The Syllable is the Token: Breaking the Token Tax with SGPE

Kusal Darshana
Remeinium Research
`thekusaldarshana@remeinium.com`

February 20, 2026

Abstract

Large Language Models (LLMs) suffer from a fundamental "linguistic blindness" when processing Abugida scripts, imposing a significant "Token Tax" on the Global South. Current tokenization methods, such as Byte-Pair Encoding (BPE), routinely fragment complex conjuncts—multi-codepoint grapheme clusters—into meaningless sub-character units, degrading model reasoning and inflating inference costs. We introduce Syllable-aware Grapheme Pair Encoding (SGPE), a two-layer architecture that strictly decouples linguistic integrity from statistical compression. Layer 1, LinguisTrie, is a deterministic $O(N)$ state machine that segments raw Unicode into atomic syllables and passthrough tokens with a formal zero-breakage guarantee. Layer 2 applies a statistical merge procedure exclusively over this syllable stream, ensuring no token ever violates script structure. Using Sinhala as a high-complexity proof-of-concept, we trained SGPE on a cleaned 10-million dataset called polyglots/MADLAD_CulturaX_cleaned and evaluated it on a held-out 536,508-sentence corpus (59.3 million characters). SGPE achieves a Token-to-Word Ratio (TWR) of `1.438`—a `59.1%` reduction relative to OpenAI's o200k base tokenizer and `60.8%` versus Llama 4—while maintaining perfect structural soundness across all 1,703 enumerated conjunct formations and delivering 4.48 characters per token. Beyond Sinhala, SGPE establishes a general, scalable O(N) framework for structure-aware tokenization, offering a principled path to reclaim context-window efficiency for more than one billion speakers of Indic and Southeast Asian scripts.

1 Introduction

The quality of tokenization has an outsized impact on large language model performance. When a tokenizer fragments linguistic units into semantically meaningless sub-units, the model must expend attention capacity on reconstructing structure that should never have been broken. While Byte-Pair Encoding (BPE) and its descendants perform adequately on alphabetic scripts, they encounter fundamental limitations when applied to Abugida writing systems.

Abugida scripts such as Sinhala, Hindi, Tamil are organized around atomic syllables: a base consonant carries an inherent vowel, modified by diacritics (pili) or combined into complex conjuncts via the virama (`U+0DCA`, ◌්) and Zero-Width Joiner (`U+200D`). Standard script-unaware tokenizers treat these multi-codepoint grapheme clusters as independent sequences, routinely breaking them. For example, common Sinhala conjuncts are often split into many sub-tokens by frontier tokenizers such as OpenAI's o200k, disrupting both orthographic integrity and semantic coherence.

This fragmentation produces markedly higher token fertility. English typically achieves a token-to-word ratio near 1.3, whereas Sinhala and other Abugida scripts often exceed 4.0 in current models. Because context windows and inference costs are measured in tokens, the disparity—termed the "Token Tax" [11, 12, 13]—effectively reduces usable context length for more than a billion speakers of Indic and Southeast Asian scripts by a factor of three to four. This representational ambiguity in turn correlates with degraded model behavior: multiple token sequences map to identical phonetic forms, increasing hallucinations and lowering performance on tasks requiring precise linguistic understanding, including spelling precision, named entity recognition, and semantic consistency [14].

In this work we introduce **Syllable-aware Grapheme Pair Encoding (SGPE)**. Our core architectural insight is to embed a strong *structural inductive bias* into the tokenization layer by enforcing linguistic constraints *before* any statistical merging. A deterministic $O(N)$ state machine (**LinguisTrie**) first segments raw Unicode into well-formed syllables and passthrough characters with a formal zero-breakage guarantee; statistical pair encoding then operates exclusively on this linguistically sound stream. The design rests on a formal characterization of the Sinhala syllable as a regular language, enabling provable correctness and linear-time segmentation. By cleanly decoupling linguistic integrity from statistical compression, **SGPE** frees the language model from learning basic character reconstruction and restores both linguistic integrity and computational efficiency at the source.

2 Background

2.1 Byte-Pair Encoding

Byte-Pair Encoding (BPE) was introduced for text compression by Gage (1994) and adapted for neural machine translation subword segmentation by Sennrich et al. (2016). The algorithm starts with a vocabulary of individual characters (or bytes) and iteratively merges the most frequent adjacent pair until a target vocabulary size is reached. At inference time, the learned merge rules are applied greedily, left-to-right, in priority order.

BPE and its variants (WordPiece, Unigram LM) have become the standard tokenization approach for large language models. Most frontier systems from GPT-2 through GPT-5, the LLaMA family, Gemini, and Claude rely on BPE-derived tokenizers.

