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

Anonymous ACL submission

Abstract

Tokenization plays a pivotal role in multilingual NLP. 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 intrinsic evaluation of tokenization strategies across 17 Indian languages. We quantify the trade-offs between bottom-up and top-down tokenizer algorithms (BPE and Unigram LM), effects of vocabulary sizes, and compare strategies of multilingual vocabulary construction such as joint and cluster-based training. We also show that extremely low-resource languages can benefit from tokenizers trained on related high-resource languages. Our study provides practical insights for building more fair, efficient, and linguistically informed tokenizers for multilingual NLP.

1 Introduction

Tokenization is the process of segmenting raw text into smaller units/tokens (words, subwords, characters, etc.) which can help 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, training speed, especially in multilingual settings. This step further influences how well a model understands the linguistic structure and semantics of the input and how well it handles the vocabulary coverage. Most of the widely used tokenizers are designed primarily based on English because of the large-scale data availability and research conducted on English. 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. 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 Rust et al. (2021); Limisiewicz et al. (2023) have addressed tokenizer evaluation in multilingual contexts, they have largely overlooked Indic languages. To the best of our knowledge, this is the first large-scale intrinsic evaluation focusing on tokenization behavior across a typologically and script-wise diverse set of 17 Indian languages. Given the rich morphological and lexical characteristics of Indian languages and the script diversity, it's crucial to study how well the tokenizers are able to capture these characteristics effectively. In this work, we present different methods of tokenizer training and vocabulary building, with a focus on multilingual Indian languages, and perform various intrinsic evaluations to understand the tokenizer's ability to capture the above-mentioned characteristics of the languages. We particularly focus on the most widely used tokenizers *viz.,* Byte Pair Encoding (BPE) (Sennrich et al., 2016) and Unigram Language Model (Kudo, 2018)[1], with vocabulary size ranging from 32K to 256K.

Our contributions are: (1) we investigate the performance and impact of multilingual tokenization for 17 Indic languages from 2 language families,

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

viz., Indo-European and Dravidian, (2) analyse the impact of Indian language character normalization on the tokenizer efficiency, and (3) study the transfer capability of multilingual tokenizers on similar, but extremely low-resource languages.

2 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. To investigate the multilingual tokenization for Indic languages, we conduct an evaluation of various methods and algorithms.

2.1 Data

We utilize the Sangraha corpus (Khan et al., 2024), which offers higher-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 from Indo-European and Dravidian language families. 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 $\alpha = 0.3$. (Refer Appendix A for detailed statistics)

2.2 Approaches

To obtain multilingual tokenizers, we adopt two methods: joint training and cluster-based training.

2.2.1 Joint

In this method, the data for all languages is concatenated into a single corpus, and the tokenizer is trained on this combined data. This method is straightforward and widely used. However, it may disproportionally favor high-resource languages during training, leading to under-representation of tokens from low-resource languages.

2.2.2 Cluster

In cluster-based method (Chung et al., 2020), languages that are typologically or script-wise similar are grouped into clusters 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. (Refer Appendix B).

3 Experiments and Results

We train a total of ten tokenizers for 17 Indian languages using existing algorithms: BPE and ULM. To assess the quality of tokenization for each language individually, we use a parallel corpus comprising 997 sentences from the FLORES-200 dev set (NLLB Team, 2022). Recent work by Ali et al. (2024) highlights that the implementation of BPE varies across tokenization libraries such as Huggingface[2] and SentencePiece. Based on their finding[3], we train all tokenizers in our experiments using *SentencePiece* library. The details of the hyperparameter settings used are presented in Table 7.

Tokenization quality can be evaluated intrinsically or extrinsically. Intrinsic evaluation involves the metrics that can be applied directly to the tokenized output and are computed independently of downstream tasks. Whereas, extrinsic evaluation is the process of measuring the tokenizer's quality based on downstream tasks, which can be computationally expensive and may have conflating effects with model capacity and tasks considered. In this work, we focus on intrinsic evaluation methods, given the simplicity, speed of computation, generalizability, and coverage of a large number of languages. In addition, these metrics allow for early feedback for any underlying model because of their task-agnostic nature. Following are the intrinsic evaluation metrics considered in this study. (i) Fertility (Ali et al., 2024) (ii) Character Per Token (CPT) (Limisiewicz et al., 2023) (iii) IndicMorphScore (iv) Word Fragmentation Rate, (v) Parity Ratio (Petrov et al., 2023). [4]

