

grapheme-kit: Grapheme-Level Metrics and Text Processing for Multilingual NLP

Izzath Nisfer^{1,*}, Ashini Kavindya^{1,*}, Ovindu Atukorala¹,
Purushoth Velayuthan², Menan Velayuthan³

¹Department of Computer Science & Engineering, University of Moratuwa, Sri Lanka

²Computer Science and Engineering, University of Nevada, Nevada, USA

³Department of Information & Computing Sciences, Utrecht University, Utrecht, The Netherlands

*Equal contribution

Correspondence: m.velayuthan@uu.nl

Abstract

Existing lexical distance, similarity, and evaluation metrics operate on Unicode code points, which can misrepresent errors in writing systems where a single grapheme is represented by multiple Unicode code points. We introduce *grapheme-kit*, an open-source Python library that extends these metrics to operate on grapheme clusters instead. The library also provides improved grapheme processing for Tamil and Sinhala, including accurate grapheme cluster identification and grapheme composition/decomposition utilities. Through an OCR case study, we demonstrate that grapheme-level metrics provide a more faithful evaluation of complex scripts.

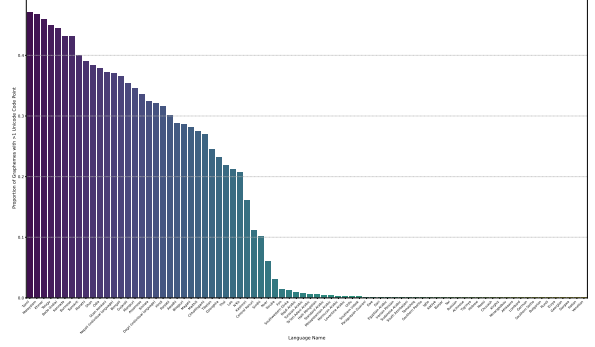


Figure 1: Proportion of grapheme clusters composed of multiple Unicode code points across FLORES+ (Gordeev et al., 2024) dataset. Languages are ordered by the computed proportion in descending order; only the first 80 are displayed.

PyPI Code Demo Docs Video

1 Introduction

“New directions in science are launched by new tools much more often than by new concepts.”

— Freeman Dyson

Modern text-based Natural Language Processing (NLP) systems operate on Unicode code points. While this representation works well for many writing systems, it becomes a major limitation for complex-script languages, where a single user-perceived character is often represented by multiple Unicode code points (Petrov et al., 2023; Velayuthan and Sarveswaran, 2025). As shown in Figure 1, a substantial number of languages exhibit a high proportion of graphemes composed of multiple Unicode code points. This disparity between writing systems that predominantly represent a character using a single Unicode code point and those that require multiple Unicode code points introduces additional computational overhead and inefficiencies in NLP systems (Petrov et al., 2023). Grapheme clusters (referred to as

graphemes hereafter) address this issue by grouping the Unicode code points that together form a single user-perceived character and treating them as a single unit.

While graphemes are extensively used in Automatic Speech Recognition (ASR) (Bunzeck et al., 2024; Dong et al., 2022; Route et al., 2019), Optical Character Recognition (OCR) (Hasan et al., 2025; Elkhayati et al., 2022), and even tokenization (Velayuthan and Sarveswaran, 2025), there remains a lack of comprehensive tooling for grapheme-level text processing and evaluation. To address this, we introduce *grapheme-kit*, a Python library for grapheme-level text processing and evaluation.

We extend key lexical metrics used in modern NLP systems to operate on graphemes instead of Unicode code points as the fundamental lexical unit. We group these metrics into three categories: (i) distance metrics, (ii) similarity metrics, and (iii) evaluation metrics. Figure 2 provides a complete overview of the features supported by our library.

Current libraries for identifying grapheme clusters fail to correctly identify certain grapheme clusters in Tamil and Sinhala (see Table 2 for examples).

We find that these errors primarily arise from the handling of Zero Width Joiner (ZWJ) and Zero Width Non-Joiner (ZWNJ) characters. Our library rectifies these issues for both languages, enabling correct grapheme cluster identification. We also observe similar issues in other Indic scripts, and our library is designed to be easily extended by language experts to support additional languages.