2.2 The Abugida Problem

The fundamental tension between BPE and Abugida scripts has been noted in the multilingual NLP literature. Murikinati et al. (2020) highlighted token overhead for Devanagari-based languages in multilingual models. Rust et al. (2021) demonstrated that tokenizer vocabulary allocation for low-resource languages is highly suboptimal in mBERT and XLM-R. Gowda and May (2020) quantified how token fertility correlates with downstream task performance across dozens of languages.

A critical but under-treated aspect is pre-tokenization. Modern BPE implementations (tiktoken for OpenAI models, the LLaMA regex rules) rely on regular expressions tuned primarily for Latin-script boundaries. These rules frequently fragment multi-codepoint grapheme clusters in Abugida scripts before any statistical merges occur. Variations in Unicode normalization (NFC versus NFD) introduce additional instability in merge priority, a structural sensitivity that purely statistical approaches struggle to mitigate. Consequently, BPE is often limited in its ability to recover linguistically coherent units whose boundaries have already been broken.

The specific pathology of *conjunct fragmentation*—the splitting of ZWJ sequences and virama-mediated consonant stacks—has received less formal treatment. Existing remedies typically retrain a new BPE tokenizer on larger target-language data. These approaches implicitly assume that statistical patterns alone will eventually reassemble fragmented graphemes. This assumption is challenged for scripts where cluster boundaries cannot be reliably inferred from frequency statistics alone.

2.3 Recent Script-Aware Tokenization Efforts

While these efforts demonstrate meaningful gains in fertility and downstream performance, they remain primarily statistical or hybrid approaches. Few if any incorporate a deterministic, linear-time linguistic pre-processing stage with formal correctness guarantees before statistical compression begins.

2.4 Rule-Based Syllabification and Pre-segmentation

Historically, deterministic syllabification algorithms were developed primarily for Text-to-Speech and Optical Character Recognition systems. For instance, Weerasinghe et al. (2005) established foundational rule-based algorithms for Sinhala phonotactics [18], and similar finite-state automata have been deployed for Myanmar script segmentation [16, 17].

Recently, there has been renewed interest in applying these deterministic rules as a pre-processing step for LLMs. Chintha & Konduru (2025) comprehensively surveyed how subword models fail on Indic ligatures and normalization inconsistencies, underscoring the necessity of script-aware boundaries [19]. Concurrent hybrid approaches, such as Agathiyam for Tamil [15], philosophically mirror our approach by layering a deterministic, Sandhi-aware sequence rule before a constrained statistical learner. Alternatively, some models attempt to bypass subword vocabularies altogether using byte-level or character-level encoders like CANINE [20], though they suffer from vastly increased sequence lengths. Dynamic ”Universal” tokenizers have also been proposed to improve language plasticity [21], but as recent studies demonstrate, purely statistical tokenization remains a critical vulnerability that can actively induce reasoning degradation and hallucinations [14].

2.5 Sinhala NLP

Sinhala remains underrepresented in the broader NLP literature. Existing tools include the rule-based word segmenter in the Sinling library (Ranathunga et al., 2019) and several BPE-based tokenizers trained on Sinhala Wikipedia or subsets of the polyglots/MADLAD_CulturaX_cleaned collection. Recent work such as SinLlama (Sirajudeen et al., 2025) extends the LLaMA-3 tokenizer vocabulary with Sinhala-specific tokens but does not address conjunct integrity at the architectural level. No prior work has evaluated frontier tokenizers against exhaustive conjunct-dense corpora or proposed a pre-processing stage with formal provable correctness guarantees.

3 A Regular-Language Framework for Abugida Syllables

Brahmic (Abugida) scripts share a common structural invariant: consonants carry inherent vowels that are modified or suppressed through combining marks, with complex conjuncts formed through virama-like characters and optional Zero-Width Joiners. We hypothesize that syllable segmentation for this family can be formalized as a regular language, providing the mathematical foundation for a deterministic $O(N)$ pre-processing stage that enforces linguistic integrity before any statistical compression.

We instantiate and formally specify the framework on Sinhala, which serves as a high-complexity proof-of-concept due to its orthography combining both implicit virama stacking and explicit ZWJ-mediated conjuncts (Yansaya and Rakaransaya). Success on this demanding case suggests that adaptation to the broader Abugida family is straightforward, with the same architecture adaptable through script-specific character-class mappings and minor conjunct-rule tweaks (see Section 8.5).