3.1 Impact of Normalization

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 script-level normalization on the sampled corpus using the Indic-

[2] https://github.com/huggingface/tokenizers

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

[4] Considering the space limitation, we have added the definitions of each of the metrics in Appendix D.2

Algorithm	Vocab			
	128k		256k	
	NN	N	NN	N
BPE	1.717	1.701	1.568	1.552
ULM	1.695	1.680	1.575	1.563

Table 1: Average fertility scores reported across 17 languages in a joint setting. Here, NN and N represent Non-normalized and normalized corpora, respectively.

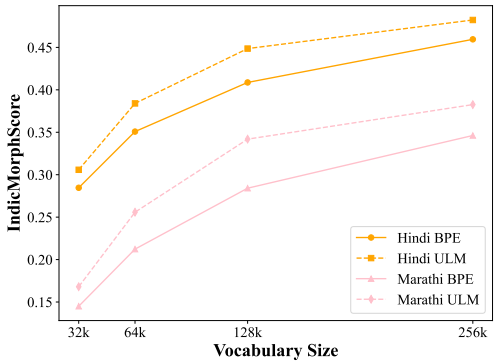

Figure 1: IndicMorphScore

NLP library (Kunchukuttan, 2020), which standardizes Unicode characters and diacritics across Indic scripts. Additionally, we apply a custom normalization rule to convert words with anusvāra into the corresponding nasal consonant for all languages. This ensures that the training corpus is standardized and with fewer character variations. Table 1 presents the average fertility scores across 17 languages for tokenizers trained using two subword segmentation algorithms: BPE and ULM. For both 128k and 256k vocabulary sizes, tokenizers trained on normalized corpora achieved lower fertility scores compared to non-normalized data. Detailed per-language fertility scores for 32k, 64k, 128k, and 256k vocabulary are provided in Table 8. 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.*

3.2 Vocabulary size

There exists a trade-off between monolingual and multilingual tokenizers as discussed in Section D.6. While monolingual tokenizers outperform multilingual tokenizers on intrinsic metrics such as fertility, characters per token, and average sequence length, multilingual tokenizers allow vocabulary from multiple languages, facilitating cross-lingual transfer. Increasing the vocab size in multilingual tokenizers from 32k to 256k achieves better scores in terms of fertility, characters per token, and fragmentation rate as reported in Table 2. However, a larger vocabulary comes with an added cost of computation during the modeling of large language models.

Variance in Token count

Table 3 shows the percentage of vocabulary overlap for different tokenizer pairs. Interestingly, the percentage overlap increases consistently from 32k to 128k, but then decreases for the 256k vocabulary. This trend is seen in all 4 cases. Our hypothesis for this change in trend is that, with the vocab size of 128k, the tokenizer captures most of the frequent tokens from all the languages under consideration, after which the inclusion of additional tokens becomes increasingly arbitrary, leading to a decrease in overlap across independently trained tokenizers. When we use a multilingual tokenizer to tokenize parallel sentences in different languages, ideally, the total number of tokens across the languages should be equal. Considering this idea, 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 10 show variance using 2 metrics *viz.,* Gini coefficient and normalized variance. Findings: *This section indicates the importance of carefully balancing tokenizer vocabulary size*.

3.3 Morphological alignment

Works like (Bostrom and Durrett, 2020; Arnett and Bergen, 2025) have studied BPE and ULM for morphological alignment. We use IndicMorphScore descibed in Section D.2 on two large-scale morphologically segmented datasets available for Hindi and Marathi (Brahma et al., 2025). The results of BPE and ULM for varying vocabulary size are illustrated in Figure 1. Finding: *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).