Vowel–consonant decomposition plays an important role in fine-grained linguistic analysis (Ansary et al., 2024). To support such analyses, our library provides utilities to decompose Tamil and Sinhala graphemes into their constituent consonant and vowel components, as well as reconstruct graphemes from these components. The same design can be readily extended to support other Indic scripts.

Our core contributions are as follows:

- We introduce `grapheme-kit`, an open-source Python library that provides a comprehensive toolkit for grapheme-aware text processing and evaluation across languages.
- We extend key lexical metrics to operate on graphemes instead of Unicode code points, supporting distance, similarity, and evaluation metrics.
- We improve grapheme processing for Tamil and Sinhala by correcting grapheme cluster identification and providing utilities for vowel–consonant decomposition, while designing the library to be easily extended to other Indic scripts.
- We make the library¹ publicly available through PyPI, accompanied by comprehensive documentation² and an interactive live demo³ to facilitate adoption by the research community.

2 Related work

2.1 Graphemes

Graphemes are the fundamental units of writing (Meletis, 2019). However, modern NLP systems predominantly operate on Unicode code points and subword tokenization algorithms such as Byte Pair Encoding (BPE) and Unigram language models (Sennrich et al., 2016; Kudo, 2018), which often

fail to preserve the orthographic structure of complex scripts.

In scripts such as Tamil and Sinhala, a single grapheme is typically represented by multiple Unicode code points comprising base consonants, vowel signs, and other combining characters. Consequently, conventional tokenization algorithms frequently split graphemes into incomplete units, resulting in fragmented linguistic representations and longer token sequences (Petrov et al., 2023; Velayuthan and Sarveswaran, 2025). Recent work has therefore proposed grapheme-aware tokenization methods such as Grapheme Pair Encoding (GPE) and Constrained Byte Pair Encoding (CBPE) (Velayuthan and Sarveswaran, 2025; Brahma et al., 2025).

Existing grapheme segmentation libraries provide only partial support for Indic scripts and fail to correctly handle several orthographic patterns in Tamil and Sinhala. We compare these libraries in Table 1.

2.2 Evaluation metrics

Automatic evaluation metrics are fundamental to measuring the quality of NLP systems (Celikyilmaz et al., 2021). Many lexical metrics are built upon string comparison techniques such as Hamming distance and Levenshtein distance (Hamming, 1950; Levenshtein, 1965).

Character Error Rate (CER) extends Levenshtein distance and serves as the standard evaluation metric for Automatic Speech Recognition (ASR) and text normalization (K et al., 2024; Ridoy et al., 2025; Bekarystankyzy et al., 2024; Ghimire et al., 2025). For machine translation and text generation, BLEU measures word-level n -gram overlap, while chrF and chrF++ operate on character n -grams and correlate well with human judgments, particularly for morphologically rich and low-resource languages (Papineni et al., 2002; Popović, 2015; Popovic, 2016).

Despite their widespread adoption, these metrics fundamentally operate on Unicode code points, characters, or words as their lexical units. To the best of our knowledge, there exists no unified toolkit that extends these commonly used lexical metrics to operate directly on grapheme clusters.

2.3 Sinhala and Tamil Languages

Sinhala is an Indo-Aryan language spoken by over 17 million people in Sri Lanka, while Tamil is a Dravidian language spoken by more than 78 million

¹<https://pypi.org/project/grapheme-kit/>

²<https://grapheme-kit-docs.pages.dev/>

³<https://grapheme-kit.pages.dev/>

Feature / Limitation	Grapheme	U-grapheme	Indic NLP	Regex	grapheme-kit
Sinhala support	'කීu200d', 'ඊ', 'ම', 'ය'	'කීu200d', 'ඊ', 'ම', 'ය'	'කී', 'u200d', 'ඊ', 'ම', 'ය'	'කීu200d', 'ඊ', 'ම', 'ය'	'කීu200dඊ', 'ම', 'ය'
Tamil support	'ஸ்', 'ரீ', ' ', 'ல', 'ங்', 'கா'	'ஸ்', 'ரீ', ' ', 'ல', 'ங்', 'கா'	'ஸ்ரீ', 'ல', 'ங்கா'	'ஸ்', 'ரீ', ' ', 'ல', 'ங்', 'கா'	'ஸ்ரீ', ' ', 'ல', 'ங்', 'கா'