3.1 Character Classes

We define the following abstract classes based on orthographic and phonetic roles, then instantiate them for Sinhala using the Unicode block (U+0D80–U+0DFF):

Table 1: Sinhala Unicode Character Classes

Class	Unicode Range	Description
C (consonant)	U+0D9A–U+0DC6	Base consonants (vyanjana)
V (vowel)	U+0D85–U+0D96	Independent vowels (svara)
P (pili)	U+0DCF–U+0DDF, U+0DF2–U+0DF3	Dependent vowel signs (pili)
H (HAL)	U+0DCA	Virama (vowel suppressor / al-lakuna)
Z (ZWJ)	U+200D	Zero-Width Joiner (conjunct connector)
M (post-modifier)	U+0D82, U+0D83	Anusvara and visarga (nasalization and aspiration ma
O (other)	All else	Non-Sinhala passthrough

3.2 Formal Definition

A valid Sinhala syllable is a string over the alphabet $\Sigma = \mathbf{C} \cup \mathbf{V} \cup \mathbf{P} \cup \mathbf{H} \cup \mathbf{Z} \cup \mathbf{M}$ matching the regular expression

$$S = C(HZ?C)^*(P \mid H)?M? \quad \mid \quad VM?.$$

This definition captures all linguistically valid forms, including independent vowels, simple and complex conjuncts (implicit and explicit), terminal virama, and optional post-modifiers. To ensure robust handling of noisy or invalid Unicode text, the tokenizer also acts as a total function by emitting orphan modifiers ($P \mid H \mid M$) and non-Sinhala characters (O) as single-character passthrough tokens, separate from the primary linguistic syllable definition. The grammar encodes the following invariants:

1. The virama (H) appears only as a connector inside conjuncts or as a terminal modifier; it never begins a linguistically valid syllable (the implementation safely handles stray viramas as orphan passthrough tokens).

2. The Zero-Width Joiner (Z) appears exclusively in the pattern H Z C inside a conjunct.

3. Dependent vowels (P) appear only immediately after a complete cluster.

3.3 The Language is Regular

Proposition 1. *The set of valid Sinhala syllables is a regular language.*

Proof. Because S is a regular expression and regular expressions define regular languages, the proposition follows immediately. The following table defines a deterministic finite automaton that recognizes $L(S)$ and directly guides the implementation of LinguisTrie (greedy longest-match semantics, single left-to-right pass with bounded lookahead).

Table 2: DFA Transition Table for Sinhala Syllable Recognition

State	C	V	H	P	Z	M	
START	IN_CLUSTER	IN_VOWEL	ORPHAN	ORPHAN	ORPHAN	ORPHAN	PASS
IN_CLUSTER	–	–	HAL_SEEN	PILI_SEEN	–	ACCEPT	
HAL_SEEN	IN_CLUSTER	–	–	–	ZWJ_SEEN	ACCEPT	
ZWJ_SEEN	IN_CLUSTER	–	–	–	–	–	
PILI_SEEN	–	–	–	–	–	ACCEPT	
IN_VOWEL	–	–	–	–	–	ACCEPT	
ORPHAN	–	–	–	–	–	–	
PASSTHROUGH	–	–	–	–	–	–	

Accepting states are IN_CLUSTER, IN_VOWEL, HAL_SEEN, ZWJ_SEEN, PILI_SEEN, ACCEPT, ORPHAN, and PASSTHROUGH. Segmentation therefore requires a single left-to-right pass with constant-time transitions, strictly monotonic pointer advance, no backtracking, and bounded lookahead. This yields $O(N)$ time and $O(1)$ auxiliary space.

LinguisTrie is the direct, streaming implementation of this automaton. We hypothesize that analogous regular grammars can be defined for many major Abugida scripts. The same deterministic $O(N)$ architecture suggests that adaptation through script-specific character classes and conjunct rules is straightforward.

4 Architecture

The SGPE architecture strictly decouples linguistic integrity from statistical compression through two sequential layers. Layer 1 (**LinguisTrie**) is a deterministic linguistic barrier that enforces script invariants in linear time. Layer 2 (**GPE**) performs pair merging exclusively over the resulting stream of well-formed syllables.