3.4 Joint vs. Cluster

We measure the parity and WFR for ULM tokenizers in joint and cluster settings. We observe that for Assamese, Bengali, Kannada, Malayalam, Oriya, Punjabi, Tamil, and Telugu trained using cluster grouping have lower WFR compared to joint settings. This is likely due to the fairer allocation

Algorithm	Vocab											
	32k			64k			128k			256k		
	F	CPT	WFR	F	CPT	WFR	F	CPT	WFR	F	CPT	WFR
BPE	2.173	3.153	58.756	1.910	3.574	50.250	1.701	4.017	42.305	1.552	4.398	35.770
ULM	2.132	3.214	56.397	1.879	3.642	48.880	1.680	4.066	42.067	1.563	4.365	38.973

Table 2: Average fertility (F), character per token (CPT), and word fragmentation rate (WFR) across in a joint setting. *Note:* Lower fertility and WFR indicate better segmentation quality. Higher CPT suggests tokens are more semantically meaningful and compact.

Algorithm	Vocabulary Overlap (in %) for vocab sizes			
	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 3: Percentage of vocabulary overlap across tokenizers.

of language-specific vocabulary units in the cluster method. Additionally, we observe lower parity scores for the cluster method. However, similar trends are not seen for a vocabulary size of 128k, suggesting that the vocabulary size affects the performance gaps of the joint and cluster methods. The detailed scores for all languages are reported in Appendix D.5. Findings: *cluster-based creation of multilingual tokenization has its own merit with reduction of the word fragmentation rate 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.*

Lang.	Method			
	Joint		Cluster	
	Parity	WFR	Parity	WFR
asm	1.027	44.267	0.916	37.489
ben	0.886	32.379	0.802	27.143
kan	0.966	55.078	0.843	47.469
mal	1.003	62.834	0.868	55.426
ory	0.990	38.990	0.843	29.886
pan	1.100	27.223	1.005	22.660
tam	0.940	51.460	0.841	45.442
tel	0.966	48.864	0.848	41.677

Table 4: Parity and WFR for a joint and cluster setting. Results reported are for the ULM algorithm for 256k vocab size. The parity scores are reported with respect to Hindi.

3.5 Lexically similar languages

There are many languages that can be categorized as extremely low-resource and do not have sufficient data to train a tokenizer effectively. In this section, we investigate whether tokenizers trained on high-resource languages, which either belong to the same language family or share a large vocabulary with low-resource languages, can transfer the tokenization ability to segment these low-resource languages efficiently. The detailed scores are presented under the Appendix section (Table 13) in a zero-shot setting, using a tokenizer trained on all 17 languages considered in this paper. We apply the tokenizers on low-resource languages *viz.,* Awadhi, Bhojpuri, Chhattisgarhi, and Magahi. We observe that tokenizers trained on related Indo-European languages perform reasonably well on these low-resource languages in terms of fertility and CPT, indicating promising transfer potential in zero-shot settings.

4 Conclusion and Future Work

In this work, we focus on the intrinsic evaluation of tokenizers for 17 Indic languages, considering the tokenization algorithms: BPE and Unigram language model and combining methods, such as joint and cluster. The goal of this work is specifically focused on providing insights for multilingual tokenizers for Indic languages.

Our findings offer practical guidance for designing fair and effective multilingual tokenizers for underrepresented language families. While our focus is on Indian languages, the methodologies and insights are broadly applicable to other low-resource, morphologically complex language settings and toward region-specific LLM programs (Ng et al., 2025; Gala et al., 2024). Future work will involve extrinsic evaluations and deeper exploration of tokenizer impact on downstream multilingual LLM performance. Determining the optimal vocabulary size that balances tokenization quality and computational efficiency is also left as future work. Furthermore, exploring the correlation between vocabulary size and extrinsic downstream performance would provide valuable insights.

Limitations

While our evaluation focused on the performance of multilingual tokenizers using intrinsic metrics, the influences of cross-lingual transfers among languages remain unexplored. A comprehensive extrinsic evaluation by training multilingual language models of varying model parameters is necessary to understand the various tokenizer performances in downstream tasks.

Ethics Statement

We do not foresee any ethical concerns with this work.

References

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.

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*.

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.

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*.

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`.

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.

