A Comparative Analysis of Tokenization Methods for Sinhala Natural Language Processing

Ransaka Ravihara *

Posted Date: 7 August 2025

doi: 10.20944/preprints202508.0561.v1

Keywords: machine learning; natural language processing; low resource language; tokenization

 doi:10.20944/preprints202508.0561.v1

Article

A Comparative Analysis of Tokenization Methods for Sinhala Natural Language Processing

R R M Withanachchi

University of Moratuwa; withanachchirrm.25@uom.lk

Abstract

Tokenization is a foundational step in Natural Language Processing (NLP), yet its impact on morphologically rich, low-resource languages like Sinhala is not well understood. This paper presents a systematic evaluation of five tokenization strategies—Byte, Character, Grapheme Cluster, WordPiece, and Word-level—to determine their effect on downstream task performance and computational efficiency. We train and assess Transformer-based models on four datasets: a clean baseline, and three variants synthetically corrupted with minor typos, aggressive typos, and code-mixing to simulate real-world text. Our results reveal a critical trade-off. Word-level tokenization achieves the highest F1-score (0.727) on clean text and is the most computationally efficient, but its performance degrades significantly on noisy text. Conversely, WordPiece demonstrates superior robustness, maintaining high performance across all conditions, making it the most reliable choice for real-world applications, albeit at a higher computational cost. Grapheme Cluster tokenization emerges as a strong, balanced alternative. This study provides crucial empirical evidence to guide the selection of tokenizers for Sinhala NLP, establishing a baseline for performance, robustness, and efficiency.

Keywords: machine learning; natural language processing; low resource language; tokenization

1. Introduction

Tokenization, the process of segmenting raw text into fundamental units, is a critical initial step in any Natural Language Processing (NLP) pipeline. The chosen strategy directly dictates how linguistic information is structured for a model, thereby influencing its performance, robustness, and computational demands. While subword-based tokenizers like WordPiece [1] and Byte Pair Encoding (BPE) [2] are standard for high-resource languages, their efficacy for morphologically complex, low-resource languages such as Sinhala remains underexplored.

This research addresses this gap by investigating a central question: **What is the optimal tokenization strategy for Sinhala NLP, considering the trade-off between model performance, robustness to textual noise, and computational efficiency?**

To answer this, we conduct a comparative analysis of five distinct tokenization methods: Byte-level, Character-level, Grapheme Cluster, WordPiece (subword), and Word-level. We evaluate these by training Transformer-based classification models from scratch on four datasets designed to mirror real-world conditions: a clean dataset from social media, and three variants synthetically generated to include typographical errors and code-mixing (the use of Romanized "Singlish").

Although pre-training large language models (e.g., BERT [3]) from scratch for each tokenizer would be ideal, the prohibitive computational cost necessitates a more focused approach. This study therefore provides a rigorous evaluation of tokenizer effectiveness within a controlled experimental framework, offering a vital baseline for researchers and practitioners building robust NLP systems for the Sinhala language.

 doi:10.20944/preprints202508.0561.v1

2. Methodology

The core of our methodology involves a systematic comparison of five distinct tokenization strategies in the four different Sinhala NLP corpus. This results in a total of 20 experimental setups (5 tokenizers $\times$ 4 datasets), allowing for a comprehensive analysis of the interplay between tokenization and task performance.

2.1. Tokenization Strategies

We evaluate five tokenization methods, ranging from simple character splits to complex subword segmentation.

2.1.1. Character-Level Tokenization

This is the most granular tokenization method, where the input text is decomposed into individual characters. While this approach creates a small, fixed vocabulary and avoids any out-of-vocabulary (OOV) issues, it disassembles semantically meaningful units (like words or morphemes) into components that lack intrinsic context.

Figure 1. An example of Character-Level Tokenization in Sinhala.

2.1.2. Grapheme Cluster Tokenization

Unlike simple character splitting, Grapheme Cluster Tokenization splits text into visually and linguistically cohesive units. In Sinhala, a single perceived "letter" is often composed of a base consonant and one or more diacritics (vowel signs, or "pili"). This tokenizer correctly groups these components into a single token, which preserves more semantic meaning than an individual character.

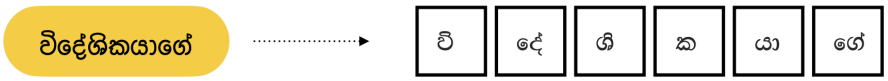

Figure 2. An example of Grapheme Cluster Tokenization, which correctly groups base consonants with their diacritics.