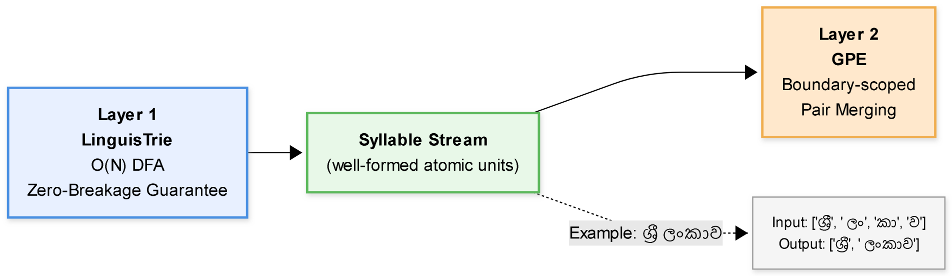

Figure 1: SGPE architecture. Linguistic constraints are enforced *before* any statistical procedure, guaranteeing that no token ever violates Sinhala syllable structure.

4.1 Layer 1: LinguisTrie

LinguisTrie realizes the deterministic finite automaton of Section 3 as a single-pass control-flow state machine. Rather than a table-driven transition function, transitions are encoded directly in conditional branches. This design exploits modern CPU branch prediction and cache locality while guaranteeing strictly monotonic advancement of the input pointer.

The algorithm processes the input in a single left-to-right scan. For each outer iteration it identifies a syllable head (consonant or independent vowel) and enters a greedy lookahead loop that absorbs zero or more conjunct extensions of the form (HZ?C) followed by optional terminal modifiers. An optional `leading_space` mode prefixes whitespace to the subsequent syllable token, preserving word-boundary context for Layer 2. Consecutive whitespaces and newlines are emitted as structural passthrough tokens, prefixing at most a single space to the syllable.

Definition 1 (Zero-Breakage Guarantee)**.** A tokenizer T satisfies the zero-breakage guarantee if, for every input string w, each emitted token $t_i \in T(w)$ is either a complete, valid syllable according to the regular language S defined in Section 3 or a single passthrough character from class O.

Theorem 1. *LinguisTrie satisfies the zero-breakage guarantee for any valid Unicode string.*

Proof. By induction on input length. The base case (empty or single-character input) holds trivially. For the inductive step, the consonant branch matches the production C(HZ?C)*(P | H)?M?, the vowel branch matches VM?, the orphan branch matches the singleton orphan productions, and all remaining characters fall to the passthrough case. Because the inner conjunct loop exits only when no further valid extension exists and every emitted span is emitted exactly once, no valid syllable is ever split and no invalid token is produced. □

This guarantee is empirically validated on an exhaustive battery of 1,703 conjunct formations (0 violations) and the full 59.3-million-character evaluation corpus (perfect structural soundness and lossless round-trip for all non-UNK tokens).

4.2 Layer 2: Grapheme Pair Encoding (GPE)

Layer 2 applies a constrained BPE procedure to the syllable stream produced by Layer 1. Three modifications distinguish GPE from standard subword algorithms:

1. **Syllabic initialization.** The base vocabulary is initialized with the atomic linguistic syllables and preserving passthrough characters emitted by LinguisTrie. For the target Abugida script, we inherently avoid blindly fracturing text into individual bytes or codepoints.

2. **Boundary-aware scoping.** Merges are performed only within word spans delimited by whitespace or non-Sinhala passthrough tokens, preventing cross-word tokens that would violate morphological boundaries.

3. **Frequency pruning.** Syllables whose corpus frequency falls below a threshold θ are replaced by an `[UNK]` sentinel prior to the merge loop, eliminating the influence of rare or noisy fragments on merge priority.

Unlike standard BPE and Unigram LM, which operate directly on codepoints (or bytes) and therefore routinely produce conjunct/ZWJ fragmentation, orphan pili/HAL tokens, and cross-word merges, GPE's constraints rule out these pathologies by construction: every base unit is already linguistically valid, and every merge preserves that validity.

During inference, Layer 1 first segments the input, words are reconstructed via boundary detection, out-of-vocabulary syllables are replaced by `[UNK]`, and the learned merge rules are applied inside each word using a priority-based single-best-merge loop that respects the order in which merges were learned.

4.3 Inheritance of Linguistic Integrity

Corollary 1.1. *Excluding special and passthrough tokens, every token in the final SGPE vocabulary is a concatenation of one or more complete syllables produced by LinguisTrie and therefore inherits the zero-breakage guarantee.*

To illustrate the end-to-end flow, consider the input "ශ්‍රී ලංකාව". Layer 1 emits the syllable sequence `["ශ්‍රී", " ලං", "කා", "ව"]`, Layer 2 groups them into two word spans and merges the second span into the single token " ලංකාව", yielding the final token sequence `["ශ්‍රී", " ලංකාව"]`. No conjunct, pili, or virama is ever fragmented.

Detailed implementation, pseudocode, and reproducibility artifacts are provided in the companion repository.