Table 1: Comparison of libraries for grapheme-level segmentation in Sinhala and Tamil

people across South Asia and the global Tamil diaspora (Krishnamurti, 2003; De Silva, 2019). Despite their large speaker populations, both remain low-resource languages in NLP research (Ranathunga and de Silva, 2022; Tissera and Saadhiq, 2025). Both writing systems are derived from the ancient Brahmi script and belong to the abugida family, where graphemes are formed by combining base consonants with vowel signs and other combining characters (Kunchukuttan et al., 2021). Consequently, a single user-perceived character is often represented by multiple Unicode code points, making Unicode-based tokenization and evaluation less suitable for these languages and motivating grapheme-aware processing (Petrov et al., 2023; Velayuthan and Sarveswaran, 2025; Fernando and Ranathunga, 2025).

3 System Overview

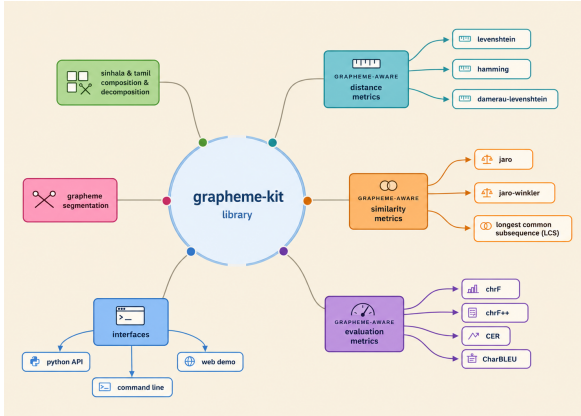


Figure 2: Overview of the grapheme-kit library architecture. Inspiration for the diagram was taken from (Suzgun et al., 2024).

3.1 Architecture Overview

Raw input text first passes through a normalization stage that resolves Unicode level inconsistencies and language specific encoding irregularities. The normalized text is then segmented into grapheme clusters using graphemizer. Then these clusters are

used to implement grapheme aware string distances, grapheme level evaluation metrics, and composition and decomposition of clusters into their constituent base consonant and vowel components. An overview of this workflow is illustrated in Figure 2.

3.2 Graphemizer

Graphemizer builds on top of grapheme library. When code points go through graphemizer, it identifies specific cases where grapheme library fails and gives exact visual units in scripts (refer Table 2).

In Sinhala there are some complex consonant clusters by joining a sequence of consonants through ZWJ characters, each carrying its own virama and vowel modifiers. Grapheme library treats a ZWJ as a boundary between two separate clusters. So a single visually and linguistically unified character gets split into multiple grapheme clusters. Graphemizer detects ZWJ linked sequences and recursively folds every consonant, virama, and vowel modifier joined by ZWJ into a single grapheme cluster regardless of how many ZWJs are involved in the chain.

3.3 Grapheme Level Metrics

Grapheme level metrics evaluate and compare strings using grapheme clusters rather than individual Unicode code points

For example, consider the hindi words किताब (book) and कताब. Their grapheme cluster representations are कि, ता, ब and क, ता, ब.

Although the Unicode representations differ by several individual code points, the grapheme level representation groups each user perceived character into a single unit. metrics operate on those visible characters perceived by readers.

This library implements several distances, similarity metrics, and evaluation metrics in grapheme level.

- **Distance Metrics:** Levenshtein, Hamming, Damerau-Levenshtein

exposes the measurement bias introduced by Unicode code point-based evaluation. While our experiments are limited to OCR, the proposed metrics are applicable to other tasks that rely on lexical evaluation, such as Automatic Speech Recognition (ASR) and Machine Translation (MT). We evaluate across 12 languages spanning Latin controls (Dutch and French), Arabic, and scripts with substantial Unicode decomposition (Hindi, Bengali, Tamil, Telugu, Malayalam, Sinhala, Sanskrit, Dzongkha, and Burmese). OCR outputs are obtained by rendering FLORES+ (Gordeev et al., 2024) sentences using Noto fonts and recognizing them with Tesseract.