2.1.3. Byte-Level Tokenization

This method tokenizes text at the raw byte level based on its UTF-8 encoding. As Sinhala characters are typically represented by 3 bytes, this results in significantly longer token sequences. The primary advantage is the ability to model any text without encountering OOV tokens. The trade-off is that it creates long sequences and splits fundamental grapheme clusters into meaningless byte tokens, posing a challenge for the model.

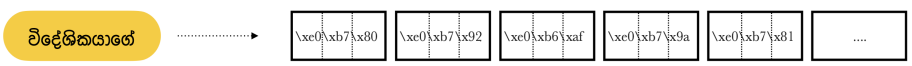

Figure 3. An example of Byte-Level Tokenization, showing how a single grapheme is split into multiple byte tokens.

2.1.4. Subword-Based Tokenization (WordPiece)

Subword tokenization is the de facto standard in modern NLP. We use the **WordPiece** algorithm, notably employed by BERT [3]. WordPiece starts with a vocabulary of individual characters and iteratively merges frequent pairs of subword units to maximize the likelihood of the training corpus. This method strikes a balance between character- and word-level approaches, keeping common words

 doi:10.20944/preprints202508.0561.v1

intact while breaking rare words into meaningful subword units, effectively handling morphology and reducing the OOV problem.

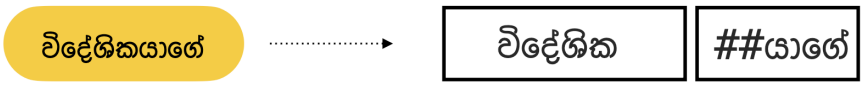

Figure 4. An example of WordPiece subword tokenization.

2.1.5. Word-Based Tokenization

We also evaluate a traditional word-based tokenizer that splits text based on spaces and punctuation. While this method produces the shortest token sequences, it suffers from the **out-of-vocabulary (OOV)** problem, where any word not seen during training is mapped to a single "unknown" token. This is particularly problematic for morphologically rich languages like Sinhala. Nevertheless, it is included in our analysis to provide a comprehensive baseline.

3. Model Architecture

The classification model used for all experiments is based on the Transformer encoder architecture introduced by Vaswani et al. [4]. As depicted in Figure 5, our classification model comprises three main components: an embedding layer, a stack of Transformer encoder blocks, and a final classification head.

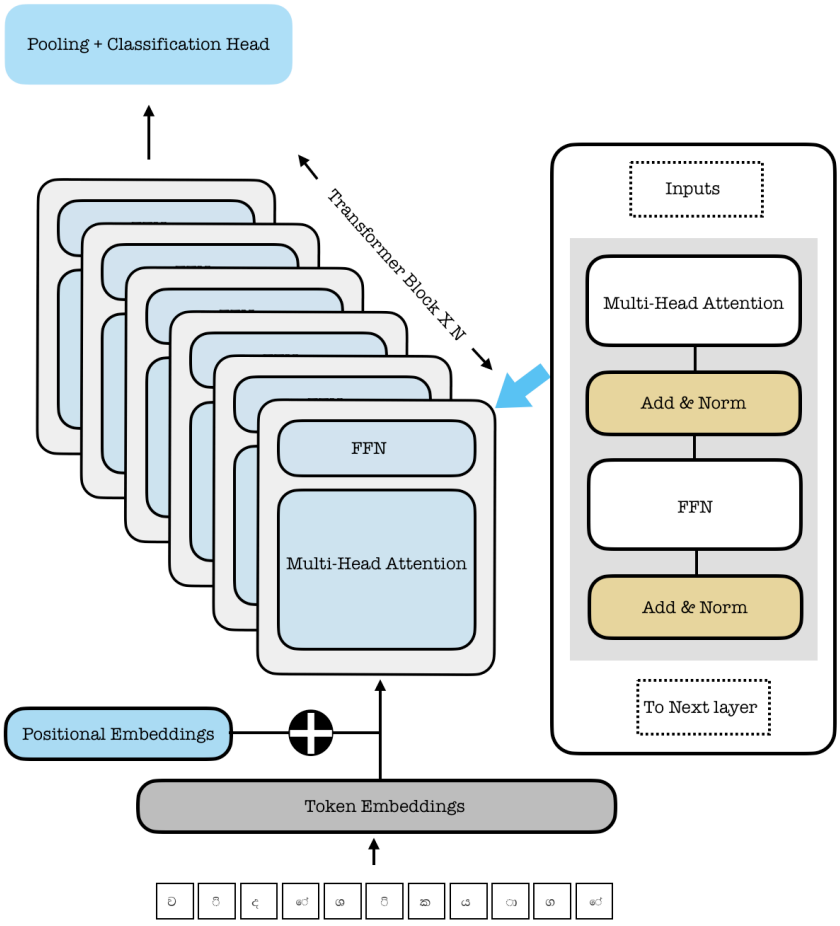

Figure 5. The Transformer-based classification architecture. Input tokens are converted into embeddings, processed by a stack of N identical encoder layers, and the final representation of the first token is passed to a linear layer for classification.