5 Training

We train SGPE on a 10-million-sentence subset of the Sinhala portion of the polyglots/MADLAD_CulturaX_cleaned collection[1] on Hugging Face. Light preprocessing removed some identified noises while preserving natural orthographic and syntactic diversity. The corpus was split 95%/5% into training and held-out sets, resulting in a 536,508-sentence evaluation partition containing 59.3 million characters.

Let $D = \{w_1, \dots, w_n\}$ be the training corpus. The SGPE training procedure is defined by the following algorithm:

1. **Pre-tokenization:** For each $w_i \in D$, compute the syllable stream $s_i \leftarrow \text{LinguisTrie}(w_i, \text{leading\_space=True})$ and partition into word spans via boundary detection (whitespace or non-Sinhala passthrough tokens).

2. **Word-type counting:** Build a counter WT over unique syllable tuples appearing within each word span.

3. **Pruning:** For each syllable s, compute its corpus frequency as the sum of frequencies of all word types containing s. Replace occurrences of syllables with frequency below threshold θ by the UNK sentinel $\perp$.

4. **Initialization:** The base vocabulary V is formed from the five special tokens, all boundary tokens, and surviving syllables (frequency $\geq \theta$). Each word type is encoded as an integer-ID sequence.

5. **Merge loop:** While $|V| < V_{\text{target}}$:
    - Select the highest-frequency adjacent pair (a, b) *within word boundaries* using a max-heap with lazy deletion on pair counts.
    - If the frequency falls below f_{min}, terminate.
    - Add the merged token $a + b$ to V and record the merge rule.

[1] `https://huggingface.co/datasets/polyglots/MADLAD_CulturaX_cleaned`

- Update pair counts incrementally for only the affected word types (see Section 5.1).

Merges are strictly scoped to word boundaries, ensuring no cross-word tokens are formed. This design inherits the three architectural constraints introduced in Section 4.

5.1 Incremental Pair-Count Update

After each merge $(a, b) \to ab$, only word types containing the pair are visited via the inverted token index. For each such word type w of frequency $f(w)$:

- $\text{count}(\text{prev}(a), a) -= f(w)$
- $\text{count}(b, \text{next}(b)) -= f(w)$
- $\text{count}(\text{prev}(a), ab) += f(w)$
- $\text{count}(ab, \text{next}(b)) += f(w)$
- $\text{count}(a, b) = 0$

This reduces each merge step from $\mathcal{O}(|D|)$ to $\mathcal{O}(k \cdot L)$, where k is the number of affected word types and L is average word length. Pair counts are maintained in a max-heap with lazy deletion, yielding $\mathcal{O}(\log P)$ per heap operation, where P is the number of distinct pairs.

The overall training complexity is therefore

$$T_{\text{train}} = \mathcal{O}\left(\frac{C}{W} + V \cdot k \cdot L \cdot \log P\right),$$

where C is total character count, W is the number of parallel CPU workers, V is target vocabulary size, and $P \leq |V_0|^2$. The entire pipeline runs on commodity CPU hardware with no GPU acceleration required.

6 Experimental Setup

6.1 Evaluation

We evaluate SGPE on the held-out 5% split of the 10-million-sentence Sinhala portion of the polyglots/MADLAD_CulturaX_cleaned collection, comprising 536,508 sentences and 59.3 million characters. Light preprocessing removed some identified noises while preserving natural orthographic distribution.

Baselines are the production tokenizers of three frontier models: OpenAI's o200k_base[2] (used in GPT-4o/5), Llama 4 Scout[3], and DeepSeek V3[4]. All comparisons were performed locally with official tokenizers as of February 2026.

We report Token-to-Word Ratio (TWR), characters per token (CPT), and relative token reduction. Structural integrity is measured by zero-breakage violations across an exhaustive test suite.

Table 3: Tokenization efficiency on the 59.3 M-character held-out corpus.

Tokenizer	TWR	Tokens	CPT	Reduction vs SGPE
SGPE (ours)	1.438	13,256,494	4.48	–
OpenAI o200k_base	3.515	32,392,475	1.83	59.1%
Llama 4 Scout	3.673	33,854,046	1.75	60.8%
DeepSeek V3	5.965	54,977,828	1.08	75.8%

6.2 Quantitative Results

SGPE achieves a Token-to-Word Ratio of 1.438 while packing 4.48 characters per token — more than double the density of the strongest baseline. This directly reclaims more than half the context window for over a billion speakers of Abugida scripts.