Script complexity. We quantify the degree of Unicode decomposition using the *grapheme inflation ratio*, defined as the average ratio between grapheme clusters and Unicode code points per reference sentence. A ratio close to 1.0 indicates that a visual character is typically represented by a single Unicode code point, whereas lower values indicate that graphemes are composed of multiple code points. As shown in Table 3, Telugu, Burmese, Malayalam, Tamil, Hindi, Sanskrit, Bengali, and Sinhala exhibit substantial decomposition (0.59–0.65), while Arabic (0.99) behaves similarly to Latin scripts and Dzongkha (0.76) lies in between. Such scripts are expected to benefit most from grapheme-level metrics, as a single grapheme spanning multiple Unicode code points may otherwise incur multiple edit operations under codepoint-level evaluation.

Language	Inflation Ratio
Telugu	0.59
Burmese	0.60
Malayalam	0.60
Tamil	0.61
Hindi	0.62
Sanskrit	0.62
Bengali	0.64
Sinhala	0.65
Dzongkha	0.76
Arabic	0.99
Dutch	1.00
French	1.00

Table 3: Grapheme inflation ratio across languages. Lower values indicate greater Unicode decomposition.

OCR: Grapheme-level metrics correct a substantial measurement bias. OCR is the setting where this bias is most apparent, as OCR errors often arise from misrecognizing a single component within an otherwise correct grapheme cluster (e.g., a dependent vowel sign). Under codepoint-level evaluation,

such errors are disproportionately penalized despite affecting only a single user-perceived character. Table 4 reports CER and chrF++ (word bigrams + character n -grams, with $n=3$ for the grapheme variant) and shows consistent improvements under grapheme-aware evaluation. As expected, languages where grapheme clustering has little effect (Arabic and the Latin controls) show virtually no change. In contrast, Burmese and Dzongkha exhibit a small decline, as the OCR system frequently fails to produce text resembling the reference, leaving little component-level information for grapheme-aware evaluation to recover.

Language	char-CER	g-CER	chrF++	g-chrF++	Δ chrF++
Tamil	5.48	0.62	74.05	98.99	+24.94
Sinhala	3.80	0.92	86.45	98.88	+12.43
Malayalam	11.35	9.68	72.55	78.14	+5.59
Bengali	6.36	4.59	89.27	93.86	+4.59
Telugu	3.38	3.09	90.76	95.30	+4.54
Sanskrit	3.53	3.24	87.59	92.01	+4.42
Hindi	0.70	0.71	98.30	98.41	+0.11
Dutch	0.13	0.13	99.52	99.52	+0.00
French	0.58	0.58	98.51	98.51	+0.00
Arabic	82.96	83.32	23.46	23.38	−0.08

Table 4: OCR results: standard vs. grapheme-level metrics (chrF++ at tuned grapheme $n=3$). Languages sorted by improvement.

4.2 Controlled Perturbation Analysis

The case study above evaluates metrics on real model outputs, where script complexity, model quality, and error types are inherently coupled. To isolate the effect of grapheme-level evaluation, we construct a controlled perturbation study following prior work on metric diagnostics, which emphasizes evaluating metrics under targeted edits of known types rather than relying only on aggregate correlations (Karpinska et al., 2022). We apply a set of controlled perturbations (ref Appendix B.1) directly to the same FLORES+ reference sentences used above (1,012 sentences per language across all 12 languages), allowing the same edit operation to be compared across scripts with varying degrees of grapheme inflation. The perturbations include word-level modifications (repetition, reordering, and shuffling), a realistic misspelling through vowel-sign substitution, two character-deletion variants, and sanity-check baselines (empty hypothesis, unrelated sentence, and reference-as-hypothesis).