James Cross Onur Çelebi Maha Elbayad Kenneth Heafield Kevin Heffernan Elahe Kalbassi Janice Lam Daniel Licht Jean Maillard Anna Sun Skyler Wang Guillaume Wenzek Al Youngblood Bapi Akula Loic Barrault Gabriel Mejia Gonzalez Prangthip Hansanti John Hoffman Semarley Jarrett Kaushik Ram Sadagopan Dirk Rowe Shannon Spruit Chau

Tran Pierre Andrews Necip Fazil Ayan Shruti Bhosale Sergey Edunov Angela Fan Cynthia Gao Vedanuj Goswami Francisco Guzmán Philipp Koehn Alexandre Mourachko Christophe Ropers Safiyyah Saleem Holger Schwenk Jeff Wang NLLB Team, Marta R. Costa-jussà. 2022. No language left behind: Scaling human-centered machine translation.

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.

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.

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.

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

Table 5 shows the detailed statistics of the tokenizer training corpus, totaling 39GB of data with 7.46M rows.

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

Table 5: Per-language statistics for tokenizer training data: number of raw and filtered rows, 10% sub-sampled entries, and final corpus sizes in millions (M).

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, and q_i is the probability of sampling a sentence from that language.

B Language Clusters

To find clusters, we follow the Chung et al. (2020) method. We first train monolingual tokenizers for 17 languages using the Unigram Language Model and take the union of all the vocabularies U_v. We then create a language-specific vector by marking entries with 1 if the token is present in its vocabulary, else we mark it as 0. We then train a K-means clustering algorithm using the vector as input. The clusters are formed as presented in Table 6. We then train individual tokenizers for each cluster. Finally, we merge the tokenizers to get the final vocabulary.

Cluster	Languages
1	pan, tam, mal, kan, tel
2	gom, guj, san, mai, hin, mar, nep
3	urd, snd
4	ori
5	asm, ben

Table 6: Clusters formed

C Tokenizer

In our experiments, we use *SentencePiece* library (Kudo and Richardson, 2018). The settings we used are listed in Table 7. The settings that are not presented in the Table 7 are considered to their default values

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 7: SentencePiece settings we used for training our tokenizers. All other options or flags are the default values.

D Results

D.1 Normalization Effect

The detailed fertility scores for non-normalized and normalized training corpus for 32k, 64, 128k, and 256k vocabulary are presented in Table 8.

D.2 Evaluation Metrics

Following are the intrinsic evaluation metrics considered in this study.
Fertility: Average number of tokens per word. A better fertility score (lower value) is often considered a necessary condition for better tokenization (Ali et al., 2024).
Character Per Token (CPT): Measures the average number of characters per token. Higher CPT indicates longer and more meaningful tokens (Limisiewicz et al., 2023).
Morphological Alignment: To measure whether the generated tokens adheres to the morphological boundaries of a language, we use IndicMorphScore (Brahma et al., 2025), calculated as an average of the morphological correctness segments.
Parity Ratio (Petrov et al., 2023): Parity measures the fairness among tokenizers for equivalent sentences in different languages. To measure the parity ratio, we consider Hindi as the pivot, as it has the largest training data, i.e., we measure the parity ratio of each language with respect to Hindi. We use FLORES-200 devset for the parallel data.

D.3 MorphScore

We evaluate MorphScore for Gujarati and Tamil on the corpus presented by Ali et al. (2024). The scores are presented in Table 9 for both BPE and ULM on varying vocab sizes of 32k, 64k, 128k, and 256k. For Gujarati, we observe ULM to perform better than BPE. Similar observation is made of Tamil. However, surprisingly for Tamil, we observe a decrease in MorphScore as the vocabulary increases. We suggest that the results may not be representative of the actual morphological alignment for these languages. Reason: The dataset divides the words into exactly two segments, but morphologically rich Indian languages can have multiple meaningful subwords for a given word, which may include prefix(es), lemma and suffix(es). The dataset enforces a binary segmentation, which oversimplifies the rich morphological structure of Indic languages. For example, complex inflections and compound derivations are inadequately captured, leading to underestimated alignment scores. Hence we use the dataset provided by Brahma et al. (2025), with morphologically alligned word-splits, to calculate a variant of MorphScore viz., IndicMorphScore (reported in Section **??**).

D.4 Variance

The variance score for token count is presented in Table 10.

