Multilingual Tokenization through the Lens of Indian Languages: Challenges and Insights

Anonymous ACL submission

Abstract

Tokenization plays a pivotal role in NLP and is fundamental to training language models. However, existing tokenizers are often skewed towards high-resource languages, limiting their effectiveness for linguistically diverse and morphologically rich languages such as those in the Indian subcontinent. This paper presents a comprehensive evaluation of tokenization strategies across 17 Indic languages with 11 diverse writing scripts from 2 language families . We quantify the trade-offs between bottom-up and top-down tokenizer algorithms – BPE (Sennrich et al., 2016) and Unigram LM (Kudo, 2018), the effect of language-specific script normalization, the effect on morphological alignment, the effects of vocabulary sizes, and compare strategies of multilingual vocabulary construction, such as joint and cluster-based methodologies. We show that the cluster-based grouping language performs well compared to the Joint method, though it is sensitive to cluster formation.

1 Introduction

Tokenization is the process of segmenting raw text into smaller units/tokens (words, subwords, characters, etc.), which helps in efficient processing by the computational models, particularly in Large Language Models (LLMs). Tokenizer step forms a fundamental step in any Natural Language Processing (NLP) task, and hence the quality of the tokenizer impacts the model accuracy and training speed. Poor tokenization can lead to an increase in sequence lengths, fragment meaningful units, and weaken model's ability to capture linguistic structure and semantics. Tokenization further influences how well a model understands the linguistic structure and semantics of the input and how well it handles the vocabulary coverage. Most widely used tokenizers are primarily designed for English, due to the large-scale data availability and extensive research conducted on the language. These tokenizers are optimized for the linguistic structure, morphology, spacing, and limited inflective properties of English and other related languages. There's a widespread tendency to reuse the same tokenization configurations across Indic languages, despite their distinct characteristics. Such an English-centric design for a tokenizer poses a challenge when applied to various other languages, especially those with rich morphology, agglutinative properties, and complex scripts. This challenge becomes more prominent in multilingual settings. Unlike monolingual tokenizers, a multilingual tokenizer must allocate a single shared vocabulary across languages, which may vary across script, morphology, and resource availability.

Given the significance of a good quality tokenizer, it is important to perform a detailed study of the working and the influence of various types of tokenizers on language models and other downstream tasks. Zouhar et al. (2023) and Ali et al. (2024) conduct an extensive study to understand the influence of tokenization with the help of various intrinsic and extrinsic evaluation metrics. While previous works like Rust et al. (2021); Limisiewicz et al. (2023) have addressed tokenizer evaluation in multilingual contexts, Indic languages have received relatively lesser attention. Indic languages in particular are interesting for tokenization studies due to their rich morphological structure, agglutinative and inflectional properties, and diverse scripts. Unlike English and other high-resource languages, Indic languages exhibit a complex word formation process, extensive inflectional paradigms, and orthographic variations. Furthermore, the disparity in resource availability across languages may lead to a disproportionate favoring of high-resource languages. To systematically study these challenges, we formulate the following questions and subsequently attempt to answer them in the paper.

(RQ1) What is the impact of script and ortho-

graphic normalization on tokenization quality for Indic languages?

(RQ2) How do different subword algorithms vary in their ability to preserve morphological alignment in Indic languages?

(RQ3) Which multilingual vocabulary creation strategies, such as joint or cluster-based methodologies, yield the most effective tokenization, and what is the optimal way to group languages?

In this work, we present various methods for tokenizer training and vocabulary building, with a focus on multilingual Indian languages, and perform intrinsic evaluations to understand the tokenizer's ability to capture the aforementioned characteristics of these languages. We particularly focus on the most widely used tokenizers *viz.,* Byte Pair Encoding (BPE) (Sennrich et al., 2016) and Unigram Language Model (ULM) (Kudo, 2018)[1], with vocabulary size ranging from 32K to 256K. Our contributions are summarized as follows:

1. Investigating the performance and impact of multilingual tokenization for 17 Indic languages from 2 language families *viz.,* Indo-European and Dravidian, with 11 different writing systems.

2. Analyse the impact of language-specific script normalization on the tokenization quality, vocabulary size, and methodologies to build a multilingual tokenizer. Additionally, we performed several analyses on two variants of BPE – PickyBPE (Chizhov et al., 2024) and OBPE (Patil et al., 2022).

2 Related Works

Subword tokenization. Tokenization is a fundamental task in Natural Language Processing, with subword tokenization algorithms such as Byte Pair Encoding (BPE) (Sennrich et al., 2016) and Unigram Language Model (ULM) (Kudo, 2018) remaining the most widely adopted algorithms for NLP tasks. Additionally, recent work have introduced several new tokenization algorithm, which are mostly variants of BPE and ULM such as Overlap BPE (Patil et al., 2022), PickyBPE (Chizhov et al., 2024), Scaffold BPE (Lian et al., 2025), BPE Knockout (Bauwens and Delobelle, 2024), and Conditional Unigram LM (Vico and Libovický, 2025). Several studies have investigated the impact of tokenization, focusing on various aspects including the effects of tokenization on model capacity, generalisation, and fairness across languages (Limisiewicz et al., 2023; Schmidt et al., 2024). Works examining these aspects from the context of Indian languages so far are limited.

Multilingual tokenization. Multilingual tokenizers face an inherent trade-off between enabling cross-lingual transfer and maintaining optimal monolingual performance. This challenge was highlighted by Rust et al. (2021) demonstrating that replacing a shared multilingual tokenizer with a language-specific one can improve the downstream performance of a model. Subsequent research has focused on this challenge by improving on how shared vocabularies are allocated across languages. Chung et al. (2020) proposed a method to improve multilingual vocabulary allocation by clustering languages based on vocabulary overlap, while Limisiewicz et al. (2023) studied the impact of vocabulary overlap and allocation for 20 languages. More recently, Abagyan et al. (2025) found that training tokenizers with more languages than pretraining languages improves the adaptation capabilities to unseen languages. Building on these insights, our work investigates optimal tokenization strategies specific to morphologically rich and script diverse Indian languages, by conducting an empirical comparison of various strategies to determine their effectiveness specifically to Indian languages.

3 Subword Tokenization

Subword tokenization is a fundamental technique in modern NLP, particularly for LLMs. Recent work has highlighted the critical roles of tokenization in multilingual settings with implications for both model performance and token fairness (Petrov et al., 2023; Ali et al., 2024). This issue is especially pronounced for Indic languages, which cover large languages with diverse scripts, rich morphology, and limited representation in the pretraining corpus.

Most contemporary subword tokenization frameworks are based on Byte Pair Encoding (BPE) and Unigram Language Model (ULM) based approaches. Recently proposed tokenizers (Lian et al., 2025; Chizhov et al., 2024; Bauwens and Delobelle, 2024), whether specific to a language or multilingual, are basically the extensions or variants of these algorithms. Hence, we choose BPE and ULM for all the experiments in this study as they provide coverage with respect to the dominant approaches and provide a comprehensive basis for evaluating the effects of multilingual vocabulary construction and the varying vocabulary sizes.

[1] https://github.com/google/sentencepiece

3.1 Algorithms

In this section, we describe two widely used algorithms for tokenization.

Byte-pair encoding (BPE) (Sennrich et al., 2016): BPE is a bottom-up subword-segmentation approach that iteratively merges the most frequent bigrams to build a vocabulary till the required vocabulary size is reached. It begins with Unicode characters as the initial vocabulary and greedily performs the merges.

Unigram Language Model (ULM) (Kudo, 2018): In contrast to BPE, ULM follows a top-down approach. It begins with the superset of the target vocabulary and iteratively prunes it down to the desired vocabulary size. Unlike BPE's purely frequency-based approach, ULM leverages a probabilistic language model, which enables sampling of multiple possible segmentations. ULM uses the Expectation-Maximization (EM) algorithm to estimate the probabilities of subword units.

3.2 Methods of Combining Vocabulary

In this section, we discuss different methods of combining vocabularies of languages to build a shared vocabulary when working in a multilingual setting. Consider a set of languages $\mathcal{L} = \{\mathcal{L}_1, \mathcal{L}_2, \cdots, \mathcal{L}_n\}$ and corresponding monolingual corpus $\mathcal{D}_i$ for language $\mathcal{L}_i$. To obtain a multilingual tokenizer, we investigate two approaches *viz.* joint and cluster.