A minimal-pair comparison isolates the mechanism directly. We focus on a character deletion perturbation implemented at two different levels:

codepoint-level deletion, where a single Unicode code point is removed (e.g., a dependent vowel sign), and grapheme-level deletion, where an entire grapheme cluster is removed. The former corrupts an otherwise intact grapheme cluster, while the latter removes a complete linguistic unit. Table 5 shows that these two operations produce opposite biases under character-level evaluation. For codepoint-level deletion, grapheme-CER assigns higher error than char-CER (+0.60 points averaged across the ten non-Arabic complex scripts), reflecting that the corrupted grapheme is a visible error despite involving only a single codepoint change. Conversely, for grapheme-level deletion, grapheme-CER assigns lower error (−0.31), as it treats the removal of a complete grapheme as a single edit, while char-CER may count multiple codepoint edits. Latin controls (Dutch and French) show no difference, as grapheme clustering is effectively a no-op. These results demonstrate that the observed differences are specific to scripts with grapheme inflation and arise from the choice of evaluation unit rather than the edit-distance computation itself.

Perturbation	CER	g-CER	Δ CER	chrF	g-chrF (Δ)
Misspelled word (25)	1.22	1.12	−0.10	97.89	97.31 (−0.58)
Delete one codepoint (26a)	0.82	1.42	+0.60	98.32	97.19 (−1.14)
Delete one grapheme (26b)	1.42	1.12	−0.31	97.70	97.59 (−0.12)

Table 5: Controlled perturbations (mean over 12 languages; chrF at tuned grapheme $n=3$). Δ CER = grapheme − standard; Δ chrF = grapheme − standard.

chrF: consistent for corruption, an open question for clean substitution. Grapheme-chrF agrees with grapheme-CER on the codepoint-deletion perturbation (26a), assigning lower scores to the corrupted outputs than standard chrF. However, for the clean-substitution perturbations (25, 26b), grapheme-chrF does not follow the more forgiving behaviour observed in grapheme-CER and instead assigns lower scores than standard chrF. This difference arises from the underlying formulation of the metrics. Grapheme-CER benefits from treating a multi-codepoint grapheme as a single edit unit, reducing the edit penalty for clean substitutions. In contrast, chrF measures n -gram overlap rather than discrete edit operations; therefore, a changed grapheme can disrupt similar numbers of n -gram matches regardless of whether it spans one or multiple codepoints. Since grapheme-level n -grams operate over a smaller unit space, localized changes can represent a larger fraction of the overall

n -gram overlap, causing grapheme-chrF to move more than standard chrF. We leave further investigation of this behaviour and potential normalization strategies for grapheme-chrF as future work.

5 Conclusion

We present grapheme-kit, an easy-to-use Python library for grapheme-level text processing and evaluation. We extend widely used lexical metrics to operate on grapheme clusters and provide language-aware grapheme processing utilities for complex scripts. Through multilingual case studies across machine translation, automatic speech recognition, and optical character recognition, we demonstrate the benefits of grapheme-level evaluation for scripts with substantial Unicode decomposition. We hope that grapheme-kit provides an accessible foundation for developing and evaluating multilingual NLP systems using linguistically meaningful text units.

Limitations

We have currently identified two limitations in our library. First, the accuracy of our evaluation metrics depends on the underlying grapheme segmentation library. Second, the composition and decomposition features are available for only two languages at present. We are actively working with a group of excellent volunteers to extend these features to more Indic languages.

Acknowledgments

We, the authors, would like to thank the developers and maintainers of the open-source libraries that supported our implementation, specifically the grapheme library⁴ for Unicode grapheme cluster segmentation, the textdistance library⁵ for string distance and similarity computations, and the sacrebleu library⁶ for evaluation metrics.