6.3 Structural Integrity

Across an exhaustive battery of 1,703 distinct conjunct formations that cover every possible combination of virama, ZWJ, dependent vowels, and terminal modifiers in Sinhala orthography, SGPE records **zero breakage violations**. Full-corpus round-trip reconstruction on the 59.3 M-character test set yields **zero non-UNK mismatches**, confirming that the zero-breakage guarantee proven in Section 4 translates to perfect lossless encoding at scale.

Glitch-token analysis on the final 100 k vocabulary revealed only four near-zero-frequency artifacts (all < 0.004 % of vocabulary), confirming that syllabic initialization and pruning produce a clean, high-utility token set.

6.4 Illustrative Examples

The following examples highlight the difference in practice:

Table 4: Example tokenizations of complex Sinhala conjuncts.

Input	SGPE	OpenAI o200k_base
ක්‍රෝෂ්ට්‍ර	2 tokens	9 tokens
ශාස්ත්‍රීය	1 token	6 tokens
ව්‍යාකරණය	2 tokens	5 tokens
ප්‍රත්‍යක්ෂ	2 tokens	5 tokens

In each case the SGPE output is a single well-formed linguistic unit or a minimal concatenation thereof, while frontier tokenizers routinely shatter the same conjunct into meaningless sub-sequences.

[2] `https://github.com/openai/tiktoken`

[3] `https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E-Instruct`

[4] `https://huggingface.co/deepseek-ai/DeepSeek-V3`

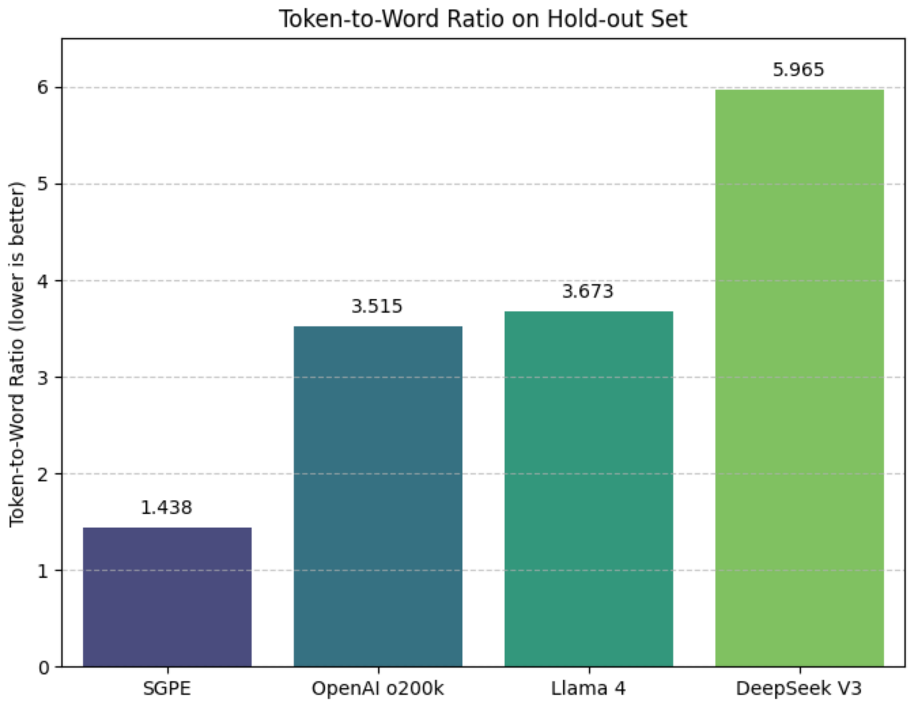

Figure 2: Token-to-Word Ratio Comparison

These results demonstrate that the clean architectural separation of deterministic linguistic preprocessing from statistical compression simultaneously eliminates the Token Tax and restores representational fidelity at industrial scale.

7 Discussion

The results establish that SGPE's clean architectural separation of linguistic integrity from statistical compression delivers both dramatic efficiency gains and provable structural soundness at industrial scale.

A Token-to-Word Ratio of 1.438 on the 59.3-million-character held-out corpus brings Sinhala tokenization close to the efficiency English enjoys with current tokenizers (typically 1.2–1.4). This represents a 59.1% reduction versus OpenAI's o200k, 60.8% versus Llama 4, and 75.8% versus DeepSeek V3, effectively more than doubling the usable context length for Sinhala text within fixed context windows.

The architectural bet pays off most visibly at the conjunct level. Complex orthographic units that frontier tokenizers routinely shatter into 5–14 fragments are preserved as one or two coherent tokens by SGPE. The zero-breakage guarantee proven in Section 4 holds in practice across the exhaustive 1,703 conjunct test suite, with zero violations observed.