3.2.1 Joint Method for Multilingual Tokenizer

In this method, a multilingual corpus is created by concatenating data from all languages. A shared vocabulary is then obtained by training the tokenizer on this data. This method is straightforward and widely used (Sennrich et al., 2016; Ramesh et al., 2022; Doddapaneni et al., 2023). However, it may disproportionally favor high-resource languages during training, leading to under-representation of tokens from low-resource languages. We construct the multilingual corpus by $\mathcal{M} = \text{CONCAT}(\mathcal{D}_1, \ldots, \mathcal{D}_n)$ and derive the vocabulary $\mathcal{V}_M = \mathcal{T}(\mathcal{M}, \mathcal{K})$, where $\mathcal{T}$ denotes a vocabularization algorithm (e.g., BPE, ULM) and $\mathcal{K}$ is desired vocabulary size.

3.2.2 Cluster-based Method for Multilingual Tokenizer

In the cluster-based method (Chung et al., 2020), languages that are typologically or script-wise similar are grouped into clusters. We train ULM tokenizers on monolingual corpora for each language.

Algorithm 1 Cluster-based Multilingual Tokenizer

Require: **[Input]** Set of languages L, target vocab size V_{total}, number of clusters K, Monolingual corpora D^l for $l \in L$
Ensure: **[Output]** Multilingual Tokenizer $\mathcal{V}_M$
  for language $l \in L$ **do**
    $V^l \leftarrow$Train Unigram LM tokenizer on D^l
  end for
  $V^L \leftarrow \bigcup_{l \in L} V^l$
  for each language $l \in L$ **do**
    Initialize $v_l \in \{0,1\}^{|V^L|}$
    for each subword index i in V^L **do**
      if $V^L[i] \in V^l$ **then**
        $v_l[i] \leftarrow 1$
      end if
    end for
  end for
  Train K-means clustering on $\{v_l\}$
  Obtain clusters $C \leftarrow \{C_1, C_2, \ldots, C_K\}$
  for each cluster $c_i \in C$ **do**
    Concatenate corpora in cluster $D^{C_i} \leftarrow \bigcup_{l \in C_i} D^l$
    Compute proportion $p_i \leftarrow \dfrac{\left|\bigcup_{l \in C_i} V^l\right|}{\sum_{j=1}^{K} \left|\bigcup_{l \in C_j} V^l\right|}$
    Set vocabulary size $|V^{C_i}| \leftarrow p_i \times V_{\text{total}}$
    Train tokenizer on D^{C_i} with vocab size $|V^{C_i}|$
    Extract cluster vocabulary V^{C_i}
  end for
  $V^{multi} \leftarrow \bigcup_i V^{C_i}$
  $\mathcal{D}_{merge} \leftarrow \bigcup_{l \in L} D^l$
  for each token $s \in V^{multi}$ **do**
    Compute $prob(s) \leftarrow \dfrac{\text{count}(s \mid \mathcal{D}_{merge})}{\sum_{t \in V^{multi}} \text{count}(t \mid \mathcal{D}_{merge})}$
  end for
  $\mathcal{V}_M \leftarrow (V^{multi}, \{prob(s)\}_{s \in V^{multi}})$

Separate tokenizers are trained for each cluster, and the resulting vocabulary is then merged to get a final multilingual vocabulary. This approach reduces over-segmentation in low-resource languages by preserving vocabulary in each cluster. To find clusters, we follow the Chung et al. (2020) method. We first train monolingual tokenizers for 17 languages using the ULM and take the union of all the vocabularies U_v. We then create a language-specific vector of size $|U_v|$ by marking entries with 1 if the token is present in its vocabulary; otherwise, we mark it as 0. We then train a K-means clustering algorithm using the vector as input. We then train individual tokenizers for each cluster. Finally, we merge the tokenizers to get the final vocabulary. The detailed algorithm is presented by Algorithm 1. We identify clusters in two ways: automatically using K-means, and manually based on language family groupings.

4 Experimental Setup

4.1 Languages and Datasets

We consider 17 Indian languages from two diverse language families with 11 different scripts. Languages considered are Hindi (hin), Assamese (asm), Bengali (ben), Konkani (gom), Gujarati (guj), Kannada (kan), Maithili (mai), Malayalam (mal), Marathi (mar), Nepali (nep), Oriya (ori), Punjabi (pan), Sanskrit (san), Sindhi (snd), Tamil (tam), Urdu (urd), and Telugu (tel).

Dataset: We utilize the Sangraha corpus (Khan et al., 2024), which offers high-quality verified data. We sample 10% of the verified data and retain only the languages with more than 10k rows, resulting in a selection of 17 Indic languages. Further, we exclude sentences containing more than ten words written in Roman script, as these are likely code-mixed or non-standard. To ensure a balanced multilingual training corpus, we follow the sampling strategy of (Conneau and Lample, 2019), with a temperature parameter $\alpha = 0.3$. Detailed data sampling process and statistics are presented in Appendix A. We apply normalization on the sampled corpus using the IndicNLP library[2] (Kunchukuttan, 2020), which standardizes Unicode characters and diacritics across Indic scripts.

4.2 Evaluation

We assess tokenizer performance using both task-independent tokenizer properties (intrinsic evaluation) and downstream model performance (extrinsic evaluation). For intrinsic evaluation, we analyze the tokenizers on the Flores-200 dev set (Costa-Jussà et al., 2022) containing 997 sentences. For morphological alignment, we use the MorphScore dataset (Ali et al., 2024) for Gujarati and Tamil, and the dataset by Brahma et al. (2025) for Hindi and Marathi.

4.2.1 Intrinsic Evaluation Metrics

Following are the task-independent tokenizer properties used to analyse their performance.

Fertility (F) (Ali et al., 2024): It is the measure of the average number of tokens a space-separated word is split into. If every word is mapped to exactly one token, the fertility score $= 1$. If a word is broken into two or more subwords/tokens, the fertility score >1.

Character Per Token (CPT): (Limisiewicz et al., 2023): It is the average number of characters in a token, and gives an idea of how granular the tokenization is. A low CPT indicates higher granularity suggesting an aggressive split, while a higher CPT value indicates lower granularity.

The metrics described above are complementary to each other, and give a holistic view of the tokenizer behaviour with granularity-level insights. In addition to these metrics, we also use the following metrics to gain insights in terms of linguistic fairness and morphology preservation respectively.

Morphological Alignment: MorphScore (Arnett and Bergen, 2025) measures how well a tokenizer splits words in alignment with their actual morpheme boundaries. In this paper, we use a variant of morphscore, which assumes that a word may be tokenized into two or more segments, while also accounting for possible modifications at the segment boundaries arising from the underlying morphemes that compose the word. For each word, we calculate the percentage of correct segments in the tokenized output compared to the morpheme-based segments (ground truth).

$$MorphScore = \frac{\text{\# correct segments}}{\text{total \# segments (in ground-truth)}}$$

4.2.2 Extrinsic Evaluation

For extrinsic evaluation of the tokenizers, we compare the performance of a simple bidirectional LSTM-based model, trained to perform NER task using different tokenization methods and vocabulary sizes. Evaluation is performed for each of the Named Entity categories *viz.,* Location (LOC), Organization (ORG), and Person (PER). Metric used for evaluation is F1 score for individual categories and the overall Macro average F1 score.

5 Experiments and Results

We trained a total of 28 tokenizers for 17 Indian languages using existing algorithms: BPE and ULM. Additionally, we experimented on two different writing scripts – Native and ISO 15919 script[3]. Recent work by Ali et al. (2024) highlights that the implementation of BPE varies across tokenization libraries such as Huggingface and SentencePiece. Based on their findings[4], we train all tokenizers in our experiments using *SentencePiece* (Kudo, 2018)

[2] https://github.com/anoopkunchukuttan/indic_nlp_library

[3] https://en.wikipedia.org/wiki/ISO_15919

[4] Their findings indicate that SentencePiece generally yields better results than Huggingface implementation.

Algorithm	Vocab							
	32k		64k		128k		256k	
	NN	N	NN	N	NN	N	NN	N
BPE	2.190	**2.137**	1.926	**1.872**	1.717	**1.664**	1.568	**1.517**
ULM	2.148	**2.093**	1.895	**1.838**	1.695	**1.663**	1.575	**1.530**

Table 1: Average fertility scores reported across 17 languages (*Joint*). Here, NN and N represent non-normalized and normalized, respectively.

library. For variants of BPE, we use their respective implementation. The details of the hyper-parameter settings used are presented in Table 8.