References

Nazmuddoha Ansary, Quazi Adibur Rahman Adib, Tahsin Reasat, Asif Shahriyar Sushmit, Ahmed Imtiaz Humayun, Sazia Mehnaz, Kanij Fatema, Mohammad Mamun Or Rashid, and Farig Sadeque. 2024. [Unicode normalization and grapheme parsing of Indic languages](#). In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING*

⁴<https://github.com/itcarroll/grapheme>

⁵<https://github.com/orsinium/textdistance>

⁶<https://github.com/mjpost/sacrebleu>

- 2024), pages 17019–17030, Torino, Italia. ELRA and ICCL.
- A. Bekarystankyzy, Orken J. Mamyrbayev, Mateus Mendes, A. Fazylzhanova, and Muhammad Assam. 2024. [Multilingual end-to-end asr for low-resource turkic languages with common alphabets](#). *Scientific Reports*, 14.
- Maharaj Brahma, NJ Karthika, Atul Singh, Devaraj Adiga, Smruti Bhate, Ganesh Ramakrishnan, Rohit Saluja, and Maunendra Sankar Desarkar. 2025. Morphtok: Morphologically grounded tokenization for indian languages. *arXiv preprint arXiv:2504.10335*.
- Bastian Bunzeck, Daniel Duran, Leonie Schade, and Sina Zarrieß. 2024. [Graphemes vs. phonemes: battling it out in character-based language models](#). In *The 2nd BabyLM Challenge at the 28th Conference on Computational Natural Language Learning*, pages 54–64, Miami, FL, USA. Association for Computational Linguistics.
- Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. 2021. [Evaluation of text generation: A survey](#). *Preprint*, arXiv:2006.14799.
- Nisansa De Silva. 2019. Survey on publicly available sinhala natural language processing tools and research. *arXiv preprint arXiv:1906.02358*.
- Lu Dong, Zhi-Qiang Guo, Chao-Hong Tan, Ya-Jun Hu, Yuan Jiang, and Zhen-Hua Ling. 2022. [Neural grapheme-to-phoneme conversion with pre-trained grapheme models](#). In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6202–6206.
- Mohsine Elkhayati, Youssfi Elkettani, and Mohammed Mourchid. 2022. [Segmentation of handwritten arabic graphemes using a directed convolutional neural network and mathematical morphology operations](#). *Pattern Recognition*, 122:108288.
- Aloka Fernando and Surangika Ranathunga. 2025. Linguistic entity masking to improve cross-lingual representation of multilingual language models for low-resource languages: A. fernando, s. ranathunga. *Knowledge and Information Systems*, 67(11):9905–9946.
- Rupak Raj Ghimire, Prakash Poudyal, and Bal Krishna Bal. 2025. [Improving accuracy of low-resource ASR using rule-based character constituency loss \(RBCCCL\)](#). In *Proceedings of the First Workshop on Challenges in Processing South Asian Languages (CHiPSAL 2025)*, pages 61–70, Abu Dhabi, UAE. International Committee on Computational Linguistics.
- Isai Gordeev, Sergey Kuldin, and David Dale. 2024. [FLORES+ translation and machine translation evaluation for the Erzya language](#). In *Proceedings of the Ninth Conference on Machine Translation*, pages 614–623, Miami, Florida, USA. Association for Computational Linguistics.
- R. W. Hamming. 1950. [Error detecting and error correcting codes](#). *The Bell System Technical Journal*, 29(2):147–160.
- Md. Mahmudul Hasan, Ahmed Nesar Tahsin Choudhury, Mahmudul Hasan, and Md Mosaddek Khan. 2025. [GraDeT-HTR: A resource-efficient Bengali handwritten text recognition system utilizing grapheme-based tokenizer and decoder-only transformer](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 696–706, Suzhou, China. Association for Computational Linguistics.
- Thennal D K, Jesin James, Deepa P Gopinath, and Muhammed Ashraf K. 2024. [Advocating character error rate for multilingual asr evaluation](#). *Preprint*, arXiv:2410.07400.
- Marzena Karpinska, Nishant Raj, Katherine Thai, Yixiao Song, Ankita Gupta, and Mohit Iyyer. 2022. Demetr: Diagnosing evaluation metrics for translation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9540–9561.
- Bhadriraju Krishnamurti. 2003. *The Dravidian Languages*. Cambridge Language Surveys. Cambridge University Press.
- Taku Kudo. 2018. [Subword regularization: Improving neural network translation models with multiple subword candidates](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.
- Anoop Kunchukuttan, Siddharth Jain, and Rahul Kejriwal. 2021. [A large-scale evaluation of neural machine transliteration for Indic languages](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3469–3475, Online. Association for Computational Linguistics.
- Vladimir I. Levenshtein. 1965. [Binary codes capable of correcting deletions, insertions, and reversals](#). *Soviet physics. Doklady*, 10:707–710.
- Dimitrios Meletis. 2019. [The grapheme as a universal basic unit of writing](#). *Writing Systems Research*, 11(1):26–49.
- Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. [Bleu: a method for automatic evaluation of machine translation](#). In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.
- Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. *Advances in neural information processing systems*, 36:36963–36990.