Lang. code	Algorithm	Vocab							
		32k		64k		128k		256k	
		NN	N	NN	N	NN	N	NN	N
hin	BPE	1.533	1.523	1.404	1.309	1.301	1.394	1.244	1.236
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	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
	UnigramLM	2.000	1.995	1.755	1.752	1.840	1.836	1.675	1.677
ory	BPE	2.284	2.264	1.960	1.942	1.703	1.683	1.534	1.515
	UnigramLM	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
	UnigramLM	1.568	1.432	1.424	1.295	1.316	1.197	1.278	1.164

Table 8: Fertility scores comparison between normalized and non-normalized text. Here, NN and N represent Non-normalized and normalized, respectively.

Lang.	BPE				ULM			
	32k	64k	128k	256k	32k	64k	128k	256k
Gujarati	0.0586	0.0797	0.0962	0.989	0.0751	0.1154	0.1291	0.1758
Tamil	0.2031	0.2059	0.1912	0.1578	0.3117	0.3020	0.2602	0.1957

Table 9: MorphScore results for Gujarati and Tamil.

D.5 Joint vs. Cluster

The joint and cluster for the remaining languages are reported in Table 11.

D.6 Monolingual verses Multilingual

To assess the impact of multilingual training on the tokenization quality of a language, we compare the Word Fragmentation Rate (WFR) using the segments tokenized by each language's monolingual tokenizer and multilingual tokenizers respectively. Monolingual tokenizers are trained on data from a single language while multilingual tokenizers are trained on a shared fixed vocabulary budget across

Lang.	BPE				ULM			
	32k	64k	128k	256k	32k	64k	128k	256k
Gini Coefficient	0.039	0.034	0.034	0.042	0.038	0.033	0.036	0.045
Normalized variance	0.071	0.063	0.063	0.073	0.069	0.062	0.065	0.078

Table 10: Token count variance

Lang.	Method			
	Joint		Cluster	
	Parity	WFR	Parity	WFR
gom	1.078	49.791	1.173	56.399
guj	0.972	34.447	1.013	39.124
hin	1.000	19.690	1.000	22.469
mar	0.875	36.363	0.917	42.596
nep	0.854	34.734	0.889	41.133
san	0.247	55.543	1.063	61.970

Table 11: Parity and WFR for a joint and cluster setting. The results reported are for the ULM algorithm with a vocab size of 256k. The parity scores are reported with respect to Hindi.

multiple languages. This forces the tokenizer to allocate vocabulary across multiple languages with diverse scripts, leading to a reduction in language-specific subword units.

We trained monolingual tokenizers for 32k and 64k vocab sizes with the ULM algorithm in the joint setting. We then measure the average WFR and the CPT (Refer Table 12). Monolingual tokenizer achieves a lower fragmentation rate and higher CPT compared to multilingual tokenizers. We observe a significantly high WFR for the multilingual tokenizer compared to monolingual ones.

Findings: *(i) There's an inherent trade-off between multilingual and monolingual tokenizers. Though Monolingual tokenizers require larger data requirement for training, they achieve a low WFR compared to multilingual tokenizers. (ii) increasing the vocabulary capacity of the multilingual tokenizers seems to reduce the gap.*

Tokenizers	Vocab			
	32k		64k	
	WFR	CPT	WFR	CPT
Multilingual	57.28	3.18	48.50	3.62
Monolingual	33.02	4.60	28.08	4.87

Table 12: Average WFR and CPT across 17 languages (Tokenizers trained in joint setting).

Lang.	32k			64k			128k			256k		
	Fertility	Parity	CPT	Fertility	Parity	CPT	Fertility	Parity	CPT	Fertility	Parity	CPT
awa	1.606	1.076	3.157	1.460	3.472	1.064	1.351	3.752	1.052	1.307	3.880	1.052
bho	1.758	1.172	2.890	1.593	3.190	1.155	1.457	3.487	1.129	1.394	3.646	1.116
hne	1.764	1.396	2.820	1.605	3.099	1.377	1.500	3.315	1.379	1.434	3.468	1.360
mag	1.695	1.107	3.019	1.523	3.360	1.080	1.395	3.668	1.059	1.345	3.806	1.055

Table 13: Zero-shot intrinsic evaluation of Awadhi (awa), Bhojpuri (bho), Chhattisgarhi (hne), and Magahi (mag) on multilingual tokenizer trained using ULM algorithm in a joint setting.