5.1 Impact of Normalization

Consider the word in Hindi, अन्दर (andara, inside). We can find the same word written/typed with slight orthographic variations, for example: Using anusvāra ंद (ṁda) instead of the use of the corresponding fifth case consonant न्द (nda). Such variation could lead to the same word-form (aṁdara) appearing as two different tokens (aṁdara and andara) while building a vocabulary, causing redundancy and inefficient usage of vocabulary capacity. The orthographic variations in text is a very common occurrence across Indian languages, driven by diverse writing, and non-standardised typing practices, often leading to multiple surface forms for the same word.

To investigate the impact of Indic script-specific normalization, we trained tokenizers on both normalized and non-normalized corpora using the joint training approach. We apply a fifth case normalization rule to convert words with anusvāra (ं) into the corresponding nasal consonant for all languages, wherever applicable. This ensures that the training corpus is standardized and with fewer character variations. This normalization is consistent with the foundational phonetic principles outlined in the Aṣṭādhyāyī, according to which, the non-normalized way of writing is incorrect. This normalization also corresponds to the concept of nasal assimilation in linguistics.

Table 1 presents the average fertility scores across 17 languages for the tokenizers trained. For both 128k and 256k vocabulary sizes, tokenizers trained on normalized corpora consistently yielded lower fertility scores compared to non-normalized data. For instance, with a vocabulary size of 128k, the BPE achieves a fertility score of 1.664, compared to 1.717 on non-normalized data. For a tokenizer trained using the ULM normalized corpus, the model achieves a better fertility score of 1.663 as compared to 1.695 for the non-normalized corpus. Similar observations are made for tokenizers trained on a 256k vocabulary. The detailed fertility scores for non-normalized and normalized training corpora are presented in Table 12, Appendix D.1.

To understand the extent of language-script normalization, we estimated the proportion of words in a widely used dataset that undergo the normalization explained above. In the evaluation set of Flores-200 dev (Costa-Jussà et al., 2022) for languages we have considered, there are a total of 312,500 words, of which **4.36%** (13,621) words undergo changes with fifth-case normalization. Considering the percentage of unique words that undergo normalization changes, upto **6%** of total unique words undergo normalization. The language-wise statistics of the normalization effect are shown in Table 11.

Findings: *Normalization plays an important role in building a multilingual tokenizer for Indian languages, with language-specific rules–such as conversion of anusvāra into a nasal consonant form – improving tokenization quality.*

5.2 Vocabulary size

Figure 1 shows the percentage of vocabulary overlap for BPE and ULM tokenizers, for various scripts under consideration, across different vocabulary sizes. Interestingly, the percentage overlap increases consistently from 32k to 128k, but drops for the 256k vocabulary. This trend holds across all scripts considered. Similar trend is also observed for different tokenizer pairs (Added in Appendix Table 14).

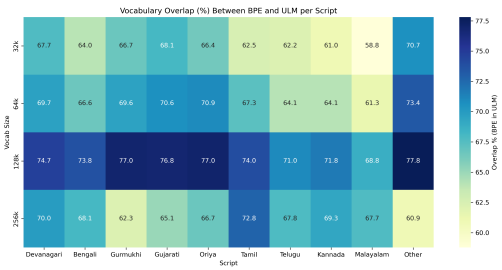


Figure 1: Overlap (%) between BPE and ULM vocabularies across scripts and vocabulary sizes.

We hypothesize that this change in trend occurs because 128k vocabulary captures most of the frequent tokens from the languages under consideration. After this point, the inclusion of additional tokens becomes increasingly arbitrary, which leads to a decrease in the overlap across independently trained tokenizers. Ideally, a robust multilingual tokenizer should produce a relatively consistent num-

ber of tokens for parallel sentences across different languages, although minor variations are expected due to linguistic differences. Considering this, we use the variance of the total token count of all languages as another evaluation metric to measure how consistently the tokenizer segments parallel content representing the same concept. Table 2 shows variance using 2 metrics *viz.,* Gini coefficient and normalized variance.

Algorithm	Vocab	Gini Coeff.	Normalized Variance
BPE	32k	0.039	0.071
	64k	0.034	0.063
	128k	0.034	0.063
	256k	0.042	0.073
ULM	32k	0.038	0.069
	64k	0.033	0.062
	128k	0.036	0.065
	256k	0.045	0.078

Table 2: Token count variance of BPE and ULM algorithm across various vocabulary sizes.

Findings: *Variance in number of tokens reduces or almost remains same upto 128k vocabulary size and then increases for 256k. The results also suggest that randomness in tokens added increases beyond 128k vocabulary size.*

5.3 Morphological alignment

A primary challenge in evaluating morphological alignment for the diverse set of Indic languages is the scarcity of high-quality, standardized datasets with morpheme-level segmentations. To conduct a meaningful analysis under these constraints, we adopt two sets of best available resources. For Hindi and Marathi, we leverage the rich, multi-morpheme dataset provided by Brahma et al. (2025), which reflects the complex morphology of Indian languages, by segmenting the words into prefix(es), lemma, and suffix(es). We use a MorphScore variant as described in Section 4.2 against this multi-segmented ground truth. To expand our analysis, we use the Gujarati and Tamil language dataset presented by Ali et al. (2024), based on binary segmentation of the words. While such a binary split may oversimplify complex word structures in morphologically rich languages, yet it provides a consistent ground truth valuable for assessing the relative performance between BPE and ULM.

Figure 2b shows the MorphScore for Hindi and Marathi data, which improves with the increase in vocabulary size. We also see that ULM consistently outperforms BPE in all four languages, which is in line with the findings by Bostrom and Durrett (2020). Figure 2a shows the MorphScore for Gujarati and Tamil data. In contrast to the trend shown with the multi-segmented data, here the morphscore decreases with the increase in vocabulary size. We hypothesize that this counter-intuitive result may be a consequence of the simplistic binary segmentation. A morphologically rich and agglutinated language like Tamil may have words made of multiple subwords fused together, for example, a word may be correctly split as m1+m2+m3. Since the dataset forces a single binary split, the ground truth may include m1+m2m3, ignoring the other valid morpheme boundary. With an increase in vocabulary size, the tokenizer's output may cater to more well-defined subwords, which doesn't match this binary split, causing the score to be lowered.

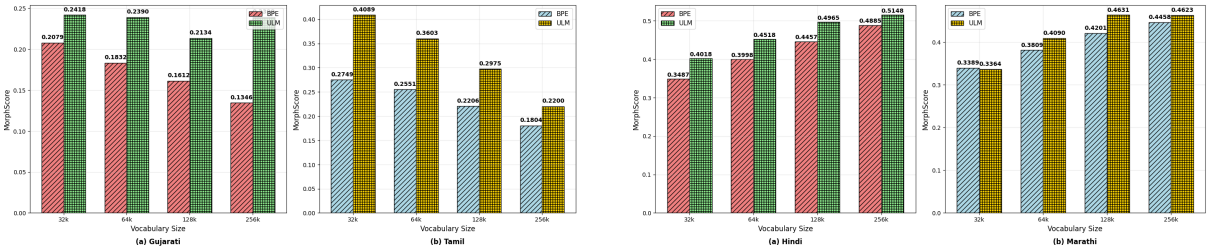
Findings: *Results suggest that ULM adheres more to the morphological segmentation compared to BPE, which is in line with the findings by Bostrom and Durrett (2020). We observe a reverse trend for Tamil and Gujarati with MorphScore decreasing as the vocabulary size increases. This trend is more prominent for ULM than for BPE. It may be due to the non-representative nature of the datasets for morphologically rich languages, where a word may be made of a number of subwords, whereas the words are split into only 2 segments in the dataset.*

5.4 Joint vs. Cluster

In the native script setting, the cluster variants (See Appendix C for details of clusters formed) achieve comparable or slightly lower fertility than the Joint method. The average fertility decreases marginally from 1.663 (Joint) to 1.650−1.658 across the Cluster methods, indicating slight improvements. Notably, Cluster_6 attains the lowest score among the cluster variants. Additionally, we observe that as the cluster size increases, the fertility increases.

In the ISO 15919 transliterated setting, fertility scores are substantially higher than native script. This is likely due to an increase in the sequence length of a word due to transliteration. Nonetheless, the cluster method achieves comparable or slight improvement compared to the Joint method, with Cluster_5 among all cluster variants achieving the lowest average fertility score of 2.988 compared to 3.06 for the Joint method. Overall, these results suggest that the cluster-based method provides modest but consistent gains over the Joint method.