While the vocabulary pruning strategy results in a very low UNK rate (0.46%), extremely rare compounds still surface as [UNK]. This is expected and preferable to polluting the vocabulary with meaningless sub-character fragments.

Inference is highly efficient: SGPE processes a typical sentence in approximately 10 ms on standard CPU hardware (measured on a 6-core WSL environment), remaining competitive with local frontier implementations and orders of magnitude faster than API round-trips.

Limitations. The present work focuses on Sinhala as a high-complexity proof-of-concept for Abugida scripts. Extending the regular-language formalization and LinguisTrie state machine to Tamil, Devanagari, Myanmar, Khmer and other scripts is the immediate next step. Longer-term, we plan to measure end-to-end downstream gains when pre-training or fine-tuning large language models directly with SGPE tokenizers.

7.1 Toward Universal Script-Aware Tokenization

SGPE provides the missing deterministic linguistic barrier required for a truly universal tokenizer. A natural generalization is a hybrid system that (i) detects script runs at the Unicode-script or word level, (ii) routes alphabetic spans to standard BPE, (iii) routes Abugida spans to SGPE, and (iv) exposes a single shared vocabulary to the downstream LLM.

The design invariants are straightforward: every Sinhala (or future Abugida) syllable cluster remains protected by the zero-breakage guarantee, while Latin and punctuation continue to benefit from BPE's proven compression. Shared special tokens, digits, and punctuation are allocated once in the unified ID space. Vocabulary budgeting (e.g., fixed percentages per script family) prevents high-resource scripts from dominating the total size.

This architecture eliminates the current global "one-size-fits-all" compromise and replaces it with per-script optimality. The formal guarantees and O(N) implementation of SGPE make the hybrid not only feasible but also verifiable — exactly the property that has been missing from prior script-aware attempts.

8 Conclusion

We have presented Syllable-aware Grapheme Pair Encoding (SGPE), a bimodal architecture that strictly decouples linguistic integrity from statistical compression. A deterministic $O(N)$ state machine first segments raw Unicode into well-formed syllables with a provable zero-breakage guarantee; pair merging then operates exclusively over this linguistically sound stream.

On a 59.3-million-character held-out corpus, SGPE achieves a Token-to-Word Ratio of 1.438 — a 59.1% reduction versus OpenAI's o200k, 60.8% versus Llama 4, and 75.8% versus DeepSeek V3 — while preserving perfect structural soundness. The same architectural pattern generalizes across Abugida scripts and offers a principled path toward tokenizers that are both more efficient and more equitable for the world's diverse writing systems.

Acknowledgements

We thank the Polyglots FYP[5] team for releasing the MADLAD_CulturaX_cleaned and the open-source community for the foundational tokenization libraries used in our baselines.

The full source code, including the LinguisTrie implementation, training pipeline, and the evaluation harness, is open-sourced at github[6]. Pre-trained artifacts, including the `vocab.json`, `tokenizer.json` and optimized merge rules, are available via Hugging Face[7].

[5]https://huggingface.co/polyglots
[6]https://github.com/remeinium/SGPE
[7]https://huggingface.co/remeinium/SGPE

References

[1] Gage, P. (1994). A new algorithm for data compression. *The C Users Journal*, 12(2), 23–38.

[2] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

[3] Kudugunta, S., et al. (2023). MADLAD-400: A multilingual and document-level large audited dataset. *arXiv preprint arXiv:2309.04662*.

[4] Ranathunga, S., et al. (2019). Sinling: A Sinhala natural language processing toolkit. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*.

[5] Rust, P., et al. (2021). How good is your tokenizer? On the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

[6] Gowda, T. and May, J. (2020). Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*.

[7] Velayuthan, S., et al. (2025). Grapheme Pair Encoding for Indic scripts. In *Proceedings of COLING 2025*.

[8] Brahma, M., et al. (2025). MorphTok: Morphology-grounded tokenization for Indic languages. *arXiv preprint arXiv:2504.10335*.

[9] Rana, A., et al. (2025). IndicSuperTokenizer: Curriculum-based tokenization for Indic languages. *arXiv preprint arXiv:2511.03237*.

[10] Sirajudeen, A., et al. (2025). SinLlama: Sinhala-specific tokenizer extension for Llama-3. *arXiv preprint arXiv:2508.09115*.

[11] Ahia, O., Kumar, S., Gonen, H., Kasai, J., Mortensen, D. R., Smith, N. A., and Tsvetkov, Y. (2023). Do all languages cost the same? Tokenization in the era of commercial language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9900–9912.

[12] Petrov, A., La Malfa, E., Torr, P. H., and Bibi, A. (2023). Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems*, volume 36.