 doi:10.20944/preprints202508.0561.v1

First, the input sequence of token IDs is converted into dense vector representations by the *Token Embedding* layer. To provide the model with information about the order of the tokens, these embeddings are added element-wise with *Positional Embeddings*. The resulting combined embeddings are passed through a stack of *N identical Transformer Blocks*. Each block contains two primary sub-layers: a **Multi-Head Self-Attention** mechanism and a **position-wise Feed-Forward Network (FFN)**. A residual connection followed by layer normalization is applied after each sub-layer.

Finally, for classification, the output representation of the first token in the sequence (i.e., the '[CLS]' token) from the final encoder block is used as an aggregate representation for the entire sequence. This vector is passed through a linear layer followed by a softmax function to produce a probability distribution over the target classes.

4. Experiment Setup

4.1. Datasets and Data Augmentation

To assess tokenizer effectiveness for a low-resource language in a manner that reflects real-world use cases, we used the original SOLD [5] dataset as a baseline and created three synthetic variants. Real-world text data often contains typographical errors and code-mixing. Therefore, our data augmentation process was designed to simulate these practical concerns by generating the following datasets:

1. **Minor Typos:** Each character in the original text has a 5% probability of being randomly deleted, replaced, or swapped with an adjacent character.
2. **Aggressive Typos:** The character-level error probability is increased to 10%.
3. **Code-Mixing and Typos:** This variant combines minor typos (5% character error rate) with code-mixing, where each word has a 30% probability of being transcribed into its romanized form ("Singlish").

An example of the data augmentation is shown below:

- **Original:** තේ නෙවෙයි තෝ බීලා ඉන්නෙ ගිනි වතුර
- **Aggressive Typos:** තේ නෙවෙයිත ෝ බලා ඉන්්නෙ ගිනි වතුර
- **Code-Mixing:** තේ නෙවෙයි tho බීලා ඉන්නෙ gini වනතුර

4.2. Model Training and Hyperparameters

For each of the 20 experiments, we train the Transformer-based classification model from scratch. While the vocabulary size varies with the tokenizer, the core architecture hyperparameters are kept constant to ensure a fair comparison. All models are trained using the AdamW optimizer. Model performance is evaluated using the macro F1-score, which is suitable for potentially imbalanced datasets. The key hyperparameters are detailed in Table 1.

Table 1. Key hyperparameters for the Transformer model.

Hyperparameter	Value
Hidden Size (d_{model})	512
FFN Intermediate Size	512
Number of Attention Heads	8
Dropout Probability	0.2
Learning Rate	5×10^{-5}
Max Sequence Length	512
Optimizer	AdamW
Weight Decay	0

5. Results and Discussion

This section analyzes our empirical findings, focusing on the trade-offs between performance, robustness across different data conditions, and computational cost.

 doi:10.20944/preprints202508.0561.v1

Discussion

Our results reveal a clear and critical trade-off between performance, robustness, and computational efficiency. We discuss these findings and their practical implications below.

Performance vs. Robustness

On clean, well-formed text ('Original' dataset), the **Word-level** tokenizer achieves the highest F1-score (0.7274). This indicates that when the vocabulary is stable, preserving whole words as semantic units is most effective. However, its performance is brittle, degrading significantly as data quality worsens.

Conversely, tokenizers that operate at a sub-word level demonstrate superior robustness. **WordPiece (WPE)** excels in noisy conditions, becoming the top performer on the 'Aggressive Typos' dataset (F1=0.7000). By breaking unknown words into known sub-units, it effectively mitigates the out-of-vocabulary problem that cripples the Word-level tokenizer.

Performance Stability Across Tasks

The standard deviation reported in Table 2 measures how much a tokenizer's performance varies across the four different datasets. A low standard deviation indicates high stability. The **Byte-level** tokenizer is the most stable (Std. Dev. = 0.0060), but this is stability at a low performance level. More importantly, **WordPiece** shows remarkable stability (Std. Dev. = 0.0142) at a high performance level, making it a reliable choice for diverse applications. The relatively high deviation for Word (0.0215) and GCT (0.0250) highlights their sensitivity to changes in data quality.