To further investigate whether grouping related languages improves fertility, we manually



(a) MorphScore for Gujarati and Tamil.

(b) MorphScore for Hindi and Marathi.

Figure 2: Comparison of morphological alignment metrics across Indic languages. (a) MorphScore for Gujarati and Tamil with varying vocabulary sizes. (b) MorphScore for Hindi and Marathi.

constructed clusters based on language families (Cluster$_{lf}$). We notice that such a linguistically motivated clustering outperforms the automatically formed clusters for a majority of languages. Quantitatively, 13 out of 17 languages achieve lower fertility scores than the joint method, indicating that language family-based grouping facilitates more effective vocabulary learning. However, we observe that three out of four languages belonging to the Dravidian language family have higher fertility than the joint. Suggesting that there is a trade-off among languages in vocabulary allocation.

Findings: *This observation suggests (*c.f. *Figure 3) that cluster-based tokenizers better preserve language-specific subword units by mitigating the dominance of high-resource languages during vocabulary construction. Cluster-based vocabulary construction of multilingual tokenization has its own merit, with the reduction of the fertility and fairer splits as compared to the joint method. However, it seems to be sensitive to the formation of clusters, and careful consideration of languages per cluster is warranted.*

5.5 Extrinsic Evaluation

For extrinsic evaluation, we assess the impact of the tokenization strategy on Named Entity Recognition (NER) performance using an LSTM-based model trained with vocabularies of size 128k. The ULM-based model attains a slightly higher average macro F1 score across languages (72.3), compared to the BPE-based model (71.9), with particularly notable gains in Location (LOC) and Person (PER) entities. In contrast, the model trained using BPE yields marginally better performance on Organization (ORG) entities. Figure 3 summarizes these results, highlighting the consistent advantage of ULM across most entity types. We also observe that results are sensitive to vocabulary size (More details in Appendix E).

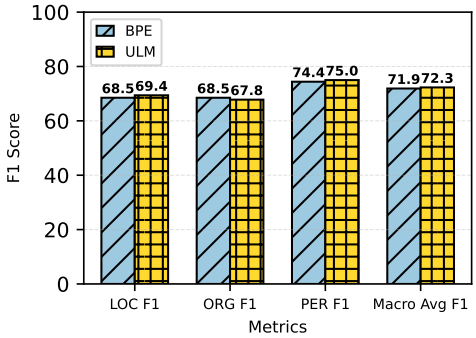


Figure 3: Comparison for BPE and ULM with vocabulary size of 128k trained to perform the NER task. Trained on a language-specific script normalized data.

6 Additional Analysis

6.1 Variant of BPE

To investigate the effect of recent variants of the BPE algorithm, we consider two algorithms, PickyBPE (Chizhov et al., 2024) and OBPE (Patil et al., 2022).

BPE vs. PickyBPE: Table 4 shows that PickyBPE (PBPE) achieves better fertility and CPT scores compared to BPE. Suggesting that PickyBPE, though designed to reduce vocabulary bloating in the tokenizer, shows that it reduces fertility and increases characters per token.

BPE vs. OBPE: Table 5 compares the BPE and Overlap BPE (OBPE) across six Indian languages with a vocabulary size of 64k[5] in the Devanagari script. We evaluate the fertility on the FLORES-200 dev set (Costa-Jussà et al., 2022). This highlights the limitations of the BPE algorithm in multilingual settings, while OBPE overcome by generating a vocabulary that maximizes transfer from HRLs to related LRLs through token overlap be-

[5] Due to computational cost, we perform experiments on six languages with 64k vocabulary size. OBPE takes substantially higher training time than BPE.

Lang.	Native Script						ISO 15919 Script					
	Joint	Cluster$_5$	Cluster$_6$	Cluster$_7$	Cluster$_8$	Cluster$_{lf}$	Joint	Cluster$_5$	Cluster$_6$	Cluster$_7$	Cluster$_8$	Cluster$_{lf}$
asm	1.742	1.751	1.764	1.757	1.751	**1.710**	2.930	2.935	2.935	2.935	2.935	2.931
ben	1.489	1.496	1.506	1.495	1.496	**1.466**	2.070	2.070	2.070	2.070	2.070	**2.066**
gom	1.813	1.821	1.830	1.843	1.843	**1.789**	4.730	**4.683**	**4.702**	**4.692**	4.705	**4.528**
guj	1.570	1.574	**1.562**	1.578	1.610	**1.540**	2.110	2.118	2.118	2.999	2.999	2.114
hin	1.300	1.317	1.310	1.320	1.319	**1.283**	3.370	3.410	3.423	3.420	3.426	**3.233**
kan	2.225	**1.990**	**1.968**	**1.989**	**1.982**	2.126	2.140	**2.139**	**2.139**	**2.139**	2.140	**2.138**
mai	1.400	**1.386**	**1.388**	**1.392**	**1.393**	**1.385**	3.640	**3.622**	**3.636**	**3.634**	3.643	**3.481**
mal	2.240	**2.165**	**2.165**	**2.165**	**2.166**	2.356	2.160	2.163	2.163	2.163	2.163	**2.158**
mar	1.574	1.598	1.600	1.611	1.610	**1.549**	4.670	**4.491**	**4.516**	**4.511**	4.524	**4.432**
nep	1.560	1.573	1.574	1.585	1.585	**1.528**	4.620	**4.282**	**4.299**	**4.292**	4.311	**4.441**
ori	1.675	**1.659**	**1.643**	**1.662**	**1.663**	**1.615**	2.060	2.061	2.062	2.062	2.062	2.061
pan	1.424	**1.419**	**1.409**	**1.420**	**1.421**	**1.392**	2.060	2.065	2.065	2.065	2.065	2.064
san	1.975	2.000	2.046	2.061	2.064	**1.954**	4.850	**4.508**	**4.508**	**4.505**	4.522	**4.573**
snd	1.385	**1.383**	**1.381**	**1.377**	**1.379**	**1.366**	3.350	**3.174**	**3.174**	**3.222**	3.361	3.228
tam	1.837	1.846	**1.831**	1.857	1.847	1.950	2.160	2.165	2.165	2.165	2.166	2.161
tel	1.869	1.874	**1.851**	1.876	1.870	1.990	2.170	2.182	2.182	2.181	2.181	2.174
urd	1.197	1.199	**1.194**	**1.192**	**1.193**	**1.182**	2.840	**2.765**	**2.765**	2.769	2.877	2.782
Avg.	**1.663**	**1.650**	**1.648**	**1.658**	**1.658**	**1.658**	**3.06**	**2.988**	**2.995**	**3.048**	**3.068**	**2.974**

Table 3: Comparison of **fertility** scores between Joint and Cluster method for 128k vocabulary size learned in **Native** and **ISO 15919** scripts. Highlighted scores indicate improvement over the joint method. Highlighted and underlined scores indicate the best scores.

Lang.	Fertility (↓)			CPT (↑)		
	BPE	PBPE	Δ_F	BPE	PBPE	Δ_{CPT}
asm	1.752	1.675	0.077	3.801	3.979	0.178
ben	1.503	1.451	0.052	4.455	4.617	0.162
gom	1.853	1.805	0.048	3.808	3.913	0.105
guj	1.595	1.528	0.067	3.817	3.986	0.169
hin	1.301	1.232	0.068	3.987	4.209	0.222
kan	2.034	1.984	0.051	4.271	4.380	0.109
mai	1.373	1.315	0.058	3.884	4.057	0.173
mal	2.241	2.186	0.055	4.431	4.542	0.111
mar	1.593	1.531	0.062	4.469	4.655	0.186
nep	1.557	1.495	0.061	4.386	4.566	0.181
ori	1.683	1.650	0.034	4.086	4.172	0.085
pan	1.433	1.356	0.077	3.671	3.833	0.211
san	1.994	1.969	0.025	3.762	3.813	0.051
snd	1.393	1.335	0.062	3.422	3.587	0.164
tam	1.882	1.814	0.068	4.841	5.022	0.181
tel	1.889	1.842	0.047	4.242	4.376	0.135
urd	1.207	1.179	0.028	3.665	3.751	0.086
Avg.	**1.664**	**1.609**	**0.055**	**4.059**	**4.209**	**0.148**

Table 4: Fertility and CPT scores across 17 languages compared for BPE and OBPE with vocabulary size of 128k. Δ_F = Fertility BPE − Fertility PBPE. Δ_{CPT} = CPT PBPE - CPT BPE.

tween LRLs and related HRLs, and maintains compact corpus representation.

