inoue, Salam Khalifa, and Nizar Habash. 2022. Morphosyntactic tagging with pre-trained language models for Arabic and its dialects. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1708–1719, Dublin, Ireland. Association for Computational Linguistics.

Paarth Neekhara, Jason Huang, Kushal Lakhotia, Vitaly Lavrukhin, Boris Ginsburg, and Jagadeesh Balam. 2024. Improving robustness of llm-based speech synthesis by learning monotonic alignment. In *Interspeech 2024*.

Thai Binh Nguyen, Xin Zhang, Yufeng Li, Yashesh Gaur, and Yifan Gong. 2023. Four-in-one: A joint single-pass model for inverse text normalization, punctuation, capitalization, and disfluency. In *INTERSPEECH 2023*.

Vahid Noroozi, Somshubra Majumdar, Ankur Kumar, Jagadeesh Balam, and Boris Ginsburg. 2024. Stateful conformer with cache-based inference for streaming automatic speech recognition. In *ICASSP 2024*. IEEE.

Dima Rekesh, Nithin Rao Koluguri, Samuel Kriman, Somshubra Majumdar, Vahid Noroozi, He Huang, Oleksii Hrinchuk, Krishna C. Puvvada, Ankur Kumar, Jagadeesh Balam, and Boris Ginsburg. 2023. Fast conformer with linearly scalable attention for efficient speech recognition. In *ASRU*, pages 1–8. IEEE.

Julian Salazar et al. 2024. Handling numeric expressions in automatic speech recognition.

George Saon, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, and Phil Hall. 2017. English conversational telephone speech recognition by humans and machines.

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

Abdelhadi Soudi, Günter Neumann, and Antal Van den Bosch. 2007. *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, volume 38, pages 3–14.

Richard Sproat and Navdeep Jaitly. 2017. Rnn approaches to text normalization: A challenge.

Zhong-Qiu Tang et al. 2025. Universal-2-tf: Robust all-neural text formatting for asr.

Yingzhi Wang, Anas Alhmoud, and Muhammad Alqurishi. 2024. Open universal arabic asr leaderboard.

W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. 2017. Achieving human parity in conversational speech recognition.

Yang Zhang, Evelina Bakhturina, Kyle Gorman, and Boris Ginsburg. 2021. Nemo inverse text normalization: From development to production.

Yang Zhang, Travis M. Bartley, Mariana Graterol-Fuenmayor, Vitaly Lavrukhin, Evelina Bakhturina, and Boris Ginsburg. 2023. A chat about boring problems: Studying gpt-based text normalization. *CoRR*, abs/2309.13426.

Author Index