Table 2. F1-Scores for all tokenization methods across four datasets. The highest F1-score in each row is in bold. The final row shows the standard deviation of F1-scores for each tokenizer **across the four tasks**, indicating performance stability under varying data conditions.

Dataset	Byte	Char	GCT	Word	WPE
Original	0.6580	0.6671	0.7073	**0.7274**	0.7100
Minor Typos	0.6566	0.6651	0.7007	**0.7044**	0.7000
Aggressive Typos	0.6604	0.6785	0.6987	0.6884	**0.7000**
Mixed Coding	0.6468	0.6526	0.6528	**0.6779**	0.6776
Std. Dev. (across tasks)	0.0060	0.0106	0.0250	0.0215	0.0142

The Efficiency-Robustness Trade-off

Figure 6 starkly illustrates the computational cost of robustness. There is an inverse relationship between representational granularity and training speed. The Word-level tokenizer is the fastest by a significant margin, making it attractive for rapid prototyping. The most robust methods, WPE and Byte-level, are the slowest due to the much longer token sequences they produce, which increases the computational load on the Transformer architecture.

 doi:10.20944/preprints202508.0561.v1

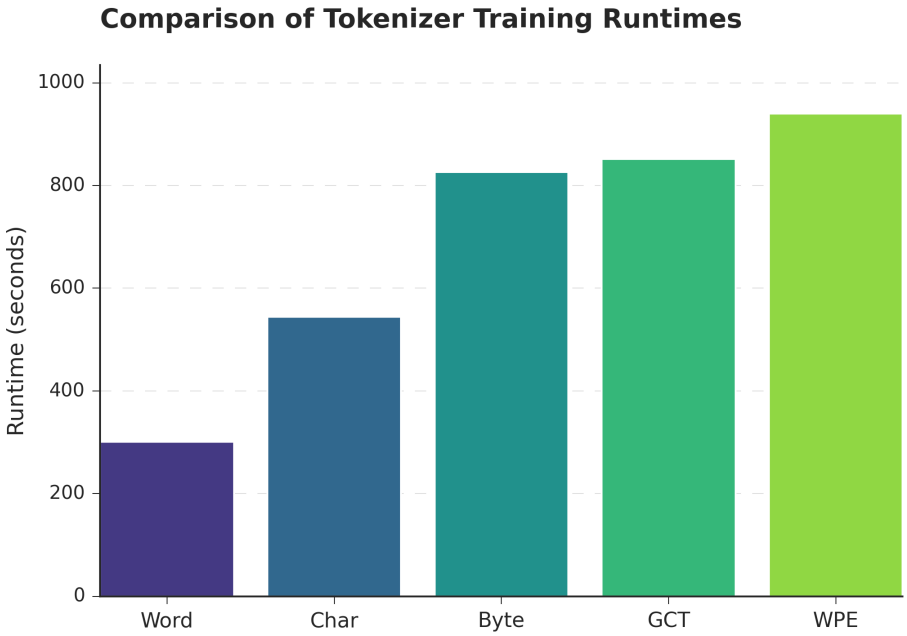

Figure 6. Total training time comparison. The Word-level tokenizer is over 3x faster to train than the most robust tokenizer, WPE, highlighting a critical performance-efficiency trade-off.

6. Conclusions

This study provides a comprehensive analysis of five tokenization methods for Sinhala NLP, evaluating the trade-offs between performance, robustness, and efficiency.

The **Word-level** tokenizer is fastest and best on clean text but is too brittle for noisy, real-world data. Conversely, **WordPiece (WPE)** emerges as the most effective all-around strategy, offering the best robustness against textual corruption, making it the premier choice for production systems despite its high computational cost. For applications where this cost is prohibitive, the **Grapheme Cluster (GCT)** tokenizer presents an excellent, balanced alternative.

Future work should extend this analysis by pre-training large language models from scratch for each tokenization strategy to explore their full potential. Nonetheless, this study provides a crucial empirical foundation for making informed, scenario-aware decisions when developing NLP systems for Sinhala and other morphologically rich, low-resource languages.

References

1. Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
2. Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.
3. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
4. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, 2017.
5. Ranasinghe, T., Anuradha, I., Premasiri, D. et al. SOLD: Sinhala offensive language dataset. *Lang Resources & Evaluation 59*, 297–337 (2025). https://doi.org/10.1007/s10579-024-09723-1