Algorithm	Languages						Avg.
	gom	hin	mai	mar	nep	san	
BPE	**1.739**	1.261	1.334	**1.520**	**1.493**	1.876	1.537
OBPE	1.747	**1.255**	**1.326**	1.521	1.494	**1.875**	**1.536**

Table 5: Fertility scores across six languages for OBPE and BPE with vocabulary size of 64k.

6.2 Effect of Cluster-based method in Morphological alignment

Table 6 compares the Joint and Cluster variants for morphological alignment. We observe that clustering languages by language family (Cluster$_{lf}$) yields slightly higher or comparable scores than the Joint method for Tamil, Hindi, and Marathi. Although the gains are modest, these results suggest that the cluster-based multilingual vocabulary construction respects morphology more than the joint method.

Lang.	MorphScore (↑)		
	Joint	Cluster$_6$	Cluster$_{lf}$
guj	**0.213**	**0.213**	0.208
tam	0.298	0.309	**0.355**
hin	0.497	0.480	**0.504**
mar	0.463	0.442	**0.465**

Table 6: Comparison of Joint vs. Cluster variants on morphological alignment using MorphScore.

7 Conclusion

In this work, we perform a comprehensive study on the tokenization quality, focusing on 17 Indic languages. We perform the experiments with BPE and ULM tokenizers, which are the most widely used algorithms in recent works, either in their base form or their variants. The different experiment settings involve the variations in vocabulary sizes, different strategies for merging vocabularies in multilingual settings, and the effect of text normalization. We perform both intrinsic evaluation (task-independent quality checks) and extrinsic evaluation of the tokenizers on a downstream task. Our findings offer insights and practical guidance for designing fair and effective multilingual tokenizers for underrepresented language families. While our focus is on Indic languages, the methodologies and insights are broadly applicable to other low-resource and morphologically complex language settings and contribute towards region-specific LLM programs like Ng et al. (2025); Gala et al. (2024).

Limitations

Our study is limited to a few languages from distinct typological families and writing scripts that are primarily spoken in India. A larger set of languages should be evaluated to understand the tokenization effect belonging to diverse language families. In this work, we evaluate only one downstream task, Named Entity Recognition, and evaluation on more diverse tasks is left as future work. While our evaluation focus on the performance of multilingual tokenizers using intrinsic metrics, the influences of cross-lingual transfers among languages remain unexplored (Hämmerl et al., 2025). Additionally, due to the lack of morphological alignment datasets, our analysis is restricted to four languages, underscoring the need for benchmark resources to better study the effect of tokenization algorithms on morphology.

A more comprehensive extrinsic evaluation of training multilingual models with controlled vocabulary sizes, varying parameter budgets, and different pretraining objectives would provide deeper insights into how tokenization design choices influence vocabulary's representational quality and downstream task performance.

Ethical Considerations

We do not identify any ethical risks based on this work.

We used generative AI for language editing purposes only, such as paraphrasing, spell-checking, and refining and polishing the author's original content.

References

Diana Abagyan, Alejandro R. Salamanca, Andres Felipe Cruz-Salinas, Kris Cao, Hangyu Lin, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. 2025. One tokenizer to rule them all: Emergent language plasticity via multilingual tokenizers. *Preprint*, arXiv:2506.10766.

Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, Charvi Jain, Alexander Weber, Lena Jurkschat, Hammam Abdelwahab, Chelsea John, Pedro Ortiz Suarez, Malte Ostendorff, Samuel Weinbach, Rafet Sifa, and 2 others. 2024. Tokenizer choice for LLM training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3907–3924, Mexico City, Mexico. Association for Computational Linguistics.

Catherine Arnett and Benjamin Bergen. 2025. Why do language models perform worse for morphologically complex languages? In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 6607–6623, Abu Dhabi, UAE. Association for Computational Linguistics.

Thomas Bauwens and Pieter Delobelle. 2024. BPE-knockout: Pruning pre-existing BPE tokenisers with backwards-compatible morphological semi-supervision. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5810–5832, Mexico City, Mexico. Association for Computational Linguistics.

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, Online. Association for Computational Linguistics.

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*.

Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. 2024. BPE gets picky: Efficient vocabulary refinement during tokenizer training. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16587–16604, Miami, Florida, USA. Association for Computational Linguistics.

Hyung Won Chung, Dan Garrette, Kiat Chuan Tan, and Jason Riesa. 2020. Improving multilingual models with language-clustered vocabularies. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4536–4546, Online. Association for Computational Linguistics.

Alexis Conneau and Guillaume Lample. 2019. Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32.

Marta R Costa-Jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, and 1 others. 2022. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*.

Sumanth Doddapaneni, Rahul Aralikatte, Gowtham Ramesh, Shreya Goyal, Mitesh M. Khapra, Anoop Kunchukuttan, and Pratyush Kumar. 2023. Towards leaving no Indic language behind: Building monolingual corpora, benchmark and models for Indic languages. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12402–12426, Toronto, Canada. Association for Computational Linguistics.

Jay Gala, Thanmay Jayakumar, Jaavid Aktar Husain, Mohammed Safi Ur Rahman Khan, Diptesh Kanojia, Ratish Puduppully, Mitesh M Khapra, Raj Dabre, Rudra Murthy, Anoop Kunchukuttan, and 1 others. 2024. Airavata: Introducing hindi instruction-tuned llm. *arXiv preprint arXiv:2401.15006*.

Katharina Hämmerl, Tomasz Limisiewicz, Jindřich Libovický, and Alexander Fraser. 2025. Beyond literal token overlap: Token alignability for multilinguality. 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 756–767, Albuquerque, New Mexico. Association for Computational Linguistics.

Mohammed Safi Ur Rahman Khan, Priyam Mehta, Ananth Sankar, Umashankar Kumaravelan, Sumanth Doddapaneni, Suriyaprasaad B, Varun G, Sparsh Jain, Anoop Kunchukuttan, Pratyush Kumar, Raj Dabre, and Mitesh M. Khapra. 2024. IndicLLMSuite: A blueprint for creating pre-training and fine-tuning datasets for Indian languages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15831–15879, Bangkok, Thailand. Association for Computational Linguistics.

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.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Anoop Kunchukuttan. 2020. The IndicNLP Library. `https://github.com/anoopkunchukuttan/indic_nlp_library/blob/master/docs/indicnlp.pdf`.

Haoran Lian, Yizhe Xiong, Jianwei Niu, Shasha Mo, Zhenpeng Su, Zijia Lin, Hui Chen, Jungong Han, and Guiguang Ding. 2025. Scaffold-bpe: Enhancing byte pair encoding for large language models with simple and effective scaffold token removal. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24539–24548.

Tomasz Limisiewicz, Jiří Balhar, and David Mareček. 2023. Tokenization impacts multilingual language modeling: Assessing vocabulary allocation and overlap across languages. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5661–5681, Toronto, Canada. Association for Computational Linguistics.

Raymond Ng, Thanh Ngan Nguyen, Yuli Huang, Ngee Chia Tai, Wai Yi Leong, Wei Qi Leong, Xianbin Yong, Jian Gang Ngui, Yosephine Susanto, Nicholas Cheng, Hamsawardhini Rengarajan, Peerat Limkonchotiwat, Adithya Venkatadri Hulagadri, Kok Wai Teng, Yeo Yeow Tong, Bryan Siow, Wei Yi Teo, Wayne Lau, Choon Meng Tan, and 12 others. 2025. Sea-lion: Southeast asian languages in one network. *Preprint*, arXiv:2504.05747.

Vaidehi Patil, Partha Talukdar, and Sunita Sarawagi. 2022. Overlap-based vocabulary generation improves cross-lingual transfer among related languages. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 219–233, Dublin, Ireland. 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.

Gowtham Ramesh, Sumanth Doddapaneni, Aravinth Bheemaraj, Mayank Jobanputra, Raghavan AK, Ajitesh Sharma, Sujit Sahoo, Harshita Diddee, Mahalakshmi J, Divyanshu Kakwani, Navneet Kumar, Aswin Pradeep, Srihari Nagaraj, Kumar Deepak, Vivek Raghavan, Anoop Kunchukuttan, Pratyush Kumar, and Mitesh Shantadevi Khapra. 2022. Samanantar: The largest publicly available parallel corpora collection for 11 Indic languages. *Transactions of the Association for Computational Linguistics*, 10:145–162.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 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 and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, Online. Association for Computational Linguistics.

Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization is more than compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 678–702, Miami, Florida, USA. 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.

Gianluca Vico and Jindřinch Libovickỳ. 2025. Conditional unigram tokenization with parallel data. *arXiv preprint arXiv:2507.07824*.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023. Tokenization and the noiseless channel. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5184–5207, Toronto, Canada. Association for Computational Linguistics.

A Tokenizer Training Corpus

We initially sampled data from the IndicCorpv2 corpus. However, upon manual inspection, we observed several language subsets included sentences in other languages and in different scripts. For instance, the Bodo corpus occasionally contained Hindi sentences. This is likely due to the quality of the filtering technique used to create the dataset. We thereafter decide to work on the Sangraha corpus (Ramesh et al., 2022).

Table 7 shows the detailed statistics of the tokenizer training corpus, totaling 39GB of data with 7.46M rows.

The equation for data sampling is presented below:

$$q_i = \frac{f_i^{\alpha}}{\sum_{j=1}^{N} f_j^{\alpha}} \qquad f_i = \frac{n_i}{\sum_{k=1}^{N} n_k}$$

Here, n_i denotes the number of sentences in language i, α is the temperature parameter, and q_i is the probability of sampling a sentence from that language.

B Tokenizer

In our experiments, we use *SentencePiece* library (Kudo and Richardson, 2018). The settings we used are listed in Table 8. The settings that are not presented in the Table 8 are set to their default values.

For PickyBPE, we use the implementation of `https://github.com/bauwenst/PickyBPE`, and for OBPE, we use `https://github.com/Vaidehi99/OBPE`. We set the threshold in PickyBPE to 0.9 as used in the original paper (Chizhov et al., 2024). Similarly, for OBPE, we use the default hyperparameter values. For transliteration of native Indic script to the ISO-15919 script, we used `https://www.aksharamukha.com/`.

C Clusters

The clusters formed for Native and ISO scripts are presented in Table 9 and 10, respectively. Cluster_5, Cluster_6, Cluster_7, Cluster_8 are formed automatically using Algorithm 1. Cluster_{lf} depicts a language grouping formed by manually grouping languages belonging to the same language family.

Non-Normalized	Normalized
गांधीनगर (gāṃdhīnagara)	गान्धीनगर (gāndhīnagara)
गंगातट (gaṃgātaṭa)	गङ्गातट (gaṅgātaṭa)
चंद्रवदन (caṃdravadana)	चन्द्रवदन (candravadana)
चंद्रवंशी (caṃdravaṃśī)	चन्द्रवंशी (candravaṃśī)
पंकजनयना (paṃkajanayanā)	पङ्कजनयना (paṅkajanayanā)
जगदंबा (jagadaṃbā)	जगदम्बा (jagadambā)
कुंभमेला (kuṃbhamelā)	कुम्भमेला (kumbhamelā)

Figure 4: Illustration of fifth-case normalization for Devanagari script. The left column shows non-normalized words containing variant Unicode characters and diacritics, while the right column presents their normalized equivalents after applying custom anusvāra rules. Bracketed text contains transliteration in ISO-15919 format for readability.

D Results

D.1 Normalization Effect

The detailed fertility scores for non-normalized and normalized training corpora for 32k, 64k, 128k, and 256k vocabulary are presented in Table 12.

D.2 BPE and ULM

Table 13 shows the performance of BPE and ULM across vocabulary size of 32k, 64k, 128k, and 256k.

D.3 Vocabulary size

The overlap between different pairs of tokenizers across vocabulary sizes are illustrated in Table 14. The overlap increases with increase in vocabulary size upto 128k, after which, we see a reduction in the overlap as seen in Section 14.

D.4 Joint vs. Cluster

Detailed Character per token (CPT) and Parity score are reported in Table 15 and 16 respectively.

Language	Code	LF	Script	# Rows (M)	# Filtered (M)	10% Sub-sampled (M)	# Training Corpus (M)
Hindi	hin	Indo-European	Devanagari	17.42	15.15	1.52	0.74
Assamese	asm	Indo-European	Assamese	0.33	0.28	0.03	0.22
Bengali	ben	Indo-European	Bengali	11.50	10.66	1.07	0.67
Konkani	gom	Indo-European	Devanagari	0.01	0.01	0.00	0.08
Gujarati	guj	Indo-European	Gujarati	3.97	3.57	0.36	0.48
Kannada	kan	Dravidian	Kannada	3.63	3.15	0.32	0.46
Maithili	mai	Indo-European	Devanagari	0.02	0.02	0.00	0.10
Malayalam	mal	Dravidian	Malayalam	6.37	5.99	0.60	0.56
Marathi	mar	Indo-European	Devanagari	5.87	4.99	0.50	0.53
Nepali	nep	Indo-European	Devanagari	8.59	8.37	0.84	0.62
Oriya	ori	Indo-European	Odia	2.00	1.90	0.19	0.40
Punjabi	pan	Indo-European	Gurmuki	1.74	1.50	0.15	0.37
Sanskrit	san	Indo-European	Devanagari	0.91	0.83	0.08	0.31
Sindhi	snd	Indo-European	Arabic	0.54	0.40	0.04	0.25
Tamil	tam	Dravidian	Tamil	7.83	6.47	0.65	0.57
Urdu	urd	Indo-European	Arabic	5.44	5.17	0.52	0.54
Telugu	tel	Dravidian	Telugu	7.08	6.10	0.61	0.56
Total							**7.46**

Table 7: Per-language statistics for tokenizer training data: number of raw and filtered rows, 10% sub-sampled entries, and final corpus sizes in millions (M).

Hyper-parameter	Value(s)
model_type	BPE | Unigram
vocab_size	32k | 64k | 128k | 256k
split_by_unicode_script	True
split_by_number	True
split_by_whitespace	True
split_digits	False
train_extremely_large_corpus	True

Table 8: SentencePiece settings used for training tokenizers. All other options or flags are the default values.

D.5 BPE vs OBPE

Table 18 presents the in-depth analysis of the BPE vs OBPE comparison of 17 Indic languages, transliterated to ISO format.

E Extrinsic Evaluation

E.1 BPE and ULM

Downstream evaluation on NER tasks is shown in Table 19. Results show that the Unigram Language Model for Location (LOC) and Person (PER) performs better than BPE in terms of F1 scores. However, for the Organization (ORG) entity type, BPE has slightly better scores.

E.2 BPE and OBPE

Table 20 shows the architecture details of the BiLSTM model used for the NER task. Table 21 present the scores on *ai4bharat/naamapadam* dataset comprises of 11 Indian languages. Table 17 further shows the correlation of metrics, referenced from Table 21.

Cluster	Language grouping	Size
Cluster_5	(mar), (san, urd, snd, ben, guj, tam, kan, asm, pan, tel, hin, ori), (mai), (nep), (gom)	5
Cluster_6	(mar), (ben, asm), (urd, mai, snd, guj, tam, kan, pan, tel, hin, ori), (nep), (gom), (san)	6
Cluster_7	(mar), (guj, tam, kan, asm, pan, tel, hin, ori), (mai), (nep), (gom), (san), (urd, snd, ben)	7
Cluster_8	(mar), (tam, kan, asm, pan, tel, hin, ori), (mai), (nep), (urd, snd, ben), (gom), (san), (guj)	8
Cluster_{lf}	(ben, ori, asm, hin, nep, urd, pan, san, guj, snd, mai, gom, mar), (kan, tel, mal, tam)	2

Table 9: Clusters formed using monolingual corpus in Native for all considered languages

Cluster	Language grouping	Size
Cluster_5	(ben, ori, asm), (kan, snd, hin, nep, urd, pan, tel, san), (mar), (gom, mal, mai, tam), (guj)	5
Cluster_6	(ben, asm), (kan, snd, hin, nep, urd, pan, tel, san), (mar), (gom, mal, mai, tam), (guj), (ori)	6
Cluster_7	(ben, asm), (kan, snd, hin, urd, pan, tel, san), (mar), (gom, mal, mai, tam), (guj), (ori), (nep)	7
Cluster_8	(ben), (kan, snd, hin, urd, pan, tel, san), (mar), (gom, mal, mai, tam), (guj), (ori), (nep), (asm)	8
Cluster_{lf}	(ben, ori, asm, hin, nep, urd, pan, san, guj, snd, mai, gom, mar), (kan, tel, mal, tam)	2