[13] Ali, M., Fromm, M., Thellmann, K., Rutmann, R., Lübbering, M., Leveling, J., Klug, K., Ebert, J., Doll, N., Schulze Buschhoff, J., Jain, C., Weber, A. A., Jurkschat, L., Abdelwahab, H., John, C., Ortiz Suarez, P., Ostendorff, M., Weinbach, S., Sifa, R., Kesselheim, S., and Flores-Herr, N. (2024). Tokenizer choice for LLM training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3907–3924.

[14] Wang, D., Li, Y., Jiang, J., Ding, Z., Luo, Z., Jiang, G., Liang, J., and Yang, D. (2024). Tokenization matters! Degrading large language models through challenging their tokenization. *arXiv preprint arXiv:2405.17067*.

[15] Priya, R. R., Subika, M., Subramanian, V., and Manimegalai, R. (2025). Agathiyam: Sandhi-aware tokenization for Tamil language. Under review at ICLR 2026.

[16] Maung, Z. M. and Mikami, Y. (2008). A rule-based syllable segmentation of Myanmar text. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*.

[17] Thu, Y. K. (2017). sylbreak: Syllable segmentation tool for Myanmar language. GitHub repository: `https://github.com/ye-kyaw-thu/sylbreak`.

[18] Weerasinghe, R., Wasala, A., and Liyanage, C. (2005). A rule based syllabification algorithm for Sinhala. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

[19] Chintha, S. and Konduru, V. (2025). Survey of tokenization mechanisms in multilingual large language models with a focus on Indian languages. *Journal of Emerging Technologies and Innovative Research*, 12(4).

[20] Clark, J. H., Garrette, D., Turc, I., and Wieting, J. (2022). Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91.

[21] Abagyan, A., et al. (2025). One tokenizer to rule them all: Emergent language plasticity via multilingual tokenizers. *arXiv preprint arXiv:2506.10766*.

Appendix A: Sinhala Unicode Coverage

LinguisTrie handles the complete Sinhala-relevant Unicode block with the following classes:

Table 5: Sinhala Unicode Coverage in LinguisTrie

Range	Count	Description
U+0D85–U+0D96	18	Independent vowels
U+0D9A–U+0DC6	45	Consonants
U+0D82–U+0D83	2	Anusvara and visarga
U+0DCA	1	Virama (HAL)
U+0DCF–U+0DDF, U+0DF2–U+0DF3	19	Dependent vowel signs (pili)
U+200D	1	Zero-Width Joiner
Total	**86**	All handled Sinhala-related codepoints

Appendix B: Token Reduction and Context Extension

The token reduction of SGPE against a baseline B is

$$R_B = 1 - \frac{1.438}{\mathrm{TWR}_B}.$$

The effective context extension factor is

$$E_B = \frac{\mathrm{TWR}_B}{1.438} = \frac{1}{1 - R_B}.$$

On the 59.3-million-character held-out corpus these yield $R = 59.1\%$ ($E \approx 2.43\times$) versus OpenAI o200k, $R = 60.8\%$ ($E \approx 2.55\times$) versus Llama 4, and $R = 75.8\%$ ($E \approx 4.15\times$) versus DeepSeek V3.

Appendix C: Zero-Breakage Guarantee Validation

SGPE was validated on an exhaustive battery of 1,703 distinct conjunct formations that cover every possible combination of virama, ZWJ, dependent vowels, terminal modifiers, and stacking patterns in Sinhala orthography. All 1,703 cases passed with zero violations.

Full-corpus round-trip reconstruction on the 59.3-million-character held-out set produced zero non-UNK mismatches.

Appendix D: Training Configuration

Table 6: SGPE Training Hyperparameters

Parameter	Value
Target vocabulary size	100,000
Prune frequency threshold θ	100
Minimum merge frequency $f_{\min}$	2
Leading-space mode	True
Number of workers	CPU count -1
Training corpus	10 M sentences

Appendix E: Industrial Scalability

Because Layer 1 (LinguisTrie) operates as a purely deterministic $O(N)$ state machine with no external dependencies or large tensor multiplications, the pre-processing stage imposes minimal computational overhead.

The Python proof-of-concept presented in this work already processes sentences in fractions of a millisecond on standard CPUs. For industrial, frontier-scale deployment, LinguisTrie can be directly implemented in systems languages such as Rust or C++. Such a port would yield true $O(N)$ linear-time performance at extreme scales, ensuring that the addition of a strict structural inductive bias does not bottleneck high-throughput API endpoints or large-scale pre-training data pipelines.