- Maja Popović. 2015. [chrF: character n-gram F-score for automatic MT evaluation](#). In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal. Association for Computational Linguistics.
- Maja Popovic. 2016. [chrf deconstructed: beta parameters and n-gram weights](#). pages 499–504.
- Surangika Ranathunga and Nisansa de Silva. 2022. [Some languages are more equal than others: Probing deeper into the linguistic disparity in the NLP world](#). In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 823–848, Online only. Association for Computational Linguistics.
- Md Sazzadul Islam Ridoy, Sumi Akter, and Md Aminur Rahman. 2025. Adaptability of asr models on low-resource language: A comparative study of whisper and wav2vec-bert on bangla. In *2025 2nd International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM)*, pages 1–6. IEEE.
- James Route, Steven Hillis, Isak Czeresnia Etinger, Han Zhang, and Alan W Black. 2019. [Multimodal, multilingual grapheme-to-phoneme conversion for low-resource languages](#). In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 192–201, Hong Kong, China. Association for Computational Linguistics.
- 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.
- Mirac Suzgun, Stuart Shieber, and Dan Jurafsky. 2024. [string2string: A modern python library for string-to-string algorithms](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 278–285, Bangkok, Thailand. Association for Computational Linguistics.
- Muditha Tissera and Hassaan Saadhiq. 2025. [Natural language processing resources for tamil language: A systematic review](#). *Journal of information and communication convergence engineering*, 23:236–258.
- Menan Velayuthan and Kengatharaiyer Sarveswaran. 2025. [Egalitarian language representation in language models: It all begins with tokenizers](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5987–5996, Abu Dhabi, UAE. Association for Computational Linguistics.

A Text Pre-processing

A.1 Normalization

grapheme-kit provides a language aware normalization module for Tamil and Sinhala to ensure that visually and linguistically equivalent text is represented in a canonical form before further processing. The normalization pipeline consists of two stages.

Unicode Normalization The input text is first transformed into Unicode Normalization Form C (NFC), which applies canonical composition to combine decomposed Unicode sequences into their standard composed representation. This step guarantees a consistent Unicode encoding for equivalent character sequences.

Language specific Normalization. After NFC normalization, a set of rule based transformations is applied to correct script specific inconsistencies. For Tamil, these rules normalize alternative vowel sign sequences, remove Zero Width Joiner (ZWJ) and Zero Width Non-Joiner (ZWNJ) characters, correct reversed vowel ordering, and replace non standard character combinations with their canonical forms. For example, the sequence ௫௫௫ is normalized to ௫௫௫ , and ௫௫ is normalized to ௫௫ .

Similarly, Sinhala normalization applies a confusion set mapping to correct frequently occurring ordering and composition errors involving dependent vowel signs and virama characters. For example, the non-canonical sequence ඳ්ඳ් is normalized to ඳ්ඳ් , and ඳ්ඳ් is normalized to ඳ්ඳ් . These transformations ensure that equivalent grapheme clusters are represented consistently, improving the reliability of grapheme segmentation and subsequent evaluation metrics.

B Evaluations

B.1 Controlled perturbation

We adapt a subset of Karpinska et al. (2022)’s (DEMETER) perturbation taxonomy. Most of the perturbations are word-level edits, where the codepoint-versus-grapheme distinction is immaterial, since word boundaries always contain whole grapheme clusters. We focus the main analysis on the three that vary edit granularity directly:

- **25 - Misspelled word:** one grapheme cluster’s vowel is substituted.
- **26a - Delete one codepoint:** corrupts a multi-codepoint cluster.

- **26b - Delete one grapheme cluster:** removes a whole, possibly multi-codepoint grapheme cluster.