Table 10: Clusters formed using monolingual corpus in ISO 15919 for all considered languages

Language	# words	Normalized word count (%)	# unique words	Normalized Unique word count (%)
Hindi	24607	1123 (4.56)	6600	604 (9.15)
Assamese	18570	154 (0.83)	8320	118 (1.42)
Bengali	18756	103 (0.55)	7801	72 (0.92)
Konkani	25322	1417 (5.6)	6979	638 (9.14)
Gujarati	20080	1033 (5.14)	8701	697 (8.01)
Kannada	15430	773 (5.01)	9730	628 (6.45)
Maithili	23455	1196 (5.1)	6832	621 (9.09)
Malayalam	14377	283 (1.97)	9494	242 (2.55)
Marathi	18065	1937 (10.72)	9560	1264 (13.22)
Nepali	17847	93 (0.52)	7811	60 (0.77)
Oriya	18914	71 (0.38)	7480	40 (0.53)
Punjabi	24539	611 (2.49)	6668	260 (3.90)
Sanskrit	16671	480 (2.88)	9187	390 (4.25)
Sindhi	23345	876 (3.75)	9534	481 (5.05)
Tamil	16134	0 (0.0)	9147	0 (0.00)
Telugu	16388	3471 (21.18)	10421	1968 (18.88)
Total	**312500**	**13621 (4.36)**	**134265**	**8083 (6.02)**

Table 11: Effect of fifth-case normalization on Flores *devtest* data

Lang.	Algorithm	Vocab							
		32k		64k		128k		256k	
		NN	N	NN	N	NN	N	NN	N
hin	BPE	1.533	1.523	1.404	1.309	1.309	1.301	1.244	1.236
	ULM	1.517	1.509	1.394	1.387	1.303	1.296	1.257	1.252
mai	BPE	1.655	1.649	1.484	1.477	1.378	1.373	1.307	1.302
	ULM	1.681	1.674	1.524	1.518	1.401	1.399	1.320	1.317
mar	BPE	2.107	1.998	1.785	1.770	1.698	1.593	1.473	1.462
	ULM	1.963	1.956	1.746	1.733	1.573	1.566	1.482	1.472
npi	BPE	1.955	1.924	1.735	1.710	1.576	1.557	1.450	1.435
	ULM	1.921	1.895	1.717	1.697	1.565	1.549	1.470	1.457
gom	BPE	2.376	2.378	2.082	2.086	1.854	1.853	1.673	1.671
	ULM	2.337	2.342	2.068	2.064	1.815	1.813	1.685	1.684
san	BPE	2.446	2.428	2.206	2.180	2.011	1.994	1.862	1.845
	ULM	2.418	2.400	2.186	2.170	1.986	1.971	1.840	1.831
snd	BPE	2.327	2.347	2.182	2.191	2.029	2.028	1.905	1.908
	ULM	2.327	2.351	2.184	2.203	1.992	2.024	1.875	1.892
pan	BPE	1.829	1.825	1.595	1.590	1.435	1.433	1.331	1.329
	ULM	1.776	1.774	1.561	1.558	1.420	1.417	1.363	1.363
ben	BPE	1.937	1.935	1.692	1.689	1.506	1.503	1.390	1.387
	ULM	1.872	1.878	1.659	1.657	1.489	1.487	1.393	1.393
asm	BPE	2.285	2.278	1.992	1.988	1.757	1.752	1.620	1.598
	ULM	2.244	2.235	1.956	1.951	1.744	1.740	1.618	1.617
kan	BPE	2.761	2.745	2.367	2.358	2.048	2.034	1.800	1.788
	ULM	2.699	2.680	2.309	2.294	1.993	1.997	1.801	1.786
tel	BPE	2.580	2.571	2.205	2.201	1.897	1.889	1.681	1.676
	ULM	2.546	2.531	2.164	2.152	1.870	1.863	1.701	1.693
mal	BPE	3.075	3.052	2.645	2.616	2.280	2.241	1.995	1.957
	ULM	3.014	2.983	2.581	2.552	2.244	2.202	1.993	1.954
tam	BPE	2.488	2.485	2.158	2.156	1.884	1.882	1.691	1.690
	ULM	2.400	2.399	2.075	2.073	1.840	1.836	1.675	1.677
guj	BPE	2.067	2.067	1.798	1.797	1.599	1.595	1.457	1.455
	ULM	2.000	1.995	1.755	1.752	1.840	1.836	1.675	1.677
ori	BPE	2.284	2.264	1.960	1.942	1.703	1.683	1.534	1.515
	ULM	2.240	2.217	1.912	1.891	1.680	1.655	1.550	1.532
urd	BPE	1.619	1.474	1.453	1.321	1.330	1.207	1.251	1.136
	ULM	1.568	1.432	1.424	1.295	1.316	1.197	1.278	1.164

Table 12: Fertility scores comparison between normalized and non-normalized text. Here, NN and N represent Non-normalized and normalized, respectively.

Algorithm	Vocab											
	32k			64k			128k			256k		
	F (↓)	CPT (↑)	WFR (↓)	F (↓)	CPT (↑)	WFR (↓)	F (↓)	CPT (↑)	WFR (↓)	F (↓)	CPT (↑)	WFR (↓)
BPE	2.137	3.178	57.280	1.872	3.619	48.496	1.664	4.059	40.509	1.517	4.444	33.879
ULM	2.093	3.243	54.679	1.838	3.682	46.844	1.642	4.110	40.049	1.530	4.405	37.141

Table 13: Average fertility (F), character per token (CPT), and word fragmentation rate (WFR) across in a joint setting. Lower fertility and WFR indicate better segmentation quality. Higher CPT suggests tokens are more semantically meaningful and compact.

Algorithm	Vocabulary Overlap (%)			
	32k	64k	128k	256k
ULM (NN) vs BPE (NN)	65.40	68.19	74.32	68.35
ULM (N) vs BPE (N)	65.48	68.15	74.28	67.74
BPE (N) vs BPE (NN)	92.95	92.72	92.25	91.95
ULM (N) vs ULM (NN)	93.35	93.14	92.87	92.36

Table 14: Percentage of vocabulary overlap across tokenizers. Here, NN and N represent non-normalized and normalized text, respectively.

Lang.	Native Script						ISO 15919 Script					
	Joint	Cluster$_5$	Cluster$_6$	Cluster$_7$	Cluster$_8$	Cluster$_{lf}$	Joint	Cluster$_5$	Cluster$_6$	Cluster$_7$	Cluster$_8$	Cluster$_{lf}$
asm	3.823	3.803	3.774	3.790	3.802	3.893	2.270	2.269	2.269	2.269	2.269	2.272
ben	4.499	4.478	4.448	4.481	4.477	4.567	3.240	3.236	3.236	3.236	3.235	3.242
gom	3.893	3.876	3.855	3.829	3.829	3.944	1.490	1.507	1.501	1.504	1.500	1.559
guj	3.878	3.870	3.899	3.860	3.782	3.954	2.880	2.875	2.875	2.031	2.031	2.880
hin	3.990	3.938	3.960	3.931	3.931	4.043	1.540	1.521	1.515	1.516	1.514	1.640
kan	3.867	4.366	4.416	4.368	4.384	4.088	4.060	4.062	4.062	4.062	4.061	4.064
mai	3.809	3.848	3.842	3.832	3.829	3.849	1.460	1.472	1.467	1.467	1.464	1.532
mal	4.446	4.586	4.585	4.585	4.583	4.214	4.600	4.591	4.590	4.590	4.589	4.602
mar	4.525	4.455	4.448	4.419	4.420	4.594	1.520	1.585	1.576	1.578	1.573	1.606
nep	4.378	4.339	4.336	4.306	4.307	4.467	1.480	1.594	1.588	1.590	1.583	1.537
ori	4.112	4.146	4.186	4.139	4.137	4.260	3.340	3.337	3.337	3.337	3.336	3.338
pan	3.696	3.707	3.733	3.704	3.703	3.780	2.550	2.548	2.548	2.548	2.547	2.549
san	3.801	3.752	3.668	3.641	3.636	3.840	1.550	1.678	1.665	1.666	1.660	1.641
snd	3.454	3.459	3.465	3.475	3.470	3.502	1.430	1.507	1.507	1.485	1.423	1.482
tam	4.961	4.935	4.977	4.906	4.933	4.672	4.220	4.208	4.208	4.208	4.206	4.217
tel	4.304	4.277	4.329	4.271	4.285	4.027	3.690	3.674	3.674	3.675	3.674	3.686
urd	3.696	3.689	3.704	3.710	3.707	3.742	1.560	1.598	1.600	1.597	1.537	1.590
Avg.	**4.067**	**4.090**	**4.096**	**4.073**	**4.071**	**4.085**	**2.520**	**2.545**	**2.542**	**2.492**	**2.483**	**2.553**

Table 15: Comparison of **CPT** scores between Joint and Cluster methods for 128k vocab learned in **Native** and **ISO 15919** scripts.

Lang.	Native Script						ISO 15919 Script					
	Joint	Cluster$_5$	Cluster$_6$	Cluster$_7$	Cluster$_8$	Cluster$_{lf}$	Joint	Cluster$_5$	Cluster$_6$	Cluster$_7$	Cluster$_8$	Cluster$_{lf}$
asm	1.025	1.017	1.031	1.018	1.015	1.020	0.670	0.658	0.656	0.656	0.655	0.694
ben	0.884	0.877	0.889	0.875	0.876	0.883	0.470	0.469	0.468	0.468	0.467	0.494
gom	1.121	1.112	1.125	1.124	1.123	1.122	1.120	1.099	1.100	1.098	1.099	1.125
guj	0.994	0.984	0.982	0.984	1.005	0.989	0.520	0.514	0.512	0.726	0.725	0.541
hin	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
kan	1.086	0.957	0.951	0.955	0.951	1.050	0.40	0.399	0.397	0.398	0.397	0.421
mai	1.039	1.015	1.022	1.017	1.018	1.042	1.040	1.023	1.023	1.023	1.023	1.036
mal	1.017	0.977	0.983	0.976	0.976	1.083	0.380	0.377	0.375	0.375	0.375	0.397
mar	0.901	0.902	0.909	0.908	0.908	0.899	1.030	0.981	0.983	0.983	0.984	1.024
nep	0.880	0.877	0.883	0.882	0.882	0.874	1.010	0.924	0.925	0.924	0.926	1.013
ori	1.000	0.977	0.974	0.977	0.978	0.977	0.480	0.471	0.469	0.470	0.469	0.497
pan	1.101	1.084	1.083	1.083	1.083	1.091	0.620	0.610	0.608	0.608	0.607	0.644
san	1.044	1.044	1.075	1.075	1.077	1.047	0.990	0.901	0.905	0.905	0.907	0.972
snd	1.046	1.032	1.036	1.025	1.027	1.047	0.980	0.915	0.912	0.927	0.965	0.983
tam	0.937	0.929	0.927	0.934	0.928	1.008	0.430	0.424	0.422	0.423	0.422	0.447
tel	0.964	0.956	0.950	0.956	0.953	1.043	0.440	0.433	0.432	0.432	0.431	0.457
urd	1.095	1.083	1.086	1.076	1.077	1.097	1.000	0.966	0.961	0.963	0.999	1.025

Table 16: Comparison of **Parity** scores between Joint and Cluster methods for 128k vocab learned in **Native** and **ISO 15919** scripts. Here, parity scores are with respect to Hindi.

Extrinsic Metric	CPT		Fertility	
	BPE	OBPE	BPE	OBPE
Macro Avg F1	-0.0553048246	-0.0547950918	-0.2811328582	-0.1324214718

Table 17: Comparison of Spearman's rank correlation coefficient of the CPT scores between OBPE and BPE algorithms for Macro Avg. F1 scores computed for 11 languages.

Language	Algorithm	Fertility	CPT
asm	BPE	1.51	5.61
	OBPE	1.74	6.74
ben	BPE	1.40	5.49
	OBPE	1.55	7.30
hin	BPE	1.24	4.85
	OBPE	1.33	7.12
nep	BPE	1.47	6.25
	OBPE	1.63	7.05
ori	BPE	1.44	5.42
	OBPE	1.53	7.40
mai	BPE	1.31	5.05
	OBPE	1.37	7.36
urd	BPE	1.81	2.47
	OBPE	2.33	3.35
snd	BPE	2.65	1.84
	OBPE	2.78	2.91
san	BPE	1.83	4.59
	OBPE	2.11	5.65
mar	BPE	1.43	5.70
	OBPE	1.68	6.87
gom	BPE	1.71	6.21
	OBPE	1.95	5.89
guj	BPE	1.41	6.98
	OBPE	1.58	6.70
pan	BPE	1.40	4.79
	OBPE	1.43	6.90
tam	BPE	1.73	5.21
	OBPE	1.94	6.67
tel	BPE	1.67	4.97
	OBPE	1.88	6.41
kan	BPE	1.76	5.29
	OBPE	2.04	6.39
mal	BPE	1.93	5.19
	OBPE	2.20	6.30

Table 18: Fertility and CPT of multiple Indic languages for the tokenization algorithms BPE and OBPE.

Tokenizers	LOC F1	ORG F1	PER F1	Macro Avg F1
BPE	69.0	**67.7**	74.8	**72.9**
ULM	**69.6**	67.2	**76.8**	72.5

Table 19: NER results for BPE and ULM averaged across languages for 256k vocabulary.

Hyper-parameters	Value
Model Name	LSTMTagger
Architecture	Embedding layer $\rightarrow$ Bidirectional LSTM $\rightarrow$ Dropout $\rightarrow$ Linear classifier
Embedding Dimension	$d =$ embedding_dim
Hidden Dimension	$h =$ hidden_dim (per direction)
Sequence Encoder	1-layer BiLSTM, batch-first, hidden size h, dropout 0.3
Output Layer	Fully connected layer of size $2h \rightarrow$ num_labels
Padding	Padding index = `pad_idx`
Task	Named Entity Recognition (token-level classification)
Input	Tokenized text (IDs), length n
Output	Logits $\in R^{n \times \text{num_labels}}$

Table 20: Model card for the BiLSTM-based NER model (`LSTMTagger`).

Lang.	Tok.	Accuracy	Macro Avg			Weighted Avg		
			Precision	Recall	F1	Precision	Recall	F1
asm	BPE	**0.95**	**0.66**	**0.54**	**0.58**	**0.95**	**0.95**	**0.95**
	OBPE	0.94	0.53	0.38	0.43	0.92	0.94	0.93
ben	BPE	0.95	0.85	0.79	0.82	0.95	0.95	0.95
	OBPE	**0.95**	**0.85**	**0.8**	**0.83**	**0.95**	**0.95**	**0.95**
guj	BPE	0.93	0.78	**0.80**	**0.79**	0.93	0.93	0.93
	OBPE	**0.93**	**0.78**	0.78	0.78	**0.93**	**0.93**	**0.93**
hin	BPE	0.95	0.83	0.81	0.82	0.95	0.95	0.95
	OBPE	**0.95**	**0.85**	**0.82**	**0.83**	**0.95**	**0.95**	**0.95**
kan	BPE	**0.93**	**0.84**	0.76	0.79	0.93	**0.93**	0.92
	OBPE	0.92	0.83	**0.77**	**0.79**	**0.93**	0.92	**0.92**
mal	BPE	0.92	0.83	**0.73**	0.77	0.91	0.92	0.91
	OBPE	**0.92**	**0.85**	0.72	**0.77**	**0.91**	**0.92**	**0.91**
mar	BPE	0.93	0.79	0.77	0.78	0.93	0.93	0.93
	OBPE	**0.94**	**0.82**	**0.77**	**0.79**	**0.94**	**0.94**	**0.94**
ori	BPE	0.88	**0.57**	**0.36**	**0.42**	0.85	0.88	0.86
	OBPE	**0.88**	0.56	0.34	0.40	**0.85**	**0.88**	**0.86**
pan	BPE	0.93	**0.77**	**0.78**	**0.77**	0.93	0.93	0.93
	OBPE	**0.93**	0.76	0.75	0.76	**0.93**	**0.93**	**0.93**
tam	BPE	0.89	0.65	**0.57**	0.60	0.88	0.89	0.88
	OBPE	**0.89**	**0.67**	0.56	**0.61**	**0.88**	**0.89**	**0.88**
tel	BPE	0.93	**0.84**	**0.81**	**0.82**	0.93	0.93	0.93
	OBPE	**0.93**	0.83	0.78	0.81	**0.93**	**0.93**	**0.93**

Table 21: NER results of 11 Indic languages in the ISO script for the tokenization algorithms (OBPE, and BPE).